

Lecture Notes on Foundations of Cyber-Physical Systems

André Platzer

Carnegie Mellon University
Lecture 0

1 Overview

Cyber-physical systems (CPSs) combine cyber capabilities (computation and/or communication) with physical capabilities (motion or other physical processes). Cars, aircraft, and robots are prime examples, because they move physically in space in a way that is determined by discrete computerized control algorithms. Designing these algorithms to control CPSs is challenging due to their tight coupling with physical behavior. At the same time, it is vital that these algorithms be correct, since we rely on CPSs for safety-critical tasks like keeping aircraft from colliding. In this course we will strive to answer the fundamental question posed by Jeannette Wing:

“How can we provide people with cyber-physical systems they can bet their lives on?”

Students who successfully complete this course will:

- Understand the core principles behind CPSs.
- Develop models and controls.
- Identify safety specifications and critical properties of CPSs.
- Understand abstraction and system architectures.
- Learn how to design by invariant.
- Reason rigorously about CPS models.
- Verify CPS models of appropriate scale.

- Understand the semantics of a CPS model.
- Develop an intuition for operational effects.

The cornerstone of our course design are hybrid programs (HPs), which capture relevant dynamical aspects of CPSs in a simple programming language with a simple semantics. One important aspect of HPs is that they directly allow the programmer to refer to real-valued variables representing real quantities and specify their dynamics as part of the HP.

This course will give you the required skills to formally analyze the CPSs that are all around us – from power plants to pace makers and everything in between – so that when you contribute to the design of a CPS, you are able to understand important safety-critical aspects and feel confident designing and analyzing system models. It will provide an excellent foundation for students who seek industry positions and for students interested in pursuing research.

2 Course Materials

Detailed lecture notes, homework assignments, lab assignments and other course material are available on the course web page.¹ There also is an optional textbook:

- André Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

More information on the design of the undergraduate course *Foundations of Cyber-Physical Systems* can be found in the [Course Syllabus](http://symbolaris.com/course/15424-syllabus.pdf).²

¹<http://symbolaris.com/course/fcps14.html>

²<http://symbolaris.com/course/15424-syllabus.pdf>

3 Lectures

These course consists of the following sequence of lectures (lecture notes are hyper-linked here but can also be found directly from the course web page):

1. [Cyber-physical systems: introduction](#)
2. [Differential equations & domains](#)
3. [Choice & control](#)
4. [Safety & contracts](#)
5. [Dynamical systems & dynamic axioms](#)
6. [Truth & proof](#)
7. [Control loops & invariants](#)
8. [Events & responses](#)
9. [Reactions & delays](#)
10. [Differential equations & differential invariants](#)
11. [Differential equations & proofs](#)
12. [Ghosts & differential ghosts](#)
13. [Differential invariants & proof theory](#)
14. [Logical foundations & CPS](#)
15. [Differential & temporal logic](#)
16. [Differential & temporal proofs](#)
17. [Virtual substitution & real equations](#)
18. [Virtual substitution & real arithmetic](#)
19. [Verified models & verified CPS runtime validation](#)
20. [Hybrid systems & games](#)
21. [Winning strategies & regions](#)
22. [Winning & proving hybrid games](#)
23. [Game proofs & separations](#)
24. [Logical theory & completeness](#)

Lecture Notes on Differential Equations & Domains

André Platzer

Carnegie Mellon University
Lecture 2

1. Introduction

In [Lecture 1](#), we have learned about the characteristic features of *cyber-physical systems* (CPS): they combine cyber capabilities (computation and/or communication as well as control) with physical capabilities (motion or other physical processes). Cars, aircraft, and robots are prime examples, because they move physically in space in a way that is determined by discrete computerized control algorithms that are adjusting the actuators (e.g., brakes) based on sensor readings of the physical state. Designing these algorithms to control CPSs is challenging due to their tight coupling with physical behavior. At the same time, it is vital that these algorithms be correct, since we rely on CPSs for safety-critical tasks like keeping aircraft from colliding.

Note 1 (Significance of CPS safety). *How can we provide people with cyber-physical systems they can bet their lives on?*
– Jeannette Wing

Since CPS combine cyber and physical capabilities, we need to understand both to understand CPS. It is not enough to understand both capabilities only in isolation, though, because we also need to understand how the cyber and the physics work together, i.e. what happens when they interface and interact, because this is what CPSs are all about.

Note 2 (CPS). *Cyber-physical systems combine cyber capabilities with physical capabilities to solve problems that neither part could solve alone.*

You already have experience with models of computation and algorithms for the cyber part of CPS, because you have seen the use of programming languages for computer programming in previous courses. In CPS, we do not program computers, but

rather program CPSs instead. Hence, we program computers that interact with physics to achieve their goals. In this lecture, we study models of physics and the most elementary part of how they can interact with the cyber part. Physics by and large is obviously a deep subject. But for CPS one of the most fundamental models of physics is sufficient, that of ordinary differential equations.

While this lecture covers the most important parts of differential equations, it is not to be understood as doing complete diligence to the fascinating area of ordinary differential equations. The crucial part about differential equations that you need to get started with the course is an intuition about differential equations as well as an understanding of their precise meaning. This will be developed in today's lecture. Subsequently, we will be coming back to the topic of differential equations for a deeper understanding of differential equations and their proof principles a number of times at a later part of the course. The other important aspect that today's lecture develops is *first-order logic of real arithmetic* for the purpose of representing domains and domain constraints of differential equations.

You are advised to refer back to your differential equations course and follow the supplementary information¹ available on the course web page as needed during this course to refresh your knowledge of differential equations. We refer, e.g., to the book by Walter [Wal98] for details and proofs about differential equations. There is a lot of further background on differential equations in the literature [Har64, Rei71, EEHJ96].

These lecture notes are based on material on cyber-physical systems, hybrid programs, and logic [Pla12, Pla10, Pla08, Pla07]. Cyber-physical systems play an important role in numerous domains [PCA07, LS10, Alu11, LSC⁺12] with applications in cars [DGV96], aircraft [TPS98], robots [PKV09], and power plants [FKV04], chemical processes [RKR10, KGDB10], medical models [GBF⁺11, KAS⁺11, LSC⁺12], and even an importance for understanding biological systems [Tiw11].

More information about CPS can be found in [Pla10, Chapter 1]. Differential equations and domains are described in [Pla10, Chapter 2.2,2.3] in more detail.

The most important learning goals of this lecture are:

Modeling and Control: We develop an understanding of one core principle behind CPS: the case of continuous dynamics and differential equations with evolution domains as models for the physics part of CPS. We introduce first-order logic of real arithmetic as the modeling language for describing evolution domains of differential equations.

Computational Thinking: Both the significance of meaning and the descriptive power of differential equations will play key roles, foreshadowing many important aspects underlying the proper understanding of cyber-physical systems. We will also begin to learn to carefully distinguish between syntax (which is notation) and semantics (what carries meaning), a core principle for computer science that continues to be crucial for CPS.

¹<http://symbolaris.com/course/fcps14-resources.html>

CPS Skills: We develop an intuition for the continuous operational effects of CPS and devote significant attention to understanding the exact semantics of differential equations, which has some subtleties in store for us.

2. Differential Equations as Models of Continuous Physical Processes

Differential equations model processes in which the (state) variables of a system evolve continuously in time. A differential equation concisely describes how the system evolves over time. It describes how the variables change locally, so it, basically, indicates the direction in which the variables evolve at each point in space. Fig. 1 shows the respective directions in which the system evolves by a vector at each point and illustrates one solution as a curve in space which follows those vectors everywhere. Of course, the figure would be rather cluttered if we would literally try to indicate the vector at each and every point, of which there are uncountably infinitely many. But this is a shortcoming only of our illustration. Differential equations actually define such a vector for the direction of evolution at every point in space.

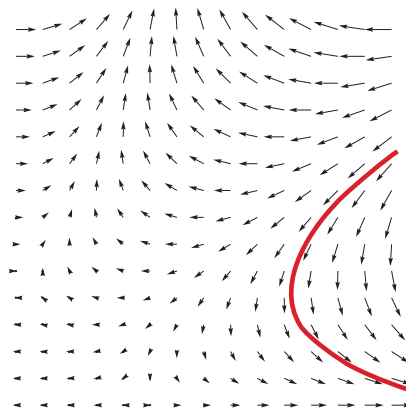


Figure 1: Vector field and one solution of a differential equation

As an example, suppose we have a car whose position is denoted by x . How the value of variable x changes over time depends on how fast the car is driving. Let v denote the velocity of the car. Since v is the velocity of the car, its position x changes such that its derivative x' is v , which we write by the differential equation $x' = v$. This differential equation is supposed to mean that the time-derivative of the position x is the velocity v . So how x evolves depends on v . If the velocity is $v = 0$, then the position x does not change at all. If $v > 0$, then the position x keeps on increasing over time. How fast x increases depends on the value of v , bigger v give quicker changes in x .

Of course, the velocity v , itself, may also be subject to change over time. The car might accelerate, so let a denote its acceleration. Then the velocity v changes with time-

derivative a , so $v' = a$. Overall, the car then follows the differential equation (system):²

$$x' = v, v' = a$$

That is, the position x of the car changes with time-derivative v , which, in turn, changes with time-derivative a .

What we mean by this differential equation, intuitively, is that the system has a vector field where all vectors point into direction a . What does this mean exactly?

3. The Meaning of Differential Equations

We can relate an intuitive concept to how differential equations describe the direction of the evolution of a system as a vector field Fig. 1. But what exactly is a vector field? What does it mean to describe directions of evolutions at every point in space? Could these directions not *possibly contradict each other so that the description becomes ambiguous*? What is the exact meaning of a differential equation in the first place?

The only way to truly understand any system is to understand exactly what each of its pieces does. CPSs are demanding and misunderstandings about their effect often have far-reaching consequences.

Note 3 (Importance of meaning). *The physical impacts of CPSs do not leave much room for failure, so we immediately want to get into the mood of consistently studying the behavior and exact meaning of all relevant aspects of CPS.*

An ordinary differential equation in explicit form is an equation $y'(t) = f(t, y)$ where $y'(t)$ is meant to be the derivative of y with respect to time t and f is a function of time t and current state y . A solution is a differentiable function Y which satisfies this equation when substituted into the differential equation, i.e., when substituting $Y(t)$ for y and the derivative $Y'(t)$ of Y at t for $y'(t)$.

² Note that the value of x changes over time, so it is really a function of time. Hence, the notation $x'(t) = v(t)$, $v'(t) = a$ is sometimes used. It is customary, however, to suppress the argument t for time and just write $x' = v$, $v' = a$ instead.

Definition 1 (Ordinary differential equation). Let $f : D \rightarrow \mathbb{R}^n$ be a function on a domain $D \subseteq \mathbb{R} \times \mathbb{R}^n$. The function $Y : I \rightarrow \mathbb{R}^n$ is a *solution* on the interval $I \subseteq \mathbb{R}$ of the *initial value problem*

$$\begin{bmatrix} y'(t) = f(t, y) \\ y(t_0) = y_0 \end{bmatrix} \quad (1)$$

with *ordinary differential equation* (ODE) $y' = f(t, y)$, if, for all $t \in I$

1. $(t, Y(t)) \in D$,
2. time-derivative $Y'(t)$ exists and is $Y'(t) = f(t, Y(t))$,
3. $Y(t_0) = y_0$, especially $t_0 \in I$.

If $f : D \rightarrow \mathbb{R}^n$ is continuous, then it is easy to see that $Y : I \rightarrow \mathbb{R}^n$ is continuously differentiable, because its derivative $Y'(t)$ is $f(t, Y(t))$. Similarly if f is k -times continuously differentiable then Y is $k + 1$ -times continuously differentiable. The definition is accordingly for higher-order differential equations, i.e., differential equations involving higher-order derivatives $y^{(n)}(t)$ for $n > 1$.

Let us consider the intuition for this definition. A differential equation (system) can be thought of as a vector field such as the one in Fig. 1, where, at each point, the vector shows in which direction the solution evolves. At every point, the vector would correspond to the right-hand side of the differential equation. A solution of a differential equation adheres to this vector field at every point, i.e., the solution (e.g., the solid curve in Fig. 1) locally follows the direction indicated by the vector of the right-hand side of the differential equation. There are many solutions of the differential equation corresponding to the vector field illustrated in Fig. 1. For the particular initial value problem, however, a solution also has to start at the prescribed position y_0 at time t_0 and then follow the differential equations or vector field from this point. In general, there could still be multiple solutions for the same initial value problem, but not for well-behaved differential equations (Appendix B).

Example 2. Some differential equations are easy to solve. The initial value problem

$$\begin{bmatrix} x'(t) = 5 \\ x(0) = 2 \end{bmatrix}$$

has a solution $x(t) = 5t + 2$. How could we verify that this is indeed a solution? This can be checked easily by inserting the solution into the differential equation and initial value equation:

$$\begin{bmatrix} (x(t))' = (5t + 2)' = 5 \\ x(0) = 5 \cdot 0 + 2 = 2 \end{bmatrix}$$

Example 3. Consider the initial value problem

$$\begin{bmatrix} x'(t) = -2x \\ x(1) = 3 \end{bmatrix}$$

which has a solution $x(t) = 3e^{-2(t-1)}$. The test, again, is to insert the solution into the (differential) equations of the initial value problems and check:

$$\left[\begin{array}{l} (3e^{-2(t-1)})' = -6e^{-2(t-1)} = -2x(t) \\ x(1) = 3e^{-2(1-1)} = 3 \end{array} \right]$$

Example 4 (Accelerated motion on a straight line). Consider the differential equation system $z' = v, v' = a$ and the initial value problem

$$\left[\begin{array}{l} z'(t) = v(t) \\ v'(t) = a \\ z(0) = z_0 \\ v(0) = v_0 \end{array} \right]$$

Note that this initial value problem is a *symbolic initial value problem* with symbols z_0, v_0 as initial values (not specific numbers like 5 and 2.3). Moreover, the differential equation has a constant symbol a , and not a specific number like 0.6, in the differential equation. In vectorial notation, the initial value problem with this differential equation system corresponds to a vectorial system when we denote $y(t) := (z(t), v(t))$, i.e., with dimension $n = 2$ in Def. 1:

$$\left[\begin{array}{l} y'(t) = \begin{pmatrix} z \\ v \end{pmatrix}'(t) = \begin{pmatrix} v(t) \\ a \end{pmatrix} \\ y(0) = \begin{pmatrix} z \\ v \end{pmatrix}(0) = \begin{pmatrix} z_0 \\ v_0 \end{pmatrix} \end{array} \right]$$

The solution of this initial value problem is

$$\begin{aligned} z(t) &= \frac{a}{2}t^2 + v_0t + z_0 \\ v(t) &= at + v_0 \end{aligned}$$

We can show that this is the solution by inserting the solution into the (differential) equations of the initial value problems and checking:

$$\left[\begin{array}{l} (\frac{a}{2}t^2 + v_0t + z_0)' = 2\frac{a}{2}t + v_0 = v(t) \\ (at + v_0)' = a \\ z(0) = \frac{a}{2}0^2 + v_00 + z_0 = z_0 \\ v(0) = a0 + v_0 = v_0 \end{array} \right]$$

Example 5. Consider the differential equation system $x' = y, y' = -x$ and the initial value problem

$$\left[\begin{array}{l} x'(t) = y(t) \\ y'(t) = -x(t) \\ x(0) = 1 \\ y(0) = 1 \end{array} \right]$$

The solution of this initial value problem is

$$\begin{aligned}x(t) &= \cos(t) + \sin(t) \\y(t) &= \cos(t) - \sin(t)\end{aligned}$$

We can show that this is the solution by inserting the solution into the (differential) equations of the initial value problems and checking:

$$\left[\begin{array}{l} (\cos(t) + \sin(t))' = -\sin(t) + \cos(t) = y(t) \\ (\cos(t) - \sin(t))' = -\sin(t) - \cos(t) = -x(t) \\ x(0) = \cos(0) + \sin(0) = 1 \\ y(0) = \cos(0) - \sin(0) = 1 \end{array} \right]$$

Example 6 (Time square oscillator). Consider the following differential equation system $x'(t) = t^2 y$, $y'(t) = -t^2 x$, which explicitly mentions the time variable t , and the initial value problem

$$\left[\begin{array}{l} x'(t) = t^2 y \\ y'(t) = -t^2 x \\ x(0) = 0 \\ y(0) = 1 \end{array} \right] \quad (2)$$

The solution shown in Fig. 2(left) illustrates that the system stays bounded but oscillates

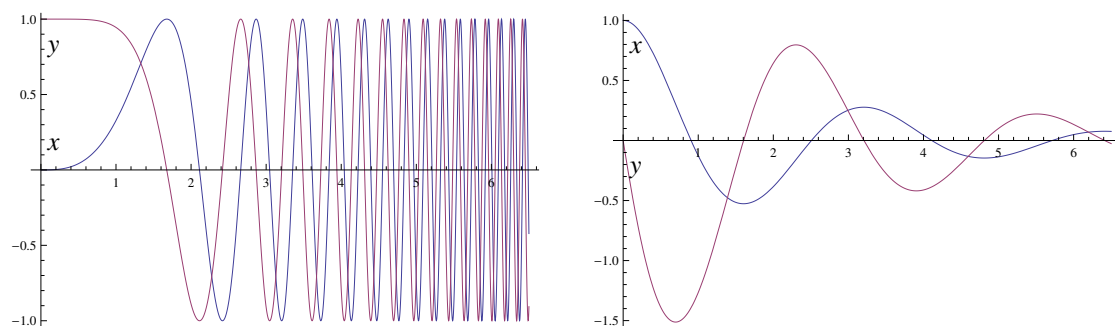


Figure 2: A solution of the time square oscillator (**left**) and of the damped oscillator (**right**) up to time 6.5

increasingly fast. In this case, the solution is

$$\left[\begin{array}{l} x(t) = \sin\left(\frac{t^3}{3}\right) \\ y(t) = \cos\left(\frac{t^3}{3}\right) \end{array} \right] \quad (3)$$

Example 7 (Damped oscillator). Consider the differential equation $x' = y$, $y' = -4x - 0.8y$ and the initial value problem

$$\left[\begin{array}{l} x'(t) = y \\ y'(t) = -4x - 0.8y \\ x(0) = 1 \\ y(0) = 0 \end{array} \right] \quad (4)$$

The solution shown in Fig. 2(right) illustrates that the dynamical system decays over time. In this case, the explicit global solution representing the dynamical system is more difficult.

Note 5 (Descriptive power of differential equations). *As a general phenomenon, observe that solutions of differential equations can be much more involved than the differential equations themselves, which is part of the representational and descriptive power of differential equations. Pretty simple differential equations can describe quite complicated physical processes.*

4. Domains of Differential Equations

Now we understand exactly what a differential equation is and how it describes a continuous physical process. In CPS, however, physical processes are not running in isolation but interact with cyber elements such as computers or embedded systems. When and how do physics and cyber elements interact? The first thing we need to understand for that is how to describe when physics stops so that the cyber elements take control of what happens next. Obviously, physics does not literally stop evolving, but rather keeps on evolving all the time. Yet, the cyber parts only take effect every now and then, because it only provides input into physics by way of its actuators every once in a while. So, our intuition may imagine physics “pauses” for a period of duration 0 and lets the cyber take action to influence the inputs that physics is based on.

The cyber and the physics could interface in more than one way. Physics might evolve and the cyber elements interrupt to inspect measurements about the state of the system periodically to decide what to do next. Or the physics might trigger certain conditions or events that cause cyber elements to compute their respective responses to these events. Another way to look at that is that a differential equation that a system follows forever without further intervention by anything would not describe a particularly well-controlled system. All those ways have in common that our model of physics needs to specify when it stops evolving to give cyber a chance to perform its task.

This information is what is called an *evolution domain* H of a differential equation, which describes a region that the system cannot leave while following that particular continuous mode of the system. If the system were ever about to leave this region, it would stop evolving right away (for the purpose of giving cyber parts of the system a chance to act) before it leaves the evolution domain.

Note 6 (Evolution domain constraints). A differential equation $x' = f(x)$ with evolution domain H is denoted by

$$x' = f(x) \& H$$

using a conjunctive notation ($\&$) between the differential equation and its evolution domain. This notation $x' = f(x) \& H$ signifies that the system obeys the differential equation $x' = f(x)$ and the evolution domain H . That is, the system follows this differential equation for any duration while inside the region H , but is never allowed to leave the region described by H . So the system evolution has to stop while the state is still in H .

If, e.g., t is a time variable with $t' = 1$, then $x' = v, v' = a, t' = 1 \& t \leq \varepsilon$ describes a system that follows the differential equation at most until time $t = \varepsilon$ and not any further, because the evolution domain $H \stackrel{\text{def}}{=} (t \leq \varepsilon)$ would be violated after time ε . That can be a useful model for physics that gives the cyber elements a chance to act at the latest at time ε . The evolution domain $H \stackrel{\text{def}}{=} (v \geq 0)$, instead, restricts the system $x' = v, v' = a \& v \geq 0$ to nonnegative velocities. Should the velocity ever become negative while following the differential equation $x' = v, v' = a$, then the system stops before that happens.

In the left two scenarios illustrated in Fig. 3, the system starts at time 0 inside the evolution domain H that is depicted as a shaded green region in Fig. 3. Then the system follows the differential equation $x' = f(x)$ for any period of time, but has to stop before it leaves H . Here, it stops at time r_0 (left) or r (middle, right) respectively.

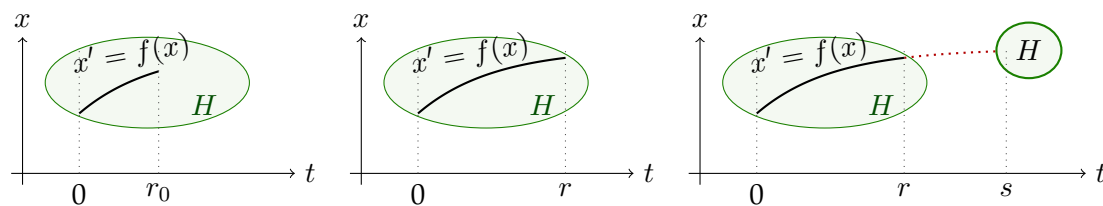


Figure 3: System $x' = f(x) \& H$ follows the differential equation $x' = f(x)$ for any duration r but cannot leave the (shaded) evolution domain H .

In contrast, consider the scenario shown on the right of Fig. 3. The system is *not* allowed to evolve until time s , because—even if the system were back in the evolution domain H at that time—it has already left the evolution domain H between time r and s (indicated by dotted lines), which is not allowed. Consequently, the continuous evolution on the right of Fig. 3 will also stop at time r at the latest and cannot continue any further.

Now that we know what the evolution domain constraint H of a differential equation is supposed to do, the question is how we can properly describe it in a CPS model? We will need some logic for that.

5. Continuous Programs: Syntax

After these preparations for understanding differential equations and domains, we start developing a programming language for cyber-physical systems. Ultimately, this programming language of *hybrid programs* will contain more features than just differential equations. But this most crucial feature is what we start with in this lecture. This course develops this programming language and its understanding and its analysis in layers one after the other. We will discuss the principles behind its design in the next lecture in more details and just start with continuous programs for now.

Continuous Programs. The first element of the syntax of hybrid programs are purely continuous programs.

Note 7. *Layer 1 of hybrid programs (HPs) are continuous programs. These are defined by the following grammar (α is a HP, x a variable, θ a term possibly containing x , and H a formula of first-order logic of real arithmetic):*

$$\alpha ::= x' = \theta \ \& \ H$$

This means that a hybrid program α consists of a single statement of the form $x' = \theta \ \& \ H$. In later lectures, we will add more statements to hybrid programs, but focus on differential equations for now. The formula H is called *evolution domain constraint* of the *continuous evolution* $x' = \theta \ \& \ H$. What form H can take will be defined below. Further x is a variable but is also allowed to be a vector of variables and, then, θ is a vector of terms of the same dimension. This corresponds to the case of differential equation systems such as:

$$x' = v, v' = a \ \& \ (v \geq 0 \wedge v \leq 10)$$

Differential equations are allowed without an evolution domain constraint H as well, for example:

$$x' = y, y' = x + y^2$$

which corresponds to choosing *true* for H , since the formula *true* is true everywhere and, thus, actually imposes no condition on the state whatsoever.

Terms. A rigorous definition of the syntax of hybrid programs also depends on defining what a term θ is and what a formula H of first-order logic of real arithmetic is.

Definition 8 (Terms). A *term* θ is a polynomial term defined by the grammar (where θ, η are terms, x a variable, and c a rational number constant):

$$\theta, \eta ::= x \mid c \mid \theta + \eta \mid \theta \cdot \eta$$

This means that a term θ (or a term η)³ is either a variable x , or a rational number constant $c \in \mathbb{Q}$, or a sum of terms θ, η , or a product of terms θ, η , which are again built of this form recursively. Subtraction $\theta - \eta$ is another useful case, but it turns out that it is already included, because the subtraction term $\theta - \eta$ is already definable by the term $\theta + (-1) \cdot \eta$. That is why we will not worry about subtraction, but use it in our examples regardless.

First-order Formulas. The formulas of first-order logic of real arithmetic are defined as usual in first-order logic, yet using the language of real arithmetic.

Definition 9 (Formulas of first-order logic of real arithmetic). The formulas of *first-order logic of real arithmetic* are defined by the following grammar (where F, G are formulas of first-order logic of real arithmetic, θ, η are terms, and x a variable):

$$F, G ::= \theta = \eta \mid \theta \geq \eta \mid \neg F \mid F \wedge G \mid F \vee G \mid F \rightarrow G \mid F \leftrightarrow G \mid \forall x F \mid \exists x F$$

The usual abbreviations are allowed, such as $\theta \leq \eta$ for $\eta \geq \theta$ and $\theta < \eta$ for $\neg(\theta \geq \eta)$.

6. Continuous Programs: Semantics

Note 10 (Syntax vs. Semantics). *Syntax just defines arbitrary notation. Its meaning is defined by the semantics.*

Terms. The meaning of a continuous evolution $x' = \theta \& H$ depends on understanding the meaning of terms θ . A term θ is a syntactic expression. Its value depends on the interpretation of the variables appearing in the term θ . What values those variables have changes depending on the state of the CPS. A *state* ν is a mapping from variables to real numbers. The set of states is denoted \mathcal{S} .

Definition 10 (Valuation of terms). The *value of term* θ in state ν is denoted $\llbracket \theta \rrbracket_\nu$ and defined by induction on the structure of θ :

$$\begin{aligned} \llbracket x \rrbracket_\nu &= \nu(x) && \text{if } x \text{ is a variable} \\ \llbracket c \rrbracket_\nu &= c && \text{if } c \in \mathbb{Q} \text{ is a rational constant} \\ \llbracket \theta + \eta \rrbracket_\nu &= \llbracket \theta \rrbracket_\nu + \llbracket \eta \rrbracket_\nu \\ \llbracket \theta \cdot \eta \rrbracket_\nu &= \llbracket \theta \rrbracket_\nu \cdot \llbracket \eta \rrbracket_\nu \end{aligned}$$

³ From a formal languages and grammar perspective, it would be fine to use the equivalent grammar

$$\theta ::= x \mid c \mid \theta + \theta \mid \theta \cdot \theta$$

We use the slightly more verbose form just to emphasize that a term can be a sum $\theta + \eta$ of any arbitrary and possibly different terms θ, η .

That is, the value of a variable x in state ν is defined by the state ν , which is a mapping from variables to real numbers. And the value of a term of the form $\theta + \eta$ in a state ν is the sum of the values of the subterms θ and η in ν , respectively. Likewise for the other forms that a term could take. Each term has a value in every state, because each case of the syntactic form of terms (Def. 8) has been given a semantics.

The value of a variable-free term like $4 + 5 \cdot 2$ does not depend on the state ν . In this case, the value is 14. The value of a term with variables, like $4 + x \cdot 2$, depends on ν . Suppose $\nu(x) = 5$, then $\llbracket 4 + x \cdot 2 \rrbracket_\nu = 14$. If $\omega(x) = 2$, then $\llbracket 4 + x \cdot 2 \rrbracket_\omega = 8$.

First-order Formulas. Unlike for terms, the value of a logical formula is not a real number but instead *true* or *false*. Whether a logical formula evaluates to *true* or *false* depends on the interpretation of its symbols. In first-order logic of real arithmetic, the meaning of all symbols except the variables is fixed. The meaning of terms and of formulas of first-order logic of real arithmetic is as usual in first-order logic, except that $+$ really means addition, \cdot means multiplication, \geq means greater or equals, and that the quantifiers $\forall x$ and $\exists x$ quantify over the reals. The meaning of the variables is again determined by the state of the CPS.

For the definition of the semantics, we need state modifications. Let ν_x^d denote the state that agrees with state ν except for the interpretation of variable x , which is changed to the value $d \in \mathbb{R}$:

$$\nu_x^d(y) = \begin{cases} d & \text{if } y \text{ is the variable } x \\ \nu(y) & \text{otherwise} \end{cases}$$

We write $\nu \models F$ to indicate that F evaluates to *true* in state ν and define it as follows.

Definition 11 (First-order logic semantics). The *satisfaction relation* $\nu \models F$ for a first-order formula F of real arithmetic in state ν is defined inductively:

- $\nu \models (\theta = \eta)$ iff $\llbracket \theta \rrbracket_\nu = \llbracket \eta \rrbracket_\nu$
That is, an equation is true in a state ν iff the terms on both sides evaluate to the same number.
- $\nu \models (\theta \geq \eta)$ iff $\llbracket \theta \rrbracket_\nu \geq \llbracket \eta \rrbracket_\nu$
That is, a greater-or-equals inequality is true in a state ν iff the term on the left evaluate to a number that is greater or equal to the value of the right term.
- $\nu \models \neg F$ iff $\nu \not\models F$, i.e. if it is not the case that $\nu \models F$
That is, a negated formula $\neg F$ is true in state ν iff the formula F itself is not true in ν .
- $\nu \models F \wedge G$ iff $\nu \models F$ and $\nu \models G$
That is, a conjunction is true in a state iff both conjuncts are true in said state.
- $\nu \models F \vee G$ iff $\nu \models F$ or $\nu \models G$
That is, a disjunction is true in a state iff either of its disjuncts is true in said state.
- $\nu \models F \rightarrow G$ iff $\nu \not\models F$ or $\nu \models G$
That is, an implication is true in a state iff either its left-hand side is false or its right-hand side true in said state.
- $\nu \models F \leftrightarrow G$ iff $(\nu \models F \text{ and } \nu \models G) \text{ or } (\nu \not\models F \text{ or } \nu \not\models G)$
That is, a biimplication is true in a state iff both sides are true or both sides are false in said state.
- $\nu \models \forall x F$ iff $\nu_x^d \models F$ for all $d \in \mathbb{R}$
That is, a universally quantified formula $\forall x F$ is true in a state iff its kernel F is true in all variations of the state, no matter what real number d the quantified variable x evaluates to in the variation ν_x^d .
- $\nu \models \exists x F$ iff $\nu_x^d \models F$ for some $d \in \mathbb{R}$
That is, an existentially quantified formula $\exists x F$ is true in a state iff its kernel F is true in some variation of the state, for a suitable real number d that the quantified variable x evaluates to in the variation ν_x^d .

If $\nu \models F$, then we say that F is true at ν or that ν is a model of F . A formula F is *valid*, written $\models F$, iff $\nu \models F$ for all states ν . A formula F is a *consequence* of a set of formulas Γ , written $\Gamma \models F$, iff, for each ν : $\nu \models G$ for all $G \in \Gamma$ implies that $\nu \models F$.

The most exciting formulas are the ones that are valid, i.e., $\models F$, because that means they are true no matter what state a system is in. Valid formulas, and how to find out

whether a formula is valid, will keep us busy quite a while in this course. Consequences of a formula set Γ are also amazing, because, even if they may not be valid per se, they are true whenever Γ is. For today's lecture, however, it is more important which formulas are true in a given state.

With the semantics, we know how to evaluate whether an evolution domain H of a continuous evolution $x' = \theta \ \& \ H$ is true in a particular state ν or not. If $\nu \models H$, then the evolution domain H holds in that state. Otherwise (i.e. if $\nu \not\models H$), H does not hold in ν . Yet, in which states ν do we even need to check the evolution domain? We need to find some way of saying that the evolution domain constraint H is checked for whether it is true (i.e. $\nu \models H$) in all states ν along the solution of the differential equation.

Continuous Programs. The semantics of continuous programs surely depends on the semantics of its pieces, which include terms and formulas. The latter have now been defined so that the next step is giving continuous programs themselves a proper semantics.

There is more than one way to define the meaning of a program, including defining a denotational semantics, an operational semantics, a structural operational semantics, an axiomatic semantics. We will be in a better position to appreciate several nuances of these aspects in later lectures. In order to keep things simple, all we care about for now is the observation that running a continuous program $x' = \theta \ \& \ H$ takes the system from an initial state ν to a new state ω . And, in fact, one crucial aspect to notice is that there is not only one state ω that $x' = \theta \ \& \ H$ can reach from ν just like there is not only one solution of the differential equation $x' = \theta$. Even in cases where there is a unique solution of maximal duration, there are still many different solutions differing only in the duration of the solution. Thus, the continuous program $x' = \theta \ \& \ H$ can lead from initial state ν to more than one possible state ω . Which states ω are reachable from an initial state ν along the continuous program $x' = \theta \ \& \ H$ exactly? Well these should be the states ω that can be connected from ν by a solution of the differential equation $x' = \theta$ that remains entirely within the set of states where the evolution domain constraint H holds true. Giving this a precise meaning requires going back and forth between syntax and semantics carefully.

Definition 12 (Semantics of continuous programs). The state ω is reachable from initial state ν by the continuous program $x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H$ iff there is a solution (or *flow*) φ of some duration $r \geq 0$ along $x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H$ from state ν to state ω , i.e. a function $\varphi : [0, r] \rightarrow \mathcal{S}$ such that:

- initial and final states match: $\varphi(0) = \nu, \varphi(r) = \omega$;
- φ respects the differential equations: For each variable x_i , the valuation $\llbracket x_i \rrbracket_{\varphi(\zeta)} = \varphi(\zeta)(x_i)$ of x_i at state $\varphi(\zeta)$ is continuous in ζ on $[0, r]$ and has a derivative of value $\llbracket \theta_i \rrbracket_{\varphi(\zeta)}$ at each time $\zeta \in (0, r)$, i.e.,

$$\frac{d\varphi(t)(x_i)}{dt}(\zeta) = \llbracket \theta_i \rrbracket_{\varphi(\zeta)}$$

- the value of other variables $z \notin \{x_1, \dots, x_n\}$ remains constant, that is, we have $\llbracket z \rrbracket_{\varphi(\zeta)} = \llbracket z \rrbracket_{\nu}$ for all $\zeta \in [0, r]$;
- and φ respects the evolution domain at all times: $\varphi(\zeta) \models H$ for each $\zeta \in [0, r]$.

The next lecture will introduce a notation for this and just write

$$(\nu, \omega) \in \rho(x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H)$$

to indicate that state ω is reachable from initial state ν by the continuous program $x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H$.

Observe that this definition is explicit about the fact that variables without differential equations do not change during a continuous program. The semantics of HP is *explicit change*: nothing changes unless (an assignment or) a differential equation specifies how. Also observe the explicit passing from syntax to semantics⁴ by the use of the valuation function $\llbracket \cdot \rrbracket$ in Def. 12.

Finally note that for duration $r = 0$, the condition on respecting the differential equation is trivially satisfied, because Def. 12 only requires the time-derivative of the value of x_i to match with its right-hand side θ_i in the open interval $(0, r)$, which, for $r = 0$, is the empty interval. Observe that this is a good choice for the semantics, because for $r = 0$, the meaning of a derivative at the only point in time 0 would not even be well-defined, so it would not be meaningful to refer to it. Consequently, the only conditions that Def. 12 imposes for duration 0 are that the initial state ν and final state ω are the same and that the evolution domain constraint H is respected at that state: $\nu \models H$.

⁴This important aspect is often overlooked. Informally, one might say that x obeys $x' = \theta$, but this certainly cannot mean that the equation $x' = \theta$ holds true, because it is not even clear what the meaning of x' would be, nor does θ have a single value, because it is a syntactic term whose value depends on the state by Def. 10. A syntactic variable x has a meaning in a state but x' does not. The semantical valuation of x along a function φ , instead, can have a well-defined derivative. This requires passing back and forth between syntax and semantics.

Note 14 (Operators and (informal) meaning in first-order logic of real arithmetic (FOL)).

FOL	Operator	Meaning
$\theta = \eta$	<i>equals</i>	<i>true iff values of θ and η are equal</i>
$\theta \geq \eta$	<i>equals</i>	<i>true iff value of θ greater-or-equal to η</i>
$\neg\phi$	<i>negation / not</i>	<i>true if ϕ is false</i>
$\phi \wedge \psi$	<i>conjunction / and</i>	<i>true if both ϕ and ψ are true</i>
$\phi \vee \psi$	<i>disjunction / or</i>	<i>true if ϕ is true or if ψ is true</i>
$\phi \rightarrow \psi$	<i>implication / implies</i>	<i>true if ϕ is false or ψ is true</i>
$\phi \leftrightarrow \psi$	<i>bi-implication / equivalent</i>	<i>true if ϕ and ψ are both true or both false</i>
$\forall x \phi$	<i>universal quantifier / for all</i>	<i>true if ϕ is true for all values of variable x</i>
$\exists x \phi$	<i>existential quantifier / exists</i>	<i>true if ϕ is true for some values of variable x</i>

7. Summary

This lecture gave a precise semantics to differential equations and presented first-order logic of real arithmetic, which we use for the evolution domain constraints within which differential equations are supposed to stay. The operators in first-order logic of real arithmetic and their informal meaning is summarized in Note 14.

A. Existence Theorems

For your reference, this appendix contains a short primer on some important results about differential equations [Pla10, Appendix B].

There are several classical theorems that guarantee existence and/or uniqueness of solutions of differential equations (not necessarily closed-form solutions with elementary functions, though). The existence theorem is due to Peano [Pea90]. A proof can be found in [Wal98, Theorem 10.IX].

Theorem 13 (Existence theorem of Peano). *Let $f : D \rightarrow \mathbb{R}^n$ be a continuous function on an open, connected domain $D \subseteq \mathbb{R} \times \mathbb{R}^n$. Then, the initial value problem (1) with $(t_0, y_0) \in D$ has a solution. Further, every solution of (1) can be continued arbitrarily close to the boundary of D .*

Peano's theorem only proves that a solution exists, not for what duration it exists. Still, it shows that every solution can be *continued arbitrarily close to the boundary* of the domain D . That is, the closure of the graph of the solution, when restricted to $[0, 0] \times \mathbb{R}^n$, is not a compact subset of D . In particular, there is a global solution on the interval $[0, \infty)$ if $D = \mathbb{R}^{n+1}$ then.

Peano's theorem shows the existence of solutions of continuous differential equations on open, connected domains, but there can still be multiple solutions.

Example 14. The initial value problem with the following continuous differential equation

$$\begin{bmatrix} y' &= \sqrt[3]{|y|} \\ y(0) &= 0 \end{bmatrix}$$

has multiple solutions:

$$\begin{aligned} y(t) &= 0 \\ y(t) &= \left(\frac{2}{3}t\right)^{\frac{3}{2}} \\ y(t) &= \begin{cases} 0 & \text{for } t \leq s \\ \left(\frac{2}{3}(t-s)\right)^{\frac{3}{2}} & \text{for } t > s \end{cases} \end{aligned}$$

where $s \geq 0$ is any nonnegative real number.

B. Existence and Uniqueness Theorems

As usual, $C^k(D, \mathbb{R}^n)$ denotes the space of k times continuously differentiable functions from domain D to \mathbb{R}^n .

If we know that the differential equation (its right-hand side) is continuously differentiable on an open, connected domain, then the Picard-Lindelöf theorem gives a stronger result than Peano's theorem. It shows that there is a unique solution (except, of course, that the restriction of any solution to a sub-interval is again a solution). For this, recall that a function $f : D \rightarrow \mathbb{R}^n$ with $D \subseteq \mathbb{R} \times \mathbb{R}^n$ is called *Lipschitz continuous* with respect to y iff there is an $L \in \mathbb{R}$ such that for all $(t, y), (t, \bar{y}) \in D$,

$$\|f(t, y) - f(t, \bar{y})\| \leq L\|y - \bar{y}\|.$$

If, for instance, $\frac{\partial f(t, y)}{\partial y}$ exists and is bounded on D , then f is Lipschitz continuous with $L = \max_{(t, y) \in D} \left\| \frac{\partial f(t, y)}{\partial y} \right\|$ by mean value theorem. Similarly, f is *locally Lipschitz continuous* iff for each $(t, y) \in D$, there is a neighbourhood in which f is Lipschitz continuous. In particular, if f is continuously differentiable, i.e., $f \in C^1(D, \mathbb{R}^n)$, then f is locally Lipschitz continuous.

Most importantly, Picard-Lindelöf's theorem [Lin94], which is also known as the Cauchy-Lipschitz theorem, guarantees existence and uniqueness of solutions. As restrictions of solutions are always solutions, we understand uniqueness up to restrictions. A proof can be found in [Wal98, Theorem 10.VI]

Theorem 15 (Uniqueness theorem of Picard-Lindelöf). *In addition to the assumptions of Theorem 13, let f be locally Lipschitz continuous with respect to y (for instance, $f \in C^1(D, \mathbb{R}^n)$ is sufficient). Then, there is a unique solution of the initial value problem (1).*

Picard-Lindelöf's theorem does not show the duration of the solution, but shows only that the solution is unique. Under the assumptions of Picard-Lindelöf's theorem,

every solution can be extended to a solution of maximal duration arbitrarily close to the boundary of D by Peano's theorem, however. The solution is unique, except that all restrictions of the solution to a sub-interval are also solutions.

Example 16. The initial value problem

$$\begin{bmatrix} y' &= y^2 \\ y(0) &= 1 \end{bmatrix}$$

has the unique maximal solution $y(t) = \frac{1}{1-t}$ on the domain $t < 1$. This solution cannot be extended to include the singularity at $t = 1$.

The following global uniqueness theorem shows a stronger property when the domain is $[0, a] \times \mathbb{R}^n$. It is a corollary to Theorems 13 and 15, but used prominently in the proof of Theorem 15, and is of independent interest. A direct proof of the following global version of the Picard-Lindelöf theorem can be found in [Wal98, Proposition 10.VII].

Corollary 17 (Global uniqueness theorem of Picard-Lindelöf). *Let $f : [0, a] \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ be a continuous function that is Lipschitz continuous with respect to y . Then, there is a unique solution of the initial value problem (1) on $[0, a]$.*

Exercises

Exercise 1. Subtraction $\theta - \eta$ is already included as a term, because it is definable. What about negation $-\theta$? What about division θ/η and powers θ^η ?

Exercise 2. Review the basic theory of ordinary differential equations and examples.

Exercise 3. Review the syntax and semantics of first-order logic.

*Exercise 4 (**).* What exactly would change and/or go wrong in which cases if Def. 12 were to demand that the derivative condition of the differential equation is respected at all times $\zeta \in [0, r]$ rather than at all times $\zeta \in (0, r)$ in an open interval?

References

- [Alu11] Rajeev Alur. Formal verification of hybrid systems. In Chakraborty et al. [CJBF11], pages 273–278.
- [CJBF11] Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors. *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*. ACM, 2011.
- [DGV96] Akash Deshpande, Aleks Göllü, and Pravin Varaiya. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems*, volume 1273 of *LNCS*, pages 113–133. Springer, 1996.

- [EEHJ96] Kenneth Eriksson, Donald Estep, Peter Hansbo, and Claes Johnson. *Computational Differential Equations*. Cambridge University Press, 1996.
- [FKV04] G. K. Fourlas, K. J. Kyriakopoulos, and C. D. Vournas. Hybrid systems modeling for power systems. *Circuits and Systems Magazine, IEEE*, 4(3):16–23, quarter 2004.
- [GBF⁺11] Radu Grosu, Grégory Batt, Flavio H. Fenton, James Glimm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. From cardiac cells to genetic regulatory networks. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 396–411. Springer, 2011. doi:[10.1007/978-3-642-22110-1_31](https://doi.org/10.1007/978-3-642-22110-1_31).
- [Har64] Philip Hartman. *Ordinary Differential Equations*. John Wiley, 1964.
- [KAS⁺11] BaekGyu Kim, Anaheed Ayoub, Oleg Sokolsky, Insup Lee, Paul L. Jones, Yi Zhang, and Raoul Praful Jetley. Safety-assured development of the gpca infusion pump software. In Chakraborty et al. [CJBF11], pages 155–164. doi:[10.1145/2038642.2038667](https://doi.org/10.1145/2038642.2038667).
- [KGDB10] Branko Kerkez, Steven D. Glaser, John A. Dracup, and Roger C. Bales. A hybrid system model of seasonal snowpack water balance. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 171–180. ACM, 2010. doi:[10.1145/1755952.1755977](https://doi.org/10.1145/1755952.1755977).
- [Lin94] M. Ernst Lindelöf. Sur l’application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre. *Comptes rendus hebdomadaires des séances de l’Académie des sciences*, 114:454–457, 1894.
- [LS10] Insup Lee and Oleg Sokolsky. Medical cyber physical systems. In Sachin S. Sapatnekar, editor, *DAC*, pages 743–748. ACM, 2010.
- [LSC⁺12] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyong Jee, BaekGyu Kim, Andrew L. King, Margaret Mullen-Fortino, Soojin Park, Alex Roederer, and Krishna K. Venkatasubramanian. Challenges and research directions in medical cyber-physical systems. *Proc. IEEE*, 100(1):75–90, 2012. doi:[10.1109/JPROC.2011.2165270](https://doi.org/10.1109/JPROC.2011.2165270).
- [PCA07] Leadership under challenge: Information technology R&D in a competitive world. an assessment of the federal networking and information technology R&D program. President’s Council of Advisors on Science and Technology, Aug 2007. http://www.ostp.gov/pdf/nitrd_review.pdf.
- [Pea90] Giuseppe Peano. Demonstration de l’intégrabilité des équations différentielles ordinaires. *Mathematische Annalen*, 37(2):182–228, 1890.
- [PKV09] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Form. Methods Syst. Des.*, 34(2):157–182, 2009.
- [Pla07] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007. doi:[10.1007/978-3-540-73099-6_17](https://doi.org/10.1007/978-3-540-73099-6_17).

- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. [doi:10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. [doi:10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. [doi:10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [Rei71] William T. Reid. *Ordinary Differential Equations*. John Wiley, 1971.
- [RKR10] Derek Riley, Xenofon Koutsoukos, and Kasandra Riley. Reachability analysis of stochastic hybrid systems: A biodiesel production system. *European Journal on Control*, 16(6):609–623, 2010.
- [Tiw11] Ashish Tiwari. Logic in software, dynamical and biological systems. In *LICS*, pages 9–10. IEEE Computer Society, 2011. [doi:10.1109/LICS.2011.20](https://doi.org/10.1109/LICS.2011.20).
- [TPS98] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE T. Automat. Contr.*, 43(4):509–521, 1998.
- [Wal98] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1998.

Lecture Notes on Choice & Control

André Platzer

Carnegie Mellon University
Lecture 3

1 Introduction

In the [Lecture 2 on Differential Equations & Domains](#), we have seen the beginning of cyber-physical systems, yet emphasized their continuous part in the form of differential equations $x' = \theta$. The sole interface between continuous physical capabilities and cyber capabilities was by way of their evolution domain. The evolution domain H in a continuous program $x' = \theta \ \& \ H$ imposes restrictions on how far or how long the system can evolve along that differential equation. Suppose a continuous evolution has succeeded and the system stops following its differential equation, e.g., because the state would otherwise leave the evolution domain if it had kept going. Then what happens now? How does the cyber take control? How do we describe what the cyber elements compute and how they interact with physics?

This lecture extends the model of continuous programs for continuous dynamics to the model of hybrid programs for hybrid dynamics.

This lecture is based on material on cyber-physical systems and hybrid programs [[Pla12b](#), [Pla10](#), [Pla08](#), [Pla07](#)].

Continuous programs $x' = \theta \ \& \ H$ are very powerful for modeling continuous processes. They cannot—on their own—model discrete changes of variables, however.¹ During the evolution along a differential equation, all variables change continuously in time, because the solution of a differential equation is (sufficiently) smooth. Discontinuous change of variables, instead, needs a way for a discrete change of state. What could be a model for describing discrete changes in a system?

¹There is a much deeper sense [[Pla12a](#)] in which continuous dynamics and discrete dynamics are quite surprisingly close together. That understanding requires a lot more logic than we have at our disposal at this stage of the course. It also leads to a full understanding of what constitutes the hybridness of hybrid systems. Yet, its understanding does rest on the foundations of hybrid systems, which we need to understand first.

There are many models for describing discrete change. You will have seen a number of them already. CPSs combine cyber and physics. In CPS, we do not program computers, but program CPSs instead. As part of that, we program the computers that control the physics. And programming computers amounts to using a programming language. So a programming language would give us a model of discrete computation. Of course, for programming an actual CPS, our programming language will ultimately have to involve physics. So, none of the conventional programming languages alone will work for CPS. But we have already seen continuous programs in the [previous lecture](#) for that very purpose. What's missing in continuous programs is a way to program the discrete and cyber aspects, which is exactly what the features of conventional programming languages provide.

Does it matter which discrete programming language we choose as a basis? It could be argued that the discrete programming language does not matter as much as the hybrid aspects do. After all, there are many programming languages that are Turing-equivalent, i.e. that compute the same functions (also see Church-Turing thesis). Yet even among all those conventional programming languages there are numerous differences for various purposes in the discrete case, which are studied in the area of Programming Languages.

For the particular purposes of CPS, however, we will find further desiderata, i.e. things that we expect from a programming language to be adequate for CPS. We will develop what we need as we go, culminating in the programming language of *hybrid programs* (HP).

More information about choice and control can be found in [\[Pla10, Chapter 2.2,2.3\]](#).

The most important learning goals of this lecture are:

Modeling and Control: We develop an understanding of the core principles behind CPS in the case of how discrete and continuous dynamics are combined and interact to model cyber and physics, respectively. We see the first example of how to develop models and controls for a (simplistic) CPS.

Computational Thinking: We introduce and study the important phenomenon of non-determinism, which is crucial for developing faithful models of a CPS's environment and helpful for developing efficient models of the CPS itself. We emphasize the importance of abstraction, which is an essential modular organization principle in CPS as well as all other parts of computer science. We capture the core aspects of CPS in a programming language, the language of hybrid programs.

CPS Skills: We develop an intuition for the operational effects of CPS. And we will develop an understanding for the semantics of the programming language of hybrid programs, which is the CPS model that this course is based on.

2 Discrete Programs and Sequential Composition

Discrete change happens in computer programs when they assign a new value to a variable. The statement $x := \theta$ assigns the value of term θ to variable x . It leads to

a discrete, discontinuous change, because the value of x does not vary smoothly but radically when suddenly assigning θ to x , which causes a discrete jump in the value of x .

This gives us a discrete model of change, $x := \theta$, in addition to the continuous model of change, $x' = \theta \ \& \ H$ from the [Lecture 2 on Differential Equations & Domains](#). Now, we can model systems that are *either* discrete *or* continuous. Yet, how can we model proper CPS that combine cyber and physics with one another and that, thus, simultaneously combine discrete and continuous dynamics? We need such hybrid behavior every time a system has both continuous dynamics (such as the continuous motion of a car down the street) and discrete dynamics (such as shifting gears).

One way how cyber and physics can interact is if a computer provides input to physics. Physics may mention a variable like a for acceleration and a computer program sets its value depending on whether the computer program wants to accelerate or brake. That is, cyber could set the values of actuators that affect physics.

In this case, cyber and physics interact in such a way that the cyber part first does something and physics then follows. Such a behavior corresponds to a sequential composition $(\alpha; \beta)$ in which first the HP α on the left of the sequential composition operator $;$ runs and, when it's done, the HP β on the right of $;$ runs. For example, the following HP²

$$a := a + 1; (x' = v, v' = a) \tag{1}$$

will first let cyber perform a discrete change of setting a to $a + 1$ and then let physics follow the differential equation $x'' = a$,³ which describes accelerated motion of point x along a straight line. The overall effect is that cyber increases the value of variable a and physics then lets x evolve with acceleration a (and increases velocity v continuously with derivative a). Thus, HP (1) models a situation where the desired acceleration is commanded once to increase and the robot then moves with that fixed acceleration. Note that the sequential composition operator $(;)$ has basically the same effect that it has in programming languages like Java or C0. It separates statements that are to be executed sequentially one after the other. If you look closely, however, you will find a subtle minor difference in that programming languages like Java and C0 expect more $;$ than hybrid programs, for example at the end of the last statement. This difference is inconsequential, and a common trait of mathematical programming languages.

The HP in (1) executes control (it sets the acceleration for physics), but it has very little choice. Actually no choice on what happens at all. So only if the CPS is very lucky will an increase in acceleration be the right action to remain safe forever. Quite likely, the robot will have to change its mind ultimately, which is what we will investigate next.

²Note that the parentheses around the differential equation are redundant and will often be left out in the lecture notes or in scientific papers. HP (1), for example, would be written $a := a + 1; x' = v, v' = a$. KeYmaera insists on more brackets, however.

³We will frequently use $x'' = a$ as an abbreviation for $x' = v, v' = a$, even if x'' is not officially permitted in KeYmaera.

3 Decisions in Hybrid Programs

In general, a CPS will have to check conditions on the state to see which action to take. One way of doing that is the use of an if-then-else, as in classical discrete programs.

$$\begin{aligned} &\text{if}(v < 4) a := a + 1 \text{ else } a := -b; \\ &x' = v, v' = a \end{aligned} \tag{2}$$

This HP will check the condition $v < 4$ to see if the current velocity is still less than 4. If it is, then a will be increased by 1. Otherwise, a will be set to $-b$ for some braking deceleration constant $b > 0$. Afterwards, i.e. when the if-then-else statement has run to completion, the HP will again evolve x with acceleration a along a differential equation.

The HP (2) takes only the current velocity into account to reach a decision on whether to accelerate or brake. That is usually not enough information to guarantee safety, because a robot doing that would be so fixated on achieving its desired speed that it would happily speed into any walls or other obstacles along the way. Consequently, programs that control robots also take other state information into account, for example the distance $x - o$ to an obstacle o from the robot's position x , not just its velocity v :

$$\begin{aligned} &\text{if}(x - o > 5) a := a + 1 \text{ else } a := -b; \\ &x' = v, v' = a \end{aligned} \tag{3}$$

They could also take both distance and velocity into account for the decision:

$$\begin{aligned} &\text{if}(x - o > 5 \wedge v < 4) a := a + 1 \text{ else } a := -b; \\ &x' = v, v' = a \end{aligned} \tag{4}$$

Note 1 (Iterative design). *As part of the labs of this course, you will develop increasingly more intelligent controllers for robots that face increasingly challenging environments. Designing controllers for robots or other CPS is a serious challenge. You will want to start with simple controllers for simple circumstances and only move on to more advanced challenges when you have fully understood and mastered the previous controllers, what behavior they guarantee and what functionality they are still missing.*

4 Choices in Hybrid Programs

What we learn from the above discussion is a common feature of CPS models: they often include only some but not all detail about the system. And for good reasons, because full detail about everything can be overwhelming and is often a distraction from the important aspects of a system. A (somewhat) more complete model of (4) might look as follows, with some further formula S as an extra condition for checking whether to actually accelerate:

$$\begin{aligned} &\text{if}(x - o > 5 \wedge v < 4 \wedge S) a := a + 1 \text{ else } a := -b; \\ &x' = v, v' = a \end{aligned} \tag{5}$$

The extra condition S may be very complicated and often depends on many factors. It could check to smooth the ride, optimize battery efficiency, or pursue secondary goals. Consequently, (4) is not actually a faithful model for (5), because (4) insists that the acceleration would always be increased just because $x - o > 5 \wedge v < 4$ holds, unlike (5), which also checks the additional condition S . Likewise, (3) certainly is no faithful model of (5). But it looks simpler.

How can we describe a model that is simpler than (5) by ignoring the details of S yet that is still faithful to the original system? What we want this model to do is characterize that the controller may either increase acceleration by 1 or brake. And we want that all acceleration certainly only happens when $x - o > 5$. But the model should make less commitment than (3) about the precise circumstances under which braking is chosen. So we want a model that allows braking under more circumstances than (3) without having to model precisely under which circumstances that is. In order to simplify the system faithfully, we want a model that allows more behavior than (3). The rationale is ultimately that if a system with more behavior is safe, the actual implementation will be safe as well, because it will only ever exercise some of the verified behavior.

Note 2 (Abstraction). *Successful CPS models often include only the relevant aspects of the system and simplify irrelevant detail. The benefit of doing so is that the model and its analysis becomes simpler, enabling us to focus on the critical parts without being bogged down in tangentials. This is the power of abstraction, arguably the primary secret weapon of computer science. It does take considerable skill, however, to find the best level of abstraction for a system. A skill that you will continue to sharpen through your entire career.*

Let us take the development of this model step by step. The first feature that the controller of the model has is a choice. The controller can choose to increase acceleration or to brake, instead. Such a choice between two actions is denoted by the operator \cup :

$$\begin{aligned} &(a := a + 1 \cup a := -b); \\ &x' = v, v' = a \end{aligned} \tag{6}$$

When running this hybrid program, the first thing that happens is that the first statement (before the $;$) runs, which is a choice (\cup) between whether to run $a := a + 1$ or whether to run $a := -b$. That is, the choice is whether to increase a by 1 or whether to reset a to $-b$ for braking. After this choice (i.e. after the $;$ sequential composition operator), the system follows the usual differential equation $x'' = a$ describing accelerated motion along a line.

Now, wait. There was a choice. Who chooses? How is the choice resolved?

Note 3 (Nondeterministic \cup). *The choice (\cup) is nondeterministic. That is, every time a choice $\alpha \cup \beta$ runs, exactly one of the two choices, α or β , is chosen to run and the choice is nondeterministic, i.e. there is no prior way of telling which of the two choices is going to be chosen. Both outcomes are perfectly possible.*

The HP (6) is a *faithful abstraction* of (5), because every way how (5) can run can be mimicked by (6) so that the outcome of (6) corresponds to that of (5). Whenever (5) runs $a := a + 1$, which happens exactly if $x - o > 5 \wedge v < 4 \wedge S$ is *true*, (6) only needs to choose to run the left choice $a := a + 1$. Whenever (5) runs $a := -b$, which happens exactly if $x - o > 5 \wedge v < 4 \wedge S$ is *false*, (6) needs to choose to run the right choice $a := -b$. So all runs of (5) are possible runs of (6). Furthermore, (6) is much simpler than (5), because it contains less detail. It does not mention $v < 4$ nor the complicated extra condition S . Yet, (6) is a little too permissive, because it suddenly allows the controller to choose $a := a + 1$ even at close distance to the obstacle, i.e. even if $x - o > 5$ is *false*. That way, even if (5) was a safe controller, (6) is still an unsafe one, and, thus, not a very suitable abstraction.

5 Tests in Hybrid Programs

In order to build a faithful and not too permissive model of (5), we need to restrict the permitted choices in (6) so that there's flexibility but only so much that the acceleration choice $a := a + 1$ can only be chosen at sufficient distance $x - o > 5$. The way to do that is to use tests on the current state of the system.

A test $?H$ is a statement that checks the value of a first-order formula H of real arithmetic in the current state. If H holds in the current state, then the test passes, nothing happens, yet the HP continues to run normally. If, instead, H does not hold in the current state, then the test fails, and the system execution is aborted and discarded. That is, when ν is the current state, then $?H$ runs successfully without changing the state when $\nu \models H$. Otherwise, i.e. if $\nu \not\models H$, the run of $?H$ is aborted and not considered any further, because it did not play by the rules of the system.

The test statement can be used to change (6) around so that it allows acceleration only at large distances while braking is still allowed always:

$$\begin{aligned} & ((?x - o > 5; a := a + 1) \cup a := -b); \\ & x' = v, v' = a \end{aligned} \tag{7}$$

The first statement of (7) is a choice (\cup) between $(?x - o > 5; a := a + 1)$ and $a := -b$. All choices in hybrid programs are nondeterministic so any outcome is always possible. In (7), this means that the left choice can always be chosen, just as well as the right one. The first statement that happens in the left choice, however, is the test $?x - o > 5$, which the system run has to pass in order to be able to continue successfully. In particular, if $x - o > 5$ is indeed *true* in the current state, then the system passes that test $?x - o > 5$ and the execution proceeds to after the sequential composition ($;$) to run $a := a + 1$. If $x - o > 5$ is *false* in the current state, however, the system fails the test $?x - o > 5$ and that run is aborted and discarded. The right option to brake is always available, because it does not involve any tests to pass.

Note 4 (Discarding failed runs). *System runs that fail tests are discarded and not considered any further, because that run did not play by the rules of the system. It is as if those failed system execution attempts had never happened. Yet, even if one execution attempt fails, other execution paths may still be successful. Operationally, you can imagine finding them by backtracking through all the choices in the system run and taking alternative choices instead.*

There are always two choices when running (7). Yet, which ones run successfully depends on the current state. If the current state is at a far distance from the obstacle ($x - o > 5$), then both options of accelerating and braking will indeed be possible and can be run successfully. Otherwise, only the braking choice runs without being discarded because of a failing test.

Comparing (7) with (5), we see that (7) is a faithful abstraction of the more complicated (5), because all runs of (5) can be mimicked by (7). Yet, unlike the intermediate guess (6), the improved HP (7) still retains the critical information that acceleration is only allowed by (5) at sufficient distance $x - o > 5$. Unlike (5), (7) does not restrict the cases where acceleration can be chosen to those that also satisfy $v < 4 \wedge S$. Hence, (7) is more permissive than (5). But (7) is also simpler and only contains crucial information about the controller. Hence, (7) is a more abstract faithful model of (5) that retains the relevant detail. Studying the abstract (7) instead of the more concrete (5) has the advantage that only relevant details need to be understood while irrelevant aspects can be ignored. It also has the additional advantage that a safety analysis of the more abstract (7), which allows lots of behavior, will imply safety of the special concrete case (5) but also implies safety of other implementations of (7). For example, replacing S by a different condition in (5) still gives a special case of (7). So if all behavior of (7) is safe, all behavior of that different replacement will also already be safe. With a single verification result about a more general, more abstract system, we can obtain verification for a whole class of systems rather than just one particular system. This important phenomenon will be investigated in more detail in later parts of the course.

Of course, which details are relevant and which ones can be simplified depends on the analysis question at hand, a question that we will be better equipped to answer in a later lecture. For now, suffice it to say that (7) has the relevant level of abstraction for our purposes.

Note 5 (Broader significance of nondeterminism). *Nondeterminism comes up in the above cases for reasons of abstraction and for focusing the system model on the most critical aspects of the system while suppressing irrelevant detail. This is an important reason for introducing nondeterminism in system models, but there are other important reasons as well. Whenever a system includes models of its environment, nondeterministic models are often a crucial idea, because there is often just a partial understanding of what the environment will do. A car controller for example, will not always know for sure what other cars in its environment will do, exactly.*

6 Repetitions in Hybrid Programs

The hybrid programs above were interesting, but only allowed the controller to choose what action to take at most once. All controllers so far inspected the state in a test or in an if-then-else condition and then chose what to do once, just to let physics take control subsequently by following a differential equation. That makes for rather short-lived controllers. They have a job only once in their lives. And most decisions they reach may end up being bad ones. Say, one of those controllers, e.g. (7), inspects the state and finds it still okay to accelerate. If it chooses $a := a + 1$ and then lets physics move in the differential equation $x'' = a$, there will probably come a time at which acceleration is no longer such a great idea. But the controller of (7) has no way to change its mind, because he has no more choices and so no control anymore.

If the controller of (7) is supposed to be able to make a second control choice later after physics has followed the differential equation for a while, then (7) can simply be sequentially composed with itself:

$$\begin{aligned}
 & ((?x - o > 5; a := a + 1) \cup a := -b); \\
 & x' = v, v' = a; \\
 & ((?x - o > 5; a := a + 1) \cup a := -b); \\
 & x' = v, v' = a
 \end{aligned} \tag{8}$$

In (8), the cyber controller can first choose to accelerate or brake (depending on whether $x - o > 5$), then physics evolves along differential equation $x'' = a$ for some while, then the controller can again choose whether to accelerate or brake (depending on whether $x - o > 5$ holds in the state reached then), and finally physics again evolves along $x'' = a$.

For a controller that is supposed to be allowed to have a third control choice, copy&paste replication would again help:

$$\begin{aligned}
 & ((?x - o > 5; a := a + 1) \cup a := -b); \\
 & x' = v, v' = a; \\
 & ((?x - o > 5; a := a + 1) \cup a := -b); \\
 & x' = v, v' = a; \\
 & ((?x - o > 5; a := a + 1) \cup a := -b); \\
 & x' = v, v' = a
 \end{aligned} \tag{9}$$

But this is neither a particularly concise nor a particularly useful modeling style. What if a controller could need 10 control decisions or 100? Or what if there is no way of telling ahead of time how many control decisions the cyber part will have to take to reach its goal? Think of how many control decisions you might need when driving in a car from the East Coast to the West Coast. Do you know that ahead of time? Even if you do, do you want to model a system by explicitly replicating its controller that often?

Note 6 (Repetition). *As a more concise and more general way of describing repeated control choices, hybrid programs allow for the repetition operator $*$, which works like the star operator in regular expressions, except that it applies to a hybrid program α as in α^* . It repeats α any number $n \in \mathbb{N}$ of times, by a nondeterministic choice.*

Thus, a way of summarizing (7), (8), (9) and the infinitely many more n -fold replications of (7) for any $n \in \mathbb{N}$, is by using a repetition operator instead:

$$\left(\left((?x - o > 5; a := a + 1) \cup a := -b \right); x' = v, v' = a \right)^* \quad (10)$$

This HP can repeat (7) any number of times $(0, 1, 2, 3, \dots)$.

But how often does a nondeterministic repetition like (10) repeat then? That choice is again nondeterministic.

Note 7 (Nondeterministic $*$). *Repetition ($*$) is nondeterministic. That is, α^* can repeat α any number ($n \in \mathbb{N}$) of times and the choice how often to run α is nondeterministic, i.e. there is no prior way of telling how often α will be repeated.*

Yet, every time the loop in (10) is run, how long does its continuous evolution take? Or, actually, even in the loop-free (8), how long does the first $x'' = a$ take before the controller has its second choice? How long did the continuous evolution take in (7) in the first place?

There is a choice even in following a differential equation. Even if the solution of the differential equation is unique (cf. [Lecture 2](#)), it is still a matter of choice how long to follow that solution. The choice is, as always in hybrid programs, nondeterministic.

Note 8 (Nondeterministic $x' = \theta$). *The duration of evolution of a differential equation ($x' = \theta \ \& \ H$) is nondeterministic (except that the evolution can never be so long that the state leaves H). That is, $x' = \theta \ \& \ H$ can follow the solution of $x' = \theta$ any amount of time ($0 \leq r \in \mathbb{R}$) of times and the choice how long to follow $x' = \theta$ is nondeterministic, i.e. there is no prior way of telling how often $x' = \theta$ will be repeated (except that it can never leave H).*

7 Syntax of Hybrid Programs

With the motivation above, we formally define hybrid programs [[Pla12a](#), [Pla10](#)], in which all of the above operators are allowed.

Definition 1 (Hybrid program). HPs are defined by the following grammar (α, β are HPs, x a variable, θ a term possibly containing x , and H a formula of first-order logic of real arithmetic):

$$\alpha, \beta ::= x := \theta \mid ?H \mid x' = \theta \ \& \ H \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

The first three cases are called atomic HPs, the last three compound. The *test* action $?H$ is used to define conditions. Its effect is that of a *no-op* if the formula H is true in the current state; otherwise, like *abort*, it allows no transitions. That is, if the test succeeds because formula H holds in the current state, then the state does not change, and the system execution continues normally. If the test fails because formula H does not hold in the current state, then the system execution cannot continue, is cut off, and not considered any further.

Nondeterministic choice $\alpha \cup \beta$, sequential composition $\alpha; \beta$, and nondeterministic repetition α^* of programs are as in regular expressions but generalized to a semantics in hybrid systems. *Nondeterministic choice* $\alpha \cup \beta$ expresses behavioral alternatives between the runs of α and β . That is, the HP $\alpha \cup \beta$ can choose nondeterministically to follow the runs of HP α , or, instead, to follow the runs of HP β . The *sequential composition* $\alpha; \beta$ models that the HP β starts running after HP α has finished (β never starts if α does not terminate). In $\alpha; \beta$, the runs of α take effect first, until α terminates (if it does), and then β continues. Observe that, like repetitions, continuous evolutions within α can take more or less time, which causes uncountable nondeterminism. This nondeterminism occurs in hybrid systems, because they can operate in so many different ways, which is as such reflected in HPs. *Nondeterministic repetition* α^* is used to express that the HP α repeats any number of times, including zero times. When following α^* , the runs of HP α can be repeated over and over again, any nondeterministic number of times (≥ 0).

Unary operators (including $*$) bind stronger than binary operators and $;$ binds stronger than \cup , so $\alpha; \beta \cup \gamma \equiv (\alpha; \beta) \cup \gamma$ and $\alpha \cup \beta; \gamma \equiv \alpha \cup (\beta; \gamma)$. Further, $\alpha; \beta^* \equiv \alpha; (\beta^*)$.

8 Semantics of Hybrid Programs

There is more than one way to define the meaning of a program, including defining a denotational semantics, an operational semantics, a structural operational semantics, an axiomatic semantics. For our purposes, what is most relevant is how a hybrid program changes the state of the system. Consequently, the semantics of HPs is based on which final states are reachable from which initial state. It considers which (final) state ω is reachable by running a HP α from an (initial) state ν . Semantical models that expose more detail, e.g., about the internal states during the run of an HP are possible [Pla10, Chapter 4] but not needed for our usual purposes.

Recall that a *state* ν is a mapping from variables to \mathbb{R} . The set of states is denoted \mathcal{S} . The meaning of an HP α is given by a reachability relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ on states. That is, $(\nu, \omega) \in \rho(\alpha)$ means that final state ω is reachable from initial state ν by running

HP α . From any initial state ν , there might be many states ω that are reachable because the HP α may involve nondeterministic choices, repetitions or differential equations, so there may be many different ω for which $(\nu, \omega) \in \rho(\alpha)$. From other initial states ν , there might be no reachable states ω at all for which $(\nu, \omega) \in \rho(\alpha)$. So $\rho(\alpha)$ is a proper relation, not a function.

HPs have a compositional semantics [Pla12b, Pla10, Pla08]. Their semantics is defined by a reachability relation. Recall that the value of term θ in ν is denoted by $\llbracket \theta \rrbracket_\nu$. Recall that $\nu \models H$ denotes that first-order formula H is true in state ν (Lecture 2 on Differential Equations & Domains).

Definition 2 (Transition semantics of HPs). Each HP α is interpreted semantically as a binary reachability relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ over states, defined inductively by

1. $\rho(x := \theta) = \{(\nu, \omega) : \omega = \nu \text{ except that } \llbracket x \rrbracket_\omega = \llbracket \theta \rrbracket_\nu\}$
That is, final state ω differs from initial state ν only in its interpretation of the variable x , which ω changes to the value that the right-hand side θ has in the initial state ν .
2. $\rho(?H) = \{(\nu, \nu) : \nu \models H\}$
That is, the final state ν is the same as the initial state ν (no change) but there only is such a self-loop transition if test formula H holds in ν , otherwise no transition is possible at all and the system is stuck because of a failed test.
3. $\rho(x' = \theta \ \& \ H) = \{(\varphi(0), \varphi(r)) : \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \text{ for a solution } \varphi : [0, r] \rightarrow \mathcal{S} \text{ of any duration } r\}$
That is, the final state $\varphi(r)$ is connected to the initial state $\varphi(0)$ by a continuous function of some duration $r \geq 0$ that solves the differential equation and satisfies H at all times, when interpreting $\varphi(t)(x') \stackrel{\text{def}}{=} \frac{d\varphi(\zeta)(x)}{d\zeta}(t)$ as the derivative of the value of x over time, see Lecture 2.
4. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
That is, $\alpha \cup \beta$ can do any of the transitions that α can do as well as any of the transitions that β is capable of.
5. $\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha) = \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\}$
That is, $\alpha; \beta$ can do any transitions that go through any intermediate state μ to which α can make a transition from the initial state ν and from which β can make a transition to the final state ω .
6. $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?true$.
That is, α^* can repeat α any number of times, i.e., for any $n \in \mathbb{N}$, α^* can act like the n -fold sequential composition α^n would.

To keep things simple, the above definition uses simplifying abbreviations for differential equations. Lecture 2 provides full detail also of the definition for differential

equation systems rather than single differential equations.

For graphical illustrations of the transition semantics of hybrid programs and example dynamics, see Fig. 1. The left of Fig. 1 illustrates the generic shape of the transition structure $\rho(\alpha)$ for transitions along various cases of hybrid programs α from state ν to state ω . The right of Fig. 1 shows examples of how the value of a variable x may evolve over time t when following the dynamics of the respective hybrid program α .

Now when α denotes the HP in (8), its semantics $\rho(\alpha)$ is a relation on states connecting the initial to the final state along the differential equation with two control decisions according to the nondeterministic choice, one at the beginning and one after following the first differential equation. How long that is, exactly? Well, that's nondeterministic, because the semantics of differential equations is such that any final state after any permitted duration is reachable from a given initial state. So the duration for the first differential equation in (8) could have been one second or two or 424 or half a second or zero or any other nonnegative real number.

If we change the HP around and consider the following modification instead:

$$\begin{aligned}
 &?x - o > 5; a := a + 1; \\
 &x' = v, v' = a; \\
 &?x - o > 5; a := a + 1; \\
 &x' = v, v' = a
 \end{aligned} \tag{11}$$

Then some behavior that was still possible in (8) is no longer possible for (11). Let β denote the HP in (11), then the semantics $\rho(\beta)$ of β now only includes relations between initial and final states which can be reached by acceleration choices (because there are no more braking choices in β). In particular, however, note that the duration of the first differential equation in (11) may suddenly be bounded, because if x keeps on accelerating for too long during the first differential equation, the intermediate state reached then will violate the test $?x - o > 5$, which, according to the semantics of tests, will fail and be discarded. Of course, if x accelerates for too long, it will surely ultimately violate the condition that its position will be at least 5 in front of the obstacle o .

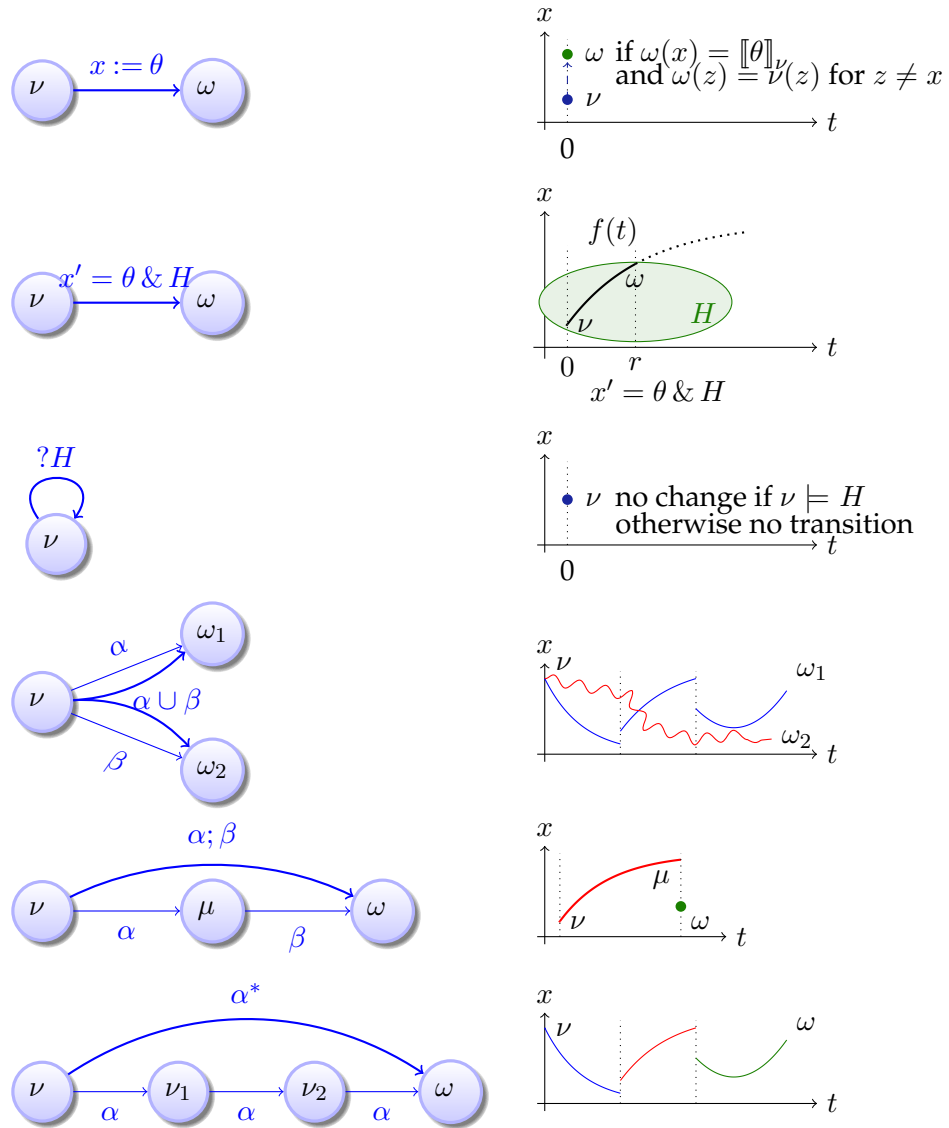


Figure 1: Transition semantics (left) and example dynamics (right) of hybrid programs

9 Summary

This lecture introduced hybrid programs as a model for cyber-physical systems, summarized in Note 11. Hybrid programs combine differential equations with conventional program constructs and discrete assignments. The programming language of hybrid programs embraces nondeterminism as a first-class citizen and features differential equations that can be combined to form hybrid systems using the compositional operators of hybrid programs.

Note 11 (Statements and effects of hybrid programs (HPs)).

HP Notation	Operation	Effect
$x := \theta$	discrete assignment	assigns term θ to variable x
$x' = \theta \ \& \ H$	continuous evolution	differential equations for x with term θ within first-order constraint H (evolution domain)
$?H$	state test / check	test first-order formula H at current state
$\alpha; \beta$	seq. composition	HP β starts after HP α finishes
$\alpha \cup \beta$	nondet. choice	choice between alternatives HP α or HP β
α^*	nondet. repetition	repeats HP α n -times for any $n \in \mathbb{N}$

Exercises

Exercise 1. The semantics of hybrid programs (Def. 2) requires evolution domain constraints H to hold always throughout a continuous evolution. What exactly happens if the system starts in a state where H does not hold to begin with?

Exercise 2. Consider your favorite programming language and discuss in what ways it introduces discrete change and discrete dynamics. Can it model all behavior that hybrid programs can describe? Can your programming language model all behavior that hybrid programs without differential equations can describe? How about the other way around?

Exercise 3. Consider the grammar of hybrid programs. The $;$ in hybrid programs is similar to the $;$ in Java and C0. If you look closely you will find a subtle difference. Identify the difference and explain why there is such a difference.

Exercise 4. Sect. 3 considered if-then-else statements for hybrid programs. But they no longer showed up in the grammar of hybrid programs. Is this a mistake?

Exercise 5. The semantics of hybrid programs (Def. 2) is defined as a transition relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ on states. Define an equivalent semantics based on functions $R(\alpha) : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ from the initial state to the set of all final states, where $2^{\mathcal{S}}$ denotes the powerset of \mathcal{S} , i.e. the set of all subsets of \mathcal{S} . Define this set-valued semantics $R(\alpha)$ without referring to the transition relation semantics $\rho(\alpha)$. Likewise, define an equivalent semantics based on functions $\varsigma(\alpha) : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$ from the set of all initial states to the set of all final states.

References

- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Pla07] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007. doi:[10.1007/978-3-540-73099-6_17](https://doi.org/10.1007/978-3-540-73099-6_17).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:[10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).

Lecture Notes on Safety & Contracts

[André Platzer](#)

Carnegie Mellon University
Lecture 4

1 Introduction

In the previous lectures, we have studied models of cyber-physical systems. Hybrid programs provide a programming language for cyber-physical systems [[Pla12c](#), [Pla08](#), [Pla10](#)] with the most prominent features being differential equations and nondeterminism alongside the usual classical control structures and discrete assignments. This gives powerful and flexible ways of modeling even very challenging systems and very complex control principles. This lecture will start studying ways of making sure that the resulting behavior meets the required correctness standards.

In the [15-122 Principles of Imperative Computation](#) course, you have experienced how contracts can be used to make properties of programs explicit. You have seen how contracts can be checked dynamically at runtime, which, if they fail, will alert you right away to flaws in the design of the programs. You have experienced first hand that it is much easier to find and fix problems in programs starting from the first contract that failed in the middle of the program, rather than from the mere observation about the symptoms that ultimately surface when the final output is not as expected (which you may not notice either unless the output is checked dynamically).

Another aspect of contracts that you have had the opportunity to observe in [Principles of Imperative Computation](#) is that they can be used in proofs that show that every program run will satisfy the contracts. Unlike in dynamic checking, the scope of correctness arguments with proofs extends far beyond the test cases that have been tried, however clever the tests may have been chosen. Both uses of contracts, dynamic checking and rigorous proofs, are very helpful to check whether a system does what we intend it to, as has been argued on numerous occasions in various contexts in the literature, e.g., [[Flo67](#), [Hoa69](#), [Pra76](#), [Mey92](#), [XJC09](#), [PCL11](#), [Log11](#)].

The principles of contracts help cyber-physical systems [Pla08, Pla10, Pla13, DLTT13] as well. Yet, their use in proving may, arguably, be more important than their use in dynamic checking. The reason has to do with the physical impact of CPS and the (relative) non-negotiability of the laws of physics. The reader is advised to imagine a situation where a self-driving car is propelling him or her down the street. Suppose the car's control software is covered with contracts all over, but all of them are exclusively for dynamic checking, none have been proved. If that self-driving car speeds up to 100mph on a 55mph highway and drives up very close to a car in front of it, then dynamically checking the contract "distance to car in front should be more than 1 meter" does not help. If that contract fails, the car's software would know that it made a mistake, but it has become too late to do anything about it, because the brakes of the car will never work out in time. So the car would be "trapped in its own physics", in the sense that it has run out of all safe control options. There are still effective ways of making use of dynamic contract checking in CPS [MP14], but the design of those contracts then requires proof to ensure that safety is always maintained.

For those reasons, this course will focus on the role of proofs as correctness arguments much more than on dynamical checking of contracts. Because of the physical consequences of malfunctions, correctness requirements on CPS are also more stringent. And their proofs involve significantly more challenging arguments than in [Principles of Imperative Computation](#). For those reasons, we will approach CPS proofs with much more rigor than what you have seen in [Principles of Imperative Computation](#). But that is a story for a later lecture. The focus of today's lecture will be to understand CPS contracts and the first basics of reasoning about CPS. Subsequent lectures will ultimately identify a much cleaner, more elegant, and more general style of reasoning about CPS of which the reasoning approach developed in today's lecture are a special case. But today's lecture is a useful stepping stone for reaching that generality.

This material is based on correctness specifications and proofs for CPS [Pla12c, Pla07, Pla08, Pla10]. We will come back to more details in later lectures, where we will also use the KeYmaera prover for verifying CPS [PQ08]. More information about safety and contracts can be found in [Pla10, Chapter 2.2,2.3].

The focus of today's lecture is on developing and studying a model of a bouncing ball and on identifying all requirements for it to be safe. Along the way, however, this lecture develops an intuitive understanding for the role of requirements and contracts in CPS as well as important ways of formalizing CPS properties and their analyzes. The most important learning goals of this lecture are:

Modeling and Control: We deepen our understanding of the core principles behind CPS by relating discrete and continuous aspects of CPS to analytic reasoning principles.

Computational Thinking: We go through a simple but very instructive example to learn how to identify specifications and critical properties of CPS. Even if the example we look at, the bouncing ball, is a rather impoverished CPS, it still formidably conveys the subtleties involved with hybrid systems models, which are crucial for understanding CPS. This lecture is further devoted to contracts in the form of pre-

and post-conditions for CPS models. We will begin to reason rigorously about CPS models, which is critical to getting CPS right. CPS designs can be flawed for very subtle reasons. Without sufficient rigor in their analysis it can be impossible to spot the flaws, and even more challenging to say for sure whether and why a design is no longer faulty. This lecture introduces *differential dynamic logic* $\text{d}\mathcal{L}$ [Pla12c, Pla08, Pla10] as the specification and verification language for CPS that we will be using throughout this course.

CPS Skills: We will begin to deepen our understanding of the semantics of CPS models by relating it to their reasoning principles. A full study of this alignment will only be covered in the next lecture, though.

2 The Adventures of a Bouncing Ball

Lecture 3 considered hybrid programs that model a choice of increasing acceleration or braking.

$$\left(\left((?x - o > 5; a := a + 1) \cup a := -b \right); x' = v, v' = a \right)^* \quad (1)$$

That model did perform interesting control choices and we could continue to study it in this lecture.

In order to sharpen our intuition about CPS, we will, however, study a very simple but also very intuitive system instead. Once upon a time, there was a little bouncing ball that had nothing else to do but bounce up and down the street until it was tired of doing that (Fig. 1). The bouncing ball was not much of a CPS, because the poor bounc-

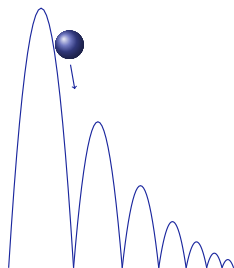


Figure 1: Sample trajectory of a bouncing ball (plotted as height over time)

ing ball does not actually have any interesting decisions to make. But it nevertheless formed a perfectly reasonable hybrid system, because, after a closer look, it turns out to involve discrete and continuous dynamics. The continuous dynamics is caused by gravity, which is pulling the ball down and makes it fall from the sky in the first place. The discrete dynamics comes from the singular discrete event of what happens when the ball hits the ground and bounces back up. There are a number of ways of modeling

the ball and its impact on the ground with physics. They include a whole range of different more or less realistic physical effects including gravity, aerodynamic resistance, the elastic deformation on the ground, and so on and so on. But the little bouncing ball didn't study enough physics to know anything about those effects. And so it had to go about understanding the world in easier terms. It was a clever bouncing ball, though, so it had experienced the phenomenon of sudden change and was trying to use that to its advantage.

If we are looking for a very simple model of what the bouncing ball does, it is easier to describe as a hybrid system. The ball at height x is falling subject to gravity:

$$x'' = -g$$

When it hits the ground, which is assumed at height $x = 0$, the ball bounces back and jumps back up in the air. Yet, as every child knows, the ball tends to come back up a little less high than before. Given enough time to bounce around, it will ultimately lie flat on the ground forever. Until it is picked up again and thrown high up in the air.

Let us model the impact on the ground as a discrete phenomenon and describe what happens so that the ball jumps back up then. One attempt of understanding this could be to make the ball jump back up rather suddenly by increasing its height by, say, 10 when it hit the ground $x = 0$:

$$\begin{aligned} x'' &= -g; \\ \text{if}(x = 0) \ x &:= x + 10 \end{aligned} \tag{2}$$

Such a model may be useful for other systems, but would be rather at odds with our physical experience with bouncing balls, because the ball is indeed slowly climbing back up rather than suddenly being way up in the air again.

The bouncing ball ponders about what happens when it hits the ground. It does not suddenly get teleported to a new position above ground like (2) would suggest. Instead, the ball suddenly changes its direction. A moment ago, it used to fall down with a negative velocity (i.e. one that is pointing down into the ground) and suddenly climbs back up with a positive velocity (pointing up into the sky). In order to be able to write such a model, the velocity v will be made explicit in the bouncing ball's differential equation:

$$\begin{aligned} x' &= v, v' = -g; \\ \text{if}(x = 0) \ v &:= -v \end{aligned} \tag{3}$$

Of course, something happens after the bouncing ball reversed its direction because it hit the ground. Physics continues until it hits the ground again.

$$\begin{aligned} x' &= v, v' = -g; \\ \text{if}(x = 0) \ v &:= -v \\ x' &= v, v' = -g; \\ \text{if}(x = 0) \ v &:= -v \end{aligned} \tag{4}$$

Then, of course, physics moves on again, so the model actually involves a repetition:

$$\begin{aligned} &(x' = v, v' = -g; \\ &\text{if}(x = 0) v := -v)^* \end{aligned} \tag{5}$$

Yet, the bouncing ball is now rather surprised. For if it follows that HP (5), it seems as if it should always be able to come back up to its initial height again. Excited about that possibility, it tries and tries again but never succeeds to bounce back up as high as it was before. So there must be something wrong with the model in (5), the ball concludes and sets out to fix (5).

Having observed itself rather carefully, the bouncing ball concludes that it feels slower when bouncing back up than it used to be when falling on down. Indeed, it feels less energetic on its way up. So its velocity must not only flip direction from down to up, at a bounce, but also seems to shrink in magnitude. The bouncing ball swiftly calls the corresponding damping factor c and quickly comes up with a better model of itself:

$$\begin{aligned} &(x' = v, v' = -g; \\ &\text{if}(x = 0) v := -cv)^* \end{aligned} \tag{6}$$

Yet, running that model in clever ways, the bouncing ball observes that model (6) could make it fall through the cracks in the ground. Terrified at that thought, the bouncing ball quickly tries to set the physics right, lest it falls through the cracks in space before it had a chance to fix its physics. The issue with (6) is that its differential equation isn't told when to stop. Yet, the bouncing ball luckily remembers that this is quite exactly what evolution domains were meant for. Above ground is what it wants to remain, and so $x \geq 0$ is what the ball asks dear physics to obey, since the table is of rather sturdy built:

$$\begin{aligned} &(x' = v, v' = -g \ \& \ x \geq 0; \\ &\text{if}(x = 0) v := -cv)^* \end{aligned} \tag{7}$$

Now, indeed, physics will have to stop evolving before gravity has made our little bouncing ball fall through the ground. Yet, physics could still choose to stop evolving while the ball is still high up in the sky. In that case, the ball will not yet be on the ground and line 2 of (7) would have no effect because $x \neq 0$ still. This is not a catastrophe, however, because the loop in (7) could simply repeat, which would allow physics to continue to evolve the differential equation further.

Quite happy with model (7) for itself, the bouncing ball goes on to explore whether the model does what the ball expects it to do.

3 Postcondition Contracts for CPS

Hybrid programs are interesting models for CPS. They describe the behavior of a CPS, ultimately captured by their semantics $\rho(\alpha)$, which is a reachability relation on states ([Lecture 3 on Choice & Control](#)). Yet, reliable development of CPS also needs a way of

ensuring that the behavior will be as expected. So, for example, we may want the behavior of a CPS to always satisfy certain crucial safety properties. A robot, for example, should never do something unsafe like running over a human being.¹

The little bouncing ball may consider itself less safety-critical, except that it may be interested in its own safety. It still wants to make sure that it couldn't ever fall through the cracks in the ground. And even though it would love to jump all the way up to the moon, the ball is also terrified of big heights and would never want to jump any higher than it was in the very beginning. So, when H denotes the initial height, the bouncing ball would love to know whether its height will always stay within $0 \leq x \leq H$ when following HP (7).

Scared of what otherwise might happen to it if $0 \leq x \leq H$ should ever be violated, the bouncing ball decides to make its goals for the HP (7) explicit. Fortunately, the bouncing ball excelled in the course [Principles of Imperative Computation](#) and recalls that contracts such as `@requires` and `@ensures` have been used in that course to make behavioral expectations for C0 programs explicit. Even though the bouncing ball clearly does not deal with a C0 program, but rather a hybrid program, it still puts an `@ensures(F)` contracts in front of HP (7) to express that all runs of that HP are expected to lead only to states in which logical formula F is true. The bouncing ball even uses `@ensures` twice, once for each of its expectations.

$$\begin{aligned} & @ensures(0 \leq x) \\ & @ensures(x \leq H) \\ & (x' = v, v' = -g \ \& \ x \geq 0; \\ & \text{if}(x = 0) \ v := -cv)^* \end{aligned} \tag{8}$$

4 Precondition Contracts for CPS

Having learned from the [Principles of Imperative Computation](#) experience, the little bouncing ball immediately starts thinking about whether the `@ensures` contracts in (8) would, in fact, always be true after running that HP. After all, the bouncing ball would really love to know that it can rely on that contract never failing.

Wondering about whether the `@ensures` contract in (8) would always succeed, the bouncing ball notices that this would have to depend on what values the bouncing ball starts with. It called H its initial height, but the HP (8) cannot know that. For one thing,

¹Safety of robots has, of course, been aptly defined by Asimov [[Asi42](#)] with his Three Laws of Robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

But their exact rendition in logic still remains a bit of a challenge due to language ambiguities and similar nuisance.

the contracts in (8) would be hard to fulfill if $H = -5$, because $0 \leq x$ and $x \leq H$ can impossibly both be true then.

So the bouncing ball figures it should demand a `@requires` contract with the precondition $x = H$ to say that the height, x , of the bouncing ball is initially H . Because that still does not (obviously) ensure that $0 \leq x$ has a chance of holding, it requires $0 \leq H$ to hold initially:

$$\begin{aligned}
 & @requires(x = H) \\
 & @requires(0 \leq H) \\
 & @ensures(0 \leq x) \\
 & @ensures(x \leq H) \\
 & (x' = v, v' = -g \ \& \ x \geq 0; \\
 & \text{if}(x = 0) \ v := -cv)^*
 \end{aligned} \tag{9}$$

5 Invariant Contracts for CPS

The little bouncing ball remembers the prominent role that invariants have played in the course [Principles of Imperative Computation](#). So, the ball ventures including an invariant with its HP. In C0, invariants were associated with loops, e.g.

```

i = 0;
while (i < 10)
  //@loop_invariant 0 <= i && i <= 10;
  {
    i++;
  }

```

The bouncing ball, thus, figures that invariants for loops in HPs should also be associated with a loop, which is written α^* for nondeterministic repetition. After a moment's thought, the bouncing ball decides that falling through the cracks in the ground is still it's biggest worry, so the invariant it'd like to maintain is $x \geq 0$:

$$\begin{aligned}
 & @requires(x = H) \\
 & @requires(0 \leq H) \\
 & @ensures(0 \leq x) \\
 & @ensures(x \leq H) \\
 & (x' = v, v' = -g \ \& \ x \geq 0; \\
 & \text{if}(x = 0) \ v := -cv)^* @invariant(x \geq 0)
 \end{aligned} \tag{10}$$

On second thought, the little bouncing ball is less sure what exactly the `@invariant(F)` contract would mean for a CPS. So it decides to first give more thought to the proper way of phrasing CPS contracts and what they mean.

We will get back to the `@invariant(F)` construct in a later lecture.

6 Logical Formulas for Hybrid Programs

CPS contracts play a very useful role in the development of CPS models and CPS programs. Using them as part of their design right from the very beginning is a good idea, probably even more crucial than it was in [15-122 Principles of Imperative Computation](#) for the development of C0 programs, because CPS have more stringent requirements on safety.

Yet, we do not only want to program CPS, we also want to and have to understand thoroughly what they mean, what their contracts mean, and how we convince ourselves that the CPS contracts are respected by the CPS program. It turns out that this is where mere contracts are at a disadvantage compared to full logic.

Note 1 (Logic is for specification and reasoning). *Logic allows not only the specification of a whole CPS program, but also an analytic inspection of its parts as well as argumentative relations between contracts and program parts.*

Differential dynamic logic (dL) [[Pla12c](#), [Pla08](#), [Pla12a](#), [Pla07](#), [Pla10](#)] is the logic of hybrid systems that this course uses for specification and verification of cyber-physical systems. There are more aspects of logic for cyber-physical systems [[Pla12c](#), [Pla12b](#)], which will be studied (to some extent) in later parts of this course.

The most unique feature of differential dynamic logic for our purposes is that it allows us to refer to hybrid systems. [Lecture 2 on Differential Equations & Domains](#) introduced first-order logic of real arithmetic.

Note 2 (Limits of first-order logic for CPS). *First-order logic of real arithmetic is a crucial basis for describing what is true and false about CPS, because it allows us to refer to real-valued quantities like positions and velocities and their arithmetic relations. Yet, that is not enough, because first-order logic describes what is true in a single state of a system. It has no way of referring to what will be true in future states of a CPS, nor of describing the relationship of the initial state of the CPS to the final state of the CPS.*

Recall that this relationship, $\rho(\alpha)$, is what ultimately constitutes the semantics of HP α .

Note 3 (Differential dynamic logic principle). *Differential dynamic logic (dL) extends first-order logic of real arithmetic with operators that refer to the future states of a CPS in the sense of referring to the states that are reachable by running a given HP. The logic dL provides a modal operator $[\alpha]$, parametrized by α , that refers to all states reachable by HP α according to the reachability relation $\rho(\alpha)$ of its semantics. This modal operator can be placed in front of any dL formula ϕ . The dL formula*

$$[\alpha]\phi$$

expresses that all states reachable by HP α satisfy formula ϕ .

The logic dL also provides a modal operator $\langle\alpha\rangle$, parametrized by α , that can be placed in front of any dL formula ϕ . The dL formula

$$\langle\alpha\rangle\phi$$

expresses that there is at least one state reachable by HP α for which ϕ holds. The modalities $[\alpha]$ and $\langle\alpha\rangle$ can be used to express necessary or possible properties of the transition behavior of α , because they refer to all or some runs of α

An $\text{@ensures}(E)$ postcondition for a HP α can be expressed directly as a logical formula in dL:

$$[\alpha]E$$

So, the first CPS postcondition $\text{@ensures}(0 \leq x)$ for the bouncing ball HP in (8) can be stated as a dL formula:

$$[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] 0 \leq x \quad (11)$$

The second CPS postcondition $\text{@ensures}(x \leq H)$ for the bouncing ball HP in (8) can be stated as a dL formula as well:

$$[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] x \leq H \quad (12)$$

The logic dL allows all other logical operators from first-order logic, including conjunction (\wedge). So, the two dL formulas (11) and (12) can be stated together as a single dL formula:

$$[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] 0 \leq x \wedge [(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] x \leq H \quad (13)$$

Stepping back, we could also have combined the two postconditions $\text{@ensures}(0 \leq x)$ and $\text{@ensures}(x \leq H)$ into a single postcondition $\text{@ensures}(0 \leq x \wedge x \leq H)$. The translation of that into dL would have gotten us an alternative way of combining both statements about the lower and upper bound on the height of the bouncing ball into a single dL formula:

$$[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] (0 \leq x \wedge x \leq H) \quad (14)$$

Which way of representing what we expect bouncing balls to do is better? Like (13) or like (14)? Are they equivalent? Or do they express different things?

It turns out that there is a very simple argument within the logic $\text{d}\mathcal{L}$ that shows that (13) and (14) are equivalent. And not just that those two particular logical formulas are equivalent but that the same equivalence holds for any $\text{d}\mathcal{L}$ formulas of this form. This will be investigated formally in a later lecture, but it is useful to observe now already to sharpen our intuition.

Having said that, do we believe $\text{d}\mathcal{L}$ formula (13) should be valid? Should (14) be valid? Before we study this question in any further detail, the first question should be what it means for a modal formula $[\alpha]\phi$ to be true. What is its semantics? Better yet, what exactly is its syntax in the first place?

7 Syntax of Differential Dynamic Logic

The formulas of differential dynamic logic are defined like the formulas of first-order logic of real arithmetic with the additional capability of using modal operators for any hybrid program α .

Definition 1 ($\text{d}\mathcal{L}$ formula). The *formulas of differential dynamic logic* ($\text{d}\mathcal{L}$) are defined by the grammar (where ϕ, ψ are $\text{d}\mathcal{L}$ formulas, θ_1, θ_2 (polynomial) terms, x a variable, α a HP):

$$\phi, \psi ::= \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi \mid \langle \alpha \rangle \phi$$

Operators $>, \leq, <, \leftrightarrow$ can be defined as usual, e.g., $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

We use the notational convention that unary operators (including \neg and quantifiers $\forall x, \exists x$ and modalities $[\alpha], \langle \alpha \rangle$)² bind stronger than binary operators. In particular, quantifiers and modal operators bind strong, i.e. their scope only extends to the formula immediately after. Thus, $[\alpha]\phi \wedge \psi \equiv ([\alpha]\phi) \wedge \psi$ and $\forall x \phi \wedge \psi \equiv (\forall x \phi) \wedge \psi$. In our notation, we also let \wedge bind stronger than \vee , which binds stronger than $\rightarrow, \leftrightarrow$. We also associate \rightarrow to the right so that $\phi \rightarrow \psi \rightarrow \varphi \equiv \phi \rightarrow (\psi \rightarrow \varphi)$. To avoid confusion, we do not adopt precedence conventions between $\rightarrow, \leftrightarrow$ but expect explicit parentheses. So $\phi \rightarrow \psi \leftrightarrow \varphi$ would be considered illegal and explicit parentheses are required to distinguish $\phi \rightarrow (\psi \leftrightarrow \varphi)$ from $(\phi \rightarrow \psi) \leftrightarrow \varphi$. Likewise $\phi \leftrightarrow \psi \rightarrow \varphi$ would be considered illegal and explicit parentheses are required to distinguish $\phi \leftrightarrow (\psi \rightarrow \varphi)$ from $(\phi \leftrightarrow \psi) \rightarrow \varphi$.

² Quantifiers are only quite arguably understood as unary operators. Yet, $\forall x$ is a unary operator on formulas while \forall would be an operator with arguments of mixed syntactic categories. In a higher-order context, it can also be understood more formally by understanding $\forall x \phi$ as an operator on functions: $\forall(\lambda x. \phi)$. Similar cautionary remarks apply to the understanding of modalities as unary operators. The primary reason for adopting this understanding is that it simplifies the precedence rules.

8 Semantics of Differential Dynamic Logic

For $\text{d}\mathcal{L}$ formulas that are also formulas of first-order real arithmetic (i.e. formulas without modalities), the semantics of $\text{d}\mathcal{L}$ formulas is the same as that of first-order real arithmetic. The semantics of modalities $[\alpha]$ and $\langle\alpha\rangle$ quantifies over all ($[\alpha]$) or some ($\langle\alpha\rangle$) of the (final) states reachable by following HP α , respectively.

Definition 2 (dL semantics). The *satisfaction relation* $\nu \models \phi$ for a dL formula ϕ in state ν is defined inductively:

- $\nu \models (\theta_1 = \theta_2)$ iff $\llbracket \theta_1 \rrbracket_\nu = \llbracket \theta_2 \rrbracket_\nu$.
That is, an equation is true in a state ν iff the terms on both sides evaluate to the same number.
- $\nu \models (\theta_1 \geq \theta_2)$ iff $\llbracket \theta_1 \rrbracket_\nu \geq \llbracket \theta_2 \rrbracket_\nu$.
That is, a greater-or-equals inequality is true in a state ν iff the term on the left evaluate to a number that is greater or equal to the value of the right term.
- $\nu \models \neg\phi$ iff $\nu \not\models \phi$, i.e. if it is not the case that $\nu \models \phi$.
That is, a negated formula $\neg\phi$ is true in state ν iff the formula ϕ itself is not true in ν .
- $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$.
That is, a conjunction is true in a state iff both conjuncts are true in said state.
- $\nu \models \phi \vee \psi$ iff $\nu \models \phi$ or $\nu \models \psi$.
That is, a disjunction is true in a state iff either of its disjuncts is true in said state.
- $\nu \models \phi \rightarrow \psi$ iff $\nu \not\models \phi$ or $\nu \models \psi$.
That is, an implication is true in a state iff either its left-hand side is false or its right-hand side true in said state.
- $\nu \models \phi \leftrightarrow \psi$ iff $(\nu \models \phi \text{ and } \nu \models \psi)$ or $(\nu \not\models \phi \text{ and } \nu \not\models \psi)$.
That is, a biimplication is true in a state iff both sides are true or both sides are false in said state.
- $\nu \models \forall x \phi$ iff $\nu_x^d \models \phi$ for all $d \in \mathbb{R}$.
That is, a universally quantified formula $\forall x \phi$ is true in a state iff its kernel ϕ is true in all variations of the state, no matter what real number d the quantified variable x evaluates to in the variation ν_x^d .
- $\nu \models \exists x \phi$ iff $\nu_x^d \models \phi$ for some $d \in \mathbb{R}$.
That is, an existentially quantified formula $\exists x \phi$ is true in a state iff its kernel ϕ is true in some variation of the state, for a suitable real number d that the quantified variable x evaluates to in the variation ν_x^d .
- $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$.
That is, a box modal formula $[\alpha]\phi$ is true in state ν iff postcondition ϕ is true in all states ω that are reachable by running α from ν .
- $\nu \models \langle \alpha \rangle \phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$.
That is, a diamond modal formula $\langle \alpha \rangle \phi$ is true in state ν iff postcondition ϕ is true in at least one state ω that is reachable by running α from ν .

If $\nu \models \phi$, then we say that ϕ is true at ν or that ν is a model of ϕ . A formula ϕ is *valid*, written $\models \phi$, iff $\nu \models \phi$ for all states ν . A formula ϕ is a *consequence* of a set of formulas Γ , written $\Gamma \models \phi$, iff, for each ν : ($\nu \models \psi$ for all $\psi \in \Gamma$) implies that $\nu \models \phi$.

A formula ϕ is called *satisfiable* iff there is a ν such that $\nu \models \phi$. The formula ϕ is called *unsatisfiable* iff there is no such ν .

The formula $x > 0 \wedge x < 1$ is satisfiable, because all it takes for it to be true is a state ν in which, indeed, the value of x is a real number between zero and one. The formula $x > 0 \wedge x < 0$ is unsatisfiable, because it is kind of hard (read: impossible) to find a state which satisfies both conjuncts. The formula $x > 0 \vee x < 1$ is valid, because there is no state in which it would not be true, because, surely, x will either be positive or smaller than one.

9 CPS Contracts in Logic

Now that we know what truth and validity are, let's go back to the previous question. Is $\text{d}\mathcal{L}$ formula (13) valid? Is (14) valid? Actually, let's first ask if they are equivalent, i.e. the $\text{d}\mathcal{L}$ formula

$$(13) \leftrightarrow (14)$$

is valid. Expanding the abbreviations this is the question whether the following $\text{d}\mathcal{L}$ formula is valid:

$$\begin{aligned} & \left([(x' = v, v' = -g \ \& \ x \geq 0; \text{ if}(x = 0) \ v := -cv)^*] \ 0 \leq x \right. \\ & \quad \left. \wedge [(x' = v, v' = -g \ \& \ x \geq 0; \text{ if}(x = 0) \ v := -cv)^*] \ x \leq H \right) \\ & \leftrightarrow [(x' = v, v' = -g \ \& \ x \geq 0; \text{ if}(x = 0) \ v := -cv)^*] (0 \leq x \wedge x \leq H) \end{aligned} \quad (15)$$

Exercise 1 gives you an opportunity to convince yourself that the equivalence (13) \leftrightarrow (14) is indeed valid.³ So if (13) is valid, then so should (14) be (Exercise 2). But is (13) even valid?

³This equivalence also foreshadows the fact that CPS provide ample opportunity for questions how multiple system models relate. The $\text{d}\mathcal{L}$ formula (15) relates three different properties of three occurrences of one and the same hybrid program, for example. Over the course of the semester, the need to relate different properties of different CPS will arise more and more even if it may lie dormant for the moment. You are advised to already take notice that this is possible, because $\text{d}\mathcal{L}$ can form any arbitrary combination and nesting of all its logical operators.

Certainly, (13) is not true in a state ν where $\nu(x) < 0$, because from that initial state, no repetitions of the loop (which is allowed by nondeterministic repetition, Exercise 4), will lead to a state $\omega \stackrel{\text{def}}{=} \nu$ in which $\omega \not\models 0 \leq x$. Thus, (13) only has a chance of being valid in initial states that satisfy further assumptions, including $0 \leq x$ and $x \leq H$. In fact, that is what the preconditions were meant for in Sect. 4. How can we express a precondition contract in a $\text{d}\mathcal{L}$ formula?

Preconditions serve a very different role than postconditions do. Postconditions of HP α are what we want to hold true after every run of α . The meaning of a postcondition is what is rather difficult to express in first-order logic (to say the least). That is what $\text{d}\mathcal{L}$ has modalities for. Do we also need any extra logical operator to express preconditions?

The meaning of a precondition $\text{@requires}(A)$ of a HP α is that it is assumed to hold before the HP starts. If A holds when the HP starts, then its postcondition $\text{@ensures}(B)$ holds after all runs of HP α . What if A does not hold when the HP starts?

If precondition A does not hold initially, then all bets are off, because the person who started the HP did not obey its requirements, which says that it should only be run if its preconditions are met. The CPS contract $\text{@requires}(A) \text{@ensures}(B)$ for a HP α promises that B will always hold after running α if A was true initially when α started. Thus, the meaning of a precondition can be expressed easily using an implication

$$A \rightarrow [\alpha]B \quad (16)$$

because an implication is valid if, in every state in which the left-hand side is true, the right-hand side is also true. The implication (16) is valid ($\models A \rightarrow [\alpha]B$), if, indeed, for every state ν in which precondition A holds ($\nu \models A$), it is the case that all runs of HP α lead to states ω (with $(\nu, \omega) \in \rho(\alpha)$) in which postcondition B holds ($\omega \models B$). The $\text{d}\mathcal{L}$ formula (16) does not say what happens in states ν in which the precondition A does not hold ($\nu \not\models A$).

How does formula (16) talk about the runs of a HP and postcondition B again? Recall that the $\text{d}\mathcal{L}$ formula $[\alpha]B$ is true in exactly those states in which all runs of HP α lead only to states in which postcondition B is true. The implication in (16), thus, ensures that this holds in all (initial) states that satisfy precondition A .

Note 6 (Contracts to $\text{d}\mathcal{L}$ Formulas). Consider a HP α with a CPS contract using a single $\text{@requires}(A)$ precondition and a single $\text{@ensures}(B)$ postcondition:

$$\begin{array}{c} \text{@requires}(A) \\ \text{@ensures}(B) \\ \alpha \end{array}$$

This CPS contract can be expressed directly as a logical formula in $\text{d}\mathcal{L}$:

$$A \rightarrow [\alpha]B$$

CPS contracts with multiple preconditions and multiple postconditions can directly be expressed as a $\text{d}\mathcal{L}$ formula as well (Exercise 7).

Recall HP (10), which is shown here in a slightly simplified form:

$$\begin{aligned} & @requires(0 \leq x \wedge x = H) \\ & @ensures(0 \leq x \wedge x \leq H) \\ & (x' = v, v' = -g \ \& \ x \geq 0; \\ & \quad \text{if}(x = 0) \ v := -cv)^* \end{aligned} \tag{17}$$

The $\text{d}\mathcal{L}$ formula expressing that the CPS contract for HP (17) holds is:

$$0 \leq x \wedge x = H \rightarrow [(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] (0 \leq x \wedge x \leq H) \tag{18}$$

So to find out whether (17) satisfies its CPS contract, we ask whether the $\text{d}\mathcal{L}$ formula (18) is valid.

In order to find out whether such a formula is valid, i.e. true in all states, we need some operational way that allows us to tell whether it is valid, because mere inspection of the semantics alone is not a particularly scalable way of approaching validity question.

10 Identifying Requirements of a CPS

Before trying to prove any formulas to be valid, it is a pretty good idea to check whether all required assumptions have been found that are necessary for the formula to hold. Otherwise, the proof will fail and we need to start over after having identified the missing requirements from the failed proof attempt. So let us scrutinize $\text{d}\mathcal{L}$ formula (18) and ponder whether there are any circumstances under which it is not true. Even though the bouncing ball is a rather impoverished CPS (it suffers from a disparate lack of control), its immediate physical intuition still makes the ball a particularly insightful example for illustrating how critical it is to identify the right requirements. Besides, unlike for heavy duty CPS, we trust you have had ample opportunities to make yourself familiar with the behavior of bouncing balls.

Maybe the first thing to notice is that the HP mentions g , which is meant to represent the standard gravity constant, but the formula (18) never says that. Certainly, if gravity were negative ($g < 0$), bouncing balls would function rather differently in a quite astonishing way. They would suddenly be floating balls disappearing into the sky and would lose all the joy of bouncing around; see Fig. 2.

So let's modify (18) to assume $g = 9.81$:

$$0 \leq x \wedge x = H \wedge g = \mathbf{9.81} \rightarrow [(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] (0 \leq x \wedge x \leq H) \tag{19}$$

Let's undo unnecessarily strict requirements right away, though. What would the bouncing ball do if it were set loose on the moon instead of on Earth? Would it still fall? Things are much lighter on the moon. Yet they still fall down ultimately, which

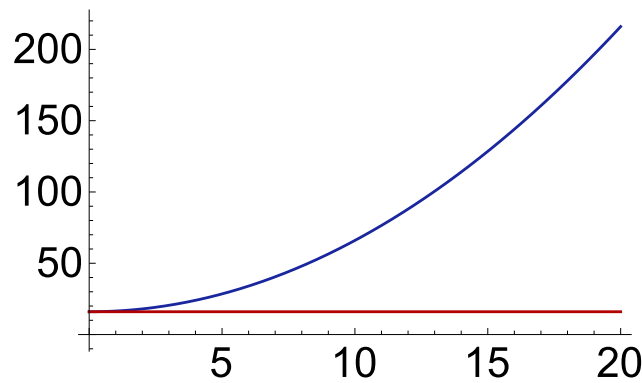


Figure 2: Sample trajectory of a bouncing ball in an anti-gravity field with $g < 0$

is again the phenomenon known as gravity, just with a different constant (1.6 on the moon and 25.9 on Jupiter). Besides, none of those constants was particularly precise. Earth's gravity is more like 9.8067. The behavior of the bouncing ball depends on the value of that parameter g . But its qualitative behavior and whether it obeys (18) does not.

Note 7 (Parameters). *A common feature of CPS is that their behavior is subject to parameters, which can have quite a non-negligible impact. Yet, it is very hard to determine precise values for all parameters by measurements. When a particular concrete value for a parameter has been assumed to prove a property of a CPS, it is not clear whether that property holds for the true system, which may in reality have a slightly different parameter value.*

Instead of a numerical value for a parameter, our analysis can proceed just fine by treating the parameter as a symbolic parameter, i.e. a variable such as g , which is not assumed to hold a specific numerical value like 9.81. Instead, we would only assume certain constraints about the parameter, say $g > 1$ without choosing a specific value. If we then analyze the CPS with this symbolic parameter g , all analysis results will continue to hold for any concrete choice of g respecting its constraints (here $g > 1$). That results in a stronger statement about the system, which is less fragile as it does not break down just because the true g is ≈ 9.8067 rather than the previously assumed $g = 9.81$. Often times, those more general statements with symbolic parameters can even be easier to prove than statements about systems with specific magic numbers chosen for their parameters.

In light of these thoughts, we could assume $9 < g < 10$ to be the gravity constant for Earth. Yet, we can also just consider all bouncing balls on all planets in the solar system or elsewhere at once by assuming only $g > 0$ instead of $g = 9.81$ as in (19), since this is the only aspect of gravity that the usual behavior of a bouncing ball depends on:

$$0 \leq x \wedge x = H \wedge g > 0 \rightarrow [(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] \ (0 \leq x \wedge x \leq H) \quad (20)$$

Do we expect $d\mathcal{L}$ formula (20) to be valid, i.e. true in all states? What could go wrong? The insight from modifying (18) to (19) and finally to (20) started with the observation that (18) did not include any assumptions about g . It is worth noting that (20) also does not assume anything about c . Bouncing balls clearly would not work as expected if $c > 1$, because such anti-damping would cause the bouncing ball to jump back up higher and higher and higher and ultimately as high up as the moon, clearly falsifying (20); see Fig. 3.

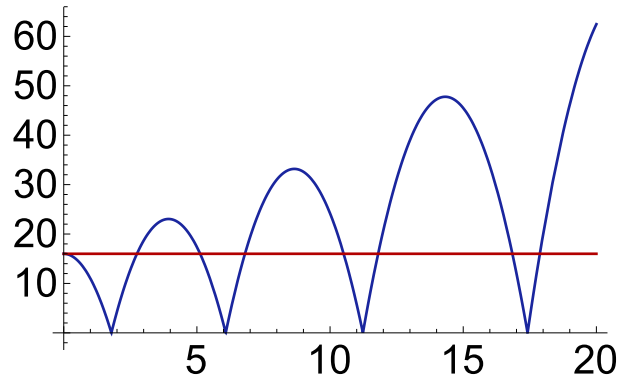


Figure 3: Sample trajectory of a bouncing ball with anti-damping $c > 1$

Consequently, (20) only has a chance of being true when assuming that c is not too big:

$$0 \leq x \wedge x = H \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \left[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^* \right] (0 \leq x \wedge x \leq H) \quad (21)$$

Is (21) valid now? Or does its truth depend on more assumptions that have not been identified yet? Now, all parameters (H, g, c) have some assumptions in (21). Is there some requirement we forgot about? Or did we find them all?

Before you read on, see if you can find the answer for yourself.

What about variable v ? Why is there no assumption about it yet? Should there be one? Velocity v changes over time. What is its initial value allowed to be? What could go wrong?

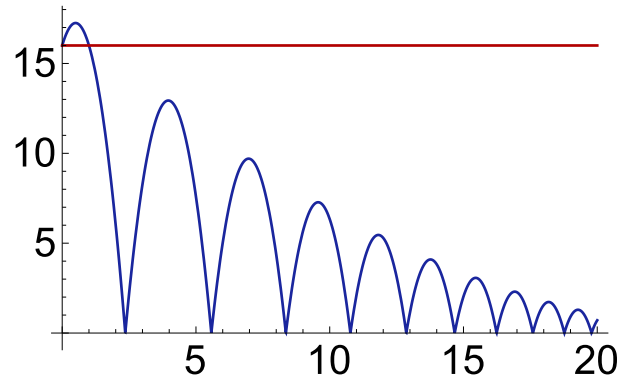


Figure 4: Sample trajectory of a bouncing ball climbing with upwards initial velocity $v > 0$

Indeed, the initial velocity v of the bouncing ball could be positive ($v > 0$), which would make the bouncing ball climb initially, clearly exceeding its initial height H ; see Fig. 4. This would correspond to the bouncing ball being thrown high up in the air in the beginning, so that its initial velocity v is upwards from its initial height $x = H$. Consequently, (21) has to be modified to assume $v \leq 0$ holds initially:

$$0 \leq x \wedge x = H \wedge v \leq 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \left[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^* \right] (0 \leq x \wedge x \leq H) \quad (22)$$

Now there's finally assumptions about all parameters and variables of (22). That does not mean that we found the right assumptions, yet, but is still a good sanity check. Before wasting cycles on trying to prove or otherwise justify (22), let's try once more whether we can find an initial state ν that satisfies all assumptions $v \leq 0 \wedge 0 \leq x \wedge x = H \wedge g > 0 \wedge 1 > c \geq 0$ in the antecedent (i.e. left-hand side of the implication) of (22) so that ν does not satisfy the succedent (i.e. right-hand side of implication) of (22). Such an initial state ν falsifies (22) and would, thus, represent a *counterexample*.

Is there still a counterexample to (22)? Or have we successfully identified all assumptions so that it is now valid?

Before you read on, see if you can find the answer for yourself.

Formula (22) still has a problem. Even if the initial state satisfies all requirements in the antecedent of (22), the bouncing ball might still jump higher than it ought to, i.e. higher than its initial height H . That happens if the bouncing ball has a very big downwards velocity, so if v is a lot smaller than 0 (sometimes written $v \ll 0$). If v is a little smaller than 0, then the damping c will eat up enough the ball's kinetic energy so that it cannot jump back up higher than it was initially (H). But if v is a lot smaller than 0, then it starts falling down with so much kinetic energy that the damping on the ground does not slow it down enough, so the ball will come bouncing back higher than it was originally like when dribbling a basket ball; see Fig. 5. Under which circumstance this happens depends on the relationship of the initial velocity and height to the damping coefficient.

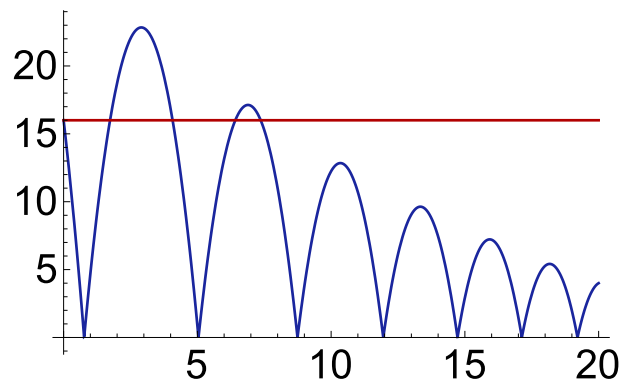


Figure 5: Sample trajectory of a bouncing ball dribbling with fast initial velocity $v < 0$

We could explore this relationship in more detail. But it is actually easier to infer this relationship by conducting a proof. So we modify (22) to simply assume $v = 0$ initially:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ [(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^*] (0 \leq x \wedge x \leq H) \quad (23)$$

Is $\text{d}\mathcal{L}$ formula (23) valid now? Or does it still have a counterexample?

Before you read on, see if you can find the answer for yourself.

It seems like all required assumptions have been identified to make the $\text{d}\mathcal{L}$ formula (23) valid so that the bouncing ball described in (23) satisfies the postcondition $0 \leq x \leq H$. But after so many failed starts and missing assumptions and requirements for the bouncing ball, it is a good idea to prove (23) once and for all beyond any doubt.

In order to be able to prove $\text{d}\mathcal{L}$ formula (23), however, we need to investigate how proving works. How can $\text{d}\mathcal{L}$ formulas be proved? And, since first-order formulas are $\text{d}\mathcal{L}$ formulas as well, one part of the question will be: how can first-order formulas be proved? How can real arithmetic be proved? How can requirements for the safety of CPS be identified systematically? All these questions will be answered in this course, but not all of them in this lecture.

In order to make sure we only need to worry about a minimal set of operators of $\text{d}\mathcal{L}$ for proving purposes, let's simplify (23) by getting rid of `if-then-else` (Exercise 12):

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ \left[(x' = v, v' = -g \& x \geq 0; (?x = 0; v := -cv \cup ?x \neq 0))^* \right] (0 \leq x \wedge x \leq H) \quad (24)$$

Observing the non-negligible difference between the original conjecture (19) and the revised and improved conjecture (24), leads us to often adopt the following principle.

Note 8 (Principle of Cartesian Doubt). *In 1641, René Descartes suggested an attitude of systematic doubt where he would be skeptical about the truth of all beliefs until he found reason that they were justified. This principle is now known as Cartesian Doubt or skepticism.*

We will have perfect justifications: proofs. But until we have found proof, it is often helpful to adopt the principle of Cartesian Doubt in a very weak and pragmatic form. Before setting out on the journey to prove a conjecture, we first scrutinize it to see if we can find a counterexample that would make it false. For such a counterexample will not only save us a lot of misguided effort in trying to prove a false conjecture, but also helps us identify missing assumptions in conjectures and justifies the assumptions to be necessary. Surely, if, without assumption A , a counterexample to a conjecture exists, then A must be rather necessary.

11 Intermediate Conditions for CPS

Before proceeding any further with ways of proving $\text{d}\mathcal{L}$ formulas, let's simplify (24) grotesquely by removing the loop:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ [x' = v, v' = -g \& x \geq 0; (?x = 0; v := -cv \cup ?x \neq 0)] (0 \leq x \wedge x \leq H) \quad (25)$$

Removing the loop clearly changes the behavior of the bouncing ball. It no longer bounces particularly well. All it can do now is fall and, if it reaches the floor, have its

velocity reverted without actually climbing back up. So if we manage to prove (25), we certainly have not shown the actual $\text{d}\mathcal{L}$ formula (24). But it's a start, because the behavior modeled in (25) is a part of the behavior of (24). So it is useful (and easier) to understand (25) first.

The $\text{d}\mathcal{L}$ formula (25) has a number of assumptions $0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0$ that can be used during the proof. It claims that the postcondition $0 \leq x \wedge x \leq H$ holds after all runs of the HP in the $[\cdot]$ modality. The top-level operator in the modality of (25) is a sequential composition $(;)$, for which we need to find a proof argument.⁴

The HP in (25) follows a differential equation first and then, after the sequential composition $(;)$, proceeds to run a discrete program $(?x = 0; v := -cv \cup ?x \neq 0)$. Depending on how long the HP follows its differential equation, the intermediate state after the differential equation and before the discrete program will be rather different.

Note 9 (Intermediate states of sequential compositions). *This phenomenon happens in general for sequential compositions $\alpha; \beta$. The first HP α may reach a whole range of states, which represent intermediate states for the sequential composition $\alpha; \beta$, i.e. states that are final states for α and initial states for β . The intermediate states of $\alpha; \beta$ are the states μ in the semantics $\rho(\alpha; \beta)$ from Lecture 3:*

$$\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha) = \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\}$$

Can we find a way of summarizing what all intermediate states between the differential equation and the discrete program of (25) have in common? They differ by how long the CPS has followed the differential equation.

If the system has followed the differential equation of (25) for time t , then the resulting velocity $v(t)$ at time t and height $x(t)$ at time t will be

$$v(t) = -gt, x(t) = H - \frac{g}{2}t^2 \tag{26}$$

This answer can be found by integrating or solving the differential equations. This knowledge (26) is useful but it is not (directly) clear how to use it to describe what all intermediate states have in common, because the time t in (26) is not available as a variable in the HP (25).⁵ Can the intermediate states be described by a relation of the variables that (unlike t) are actually in the system? That is, an (arithmetic) formula relating x, v, g, H ?

Before you read on, see if you can find the answer for yourself.

⁴ The way we proceed here to prove (25) is actually not the recommended way. Later on, we will see a much easier way. But it is instructive to understand the more verbose approach we take first. This also prepares us for the challenges that lie ahead when proving properties of loops.

⁵ Following these thoughts a bit further reveals how (26) can actually be used perfectly well to describe intermediate states when changing the HP (25) a little bit. But working with solutions is still not the way that gets us to the goal the quickest, usually.

One way of producing a relation from (26) is to get the units aligned and get rid of time t . Time drops out of the “equation” when squaring the identity for velocity:

$$v(t)^2 = g^2 t^2, \quad x(t) = H - \frac{g}{2} t^2$$

and multiplying the identity for position by $2g$:

$$v(t)^2 = g^2 t^2, \quad 2gx(t) = 2gH - 2\frac{g^2}{2} t^2$$

Then substituting the first equation into the second yields

$$2gx(t) = 2gH - v(t)^2$$

This equation does not depend on time t , so we expect it to hold after all runs of the differential equation irrespective of t :

$$2gx = 2gH - v^2 \tag{27}$$

We conjecture the intermediate condition (27) to hold in the intermediate state of the sequential composition in (25). In order to prove (25) we can decompose our reasoning into two parts. The first part will prove that the intermediate condition (27) holds after all runs of the first differential equation. The second part will assume (27) to hold and prove that all runs of the discrete program in (25) from any state satisfying (27) satisfy the postcondition $0 \leq x \wedge x \leq H$.

Note 10 (Intermediate conditions as contracts for sequential composition). For a HP that is a sequential composition $\alpha; \beta$ an intermediate condition is a formula that characterizes the intermediate states in between HP α and β . That is, for a dL formula

$$A \rightarrow [\alpha; \beta] B$$

an intermediate condition is a formula E such that the following dL formulas are valid:

$$A \rightarrow [\alpha] E \quad \text{and} \quad E \rightarrow [\beta] B$$

The first dL formula expresses that intermediate condition E characterizes the intermediate states accurately, i.e. E actually holds after all runs of HP α from states satisfying A . The second dL formula says that the intermediate condition E characterizes intermediate states well enough, i.e. E is all we need to know about a state to conclude that all runs of β end up in B . That is, from all states satisfying E (in particular from those that result by running α from a state satisfying A), B holds after all runs of β .

For proving (25), we conjecture that (27) is an intermediate condition, which requires us to prove the following two dL formulas:

$$\begin{aligned} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 &\rightarrow [x' = v, v' = -g \ \& \ x \geq 0] 2gx = 2gH - v^2 \\ 2gx = 2gH - v^2 &\rightarrow [?x = 0; v := -cv \cup ?x \neq 0] (0 \leq x \wedge x \leq H) \end{aligned} \tag{28}$$

Let's focus on the latter formula. Do we expect to be able to prove it? Do we expect it to be valid?

Before you read on, see if you can find the answer for yourself.

The second formula of (28) claims that $0 \leq x$ holds after all runs of $?x = 0; v := -cv \cup ?x \neq 0$ from all states that satisfy $2gx = 2gH - v^2$. That is a bit much to hope for, however, because $0 \leq hx$ is not even ensured in the precondition of this second formula. So the second formula of (28) is not valid. How can this problem be resolved? By adding $0 \leq x$ into the intermediate condition, thus, requiring us to prove:

$$\begin{aligned} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow [x' = v, v' = -g \ \& \ x \geq 0] (2gx = 2gH - v^2 \wedge x \geq 0) \\ 2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [?x = 0; v := -cv \cup ?x \neq 0] (0 \leq x \wedge x \leq H) \end{aligned} \quad (29)$$

Proving the first formula in (29) requires us to handle differential equations, which we will get to later. The second formula in (29) is the one whose proof is discussed first.

12 A Proof of Choice

The second formula in (29) has a nondeterministic choice (\cup) as the top-level operator in its $[\cdot]$ modality. How can we prove a formula of the form

$$A \rightarrow [\alpha \cup \beta]B \quad (30)$$

Recalling its semantics from [Lecture 3](#),

$$\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$$

HP $\alpha \cup \beta$ has two possible behaviors. It could run as HP α does or as β does. And it is chosen nondeterministically which of the two behaviors happens. Since the behavior of $\alpha \cup \beta$ could be either α or β , proving (30) requires proving B to hold after α and after β . More precisely, (30) assumes A to hold initially, otherwise (30) is vacuously true. Thus, proving (30) allows us to assume A and requires us to prove that B holds after all runs of α (which is permitted behavior for $\alpha \cup \beta$) and to prove that, assuming A holds initially, that B holds after all runs of β (which is also permitted behavior of $\alpha \cup \beta$).

Note 11 (Proving choices). *For a HP that is a nondeterministic choice $\alpha \cup \beta$, we can prove*

$$A \rightarrow [\alpha \cup \beta]B$$

by proving the following dL formulas:

$$A \rightarrow [\alpha]B \quad \text{and} \quad A \rightarrow [\beta]B$$

Using these thoughts on the second formula of (29), we could prove that formula if we would manage to prove both of the following dL formulas:

$$\begin{aligned} 2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [?x = 0; v := -cv] (0 \leq x \wedge x \leq H) \\ 2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [?x \neq 0] (0 \leq x \wedge x \leq H) \end{aligned} \quad (31)$$

13 Proofs of Tests

Consider the second formula of (31). Proving it requires us to understand how to handle a test $?H$ in a modality $[?H]$. The semantics of a test $?H$ from [Lecture 3 on Choice & Control](#)

$$\rho(?H) = \{(\nu, \nu) : \nu \models H\} \quad (32)$$

says that a test $?H$ completes successfully without changing the state in any state ν in which H holds (i.e. $\nu \models H$) and fails to run in all other states (i.e. where $\nu \not\models H$). How can we prove a formula with a test:

$$A \rightarrow [?H]B \quad (33)$$

This formula expresses that from all initial states satisfying A all runs of $?H$ reach states satisfying B . When is there a run of $?H$ at all? There is a run from state ν if and only if H holds in ν . So the only cases to worry about those initial states that satisfy H as, otherwise, the HP in (33) cannot execute at all by fails miserably so that the run is discarded. Hence, we get to assume H holds, as the HP $?H$ does not otherwise execute. In all states that the HP $?H$ reaches from states satisfying A , (33) conjectures that B holds. Now, by (32), the final states that $?H$ reaches are the same as the initial state (as long as they satisfy H so that HP $?H$ can be executed at all). That is, postcondition B needs to hold in all states from which $?H$ runs (i.e. that satisfy H) and that satisfy the precondition A . So (33) can be proved by proving

$$A \wedge H \rightarrow B$$

Note 12 (Proving tests). *For a HP that is a test $?H$, we can prove*

$$A \rightarrow [?H]B$$

by proving the following dL formula:

$$A \wedge H \rightarrow B$$

Using this for the second formula of (31), Note 12 reduces proving the second formula of (31)

$$2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [?x \neq 0] (0 \leq x \wedge x \leq H)$$

to proving

$$2gx = 2gH - v^2 \wedge x \geq 0 \wedge x \neq 0 \rightarrow 0 \leq x \wedge x \leq H \quad (34)$$

Now we are left with arithmetic that we need to prove. Proofs for arithmetic and propositional logical operators such as \wedge and \rightarrow will be considered in a later lecture. For now, we notice that the formula $0 \leq x$ in the right-hand side of \rightarrow is justified by assumption $x \geq 0$ if we flip the inequality around. And that $x \leq H$ does not exactly have a justification in (34), because we lost the assumptions about H somewhere.

How could that happen? We used to know $x \leq H$ in (25). We also still knew about it in the first formula of (29). But we let it disappear from the second formula of (29), because we chose an intermediate condition that was too weak when constructing (29).

This is a common problem in trying to prove properties of CPS or of any other mathematical statements. One of our intermediate steps might have been too weak, so that our attempt of proving it fails and we need to revisit how we got there. For sequential compositions, this is actually a nonissue as soon as we move on (in the next lecture) to a proof technique that is more useful than the intermediate conditions from Note 10. But similar difficulties can arise in other parts of proof attempts.

In this case, the fact that we lost $x \leq H$ can be fixed by including it in the intermediate conditions, because it can be shown to hold after the differential equation. Other crucial assumptions have also suddenly disappeared in our reasoning. An extra assumption $1 > c \geq 0$, for example, is crucially needed to justify the first formula of (31). It is easier to see why that particular assumption can be added to the intermediate contract without changing the argument much. The reason is that c never ever changes during the system run.

Note 13. *It is very difficult to come up with bug-free code. Just thinking about your assumptions really hard does not ensure correctness, but we can gain confidence that our system does what we want it to by proving that certain properties are satisfied.*

Changing the assumptions and arguments in a hybrid program around during the search for a proof of safety is something that happens frequently. It is easy to make subtle mistakes in informal arguments such as I need to know C here and I would know C if I had included it here or there, so now I hope the argument holds. This is one of many reasons why we are better off if our CPS proofs are rigorous, because we would rather not end up in trouble because of a subtle flaw in a correctness argument. A rigorous, formal proof calculus for differential dynamic logic (dL) will help us avoid the pitfalls of informal arguments. The theorem prover KeYmaera that you will use in this course implements a proof calculus for dL.

A related observation from our informal arguments in this lecture is that we desperately need a way to keep an argument consistent as a single argument justifying one conjecture. Quite the contrary to the informal loose threads of argumentation we have pursued in this lecture for the sake of developing an intuition. Consequently, we will investigate what holds all arguments together and what constitutes an actual proof in subsequent lectures. A proof in which the relationship of premises to conclusions via proof steps is rigorous.

Moreover, there's two loose ends in our arguments. For one, the differential equation in (29) is still waiting for an argument that could help us prove it. Also, the assignment in (31) still needs to be handled and its sequential composition needs an intermediate contract (Exercise 14). Both will be pursued in the next lecture, where we move to a systematic and rigorous reasoning style for CPS.

Note 14 (Operators and (informal) meaning in differential dynamic logic ($\text{d}\mathcal{L}$)).

$\text{d}\mathcal{L}$	Operator	Meaning
$\theta = \eta$	<i>equals</i>	true iff values of θ and η are equal
$\theta \geq \eta$	<i>equals</i>	true iff value of θ greater-or-equal to η
$\neg\phi$	<i>negation / not</i>	true if ϕ is false
$\phi \wedge \psi$	<i>conjunction / and</i>	true if both ϕ and ψ are true
$\phi \vee \psi$	<i>disjunction / or</i>	true if ϕ is true or if ψ is true
$\phi \rightarrow \psi$	<i>implication / implies</i>	true if ϕ is false or ψ is true
$\phi \leftrightarrow \psi$	<i>bi-implication / equivalent</i>	true if ϕ and ψ are both true or both false
$\forall x \phi$	<i>universal quantifier / for all</i>	true if ϕ is true for all values of variable x
$\exists x \phi$	<i>existential quantifier / exists</i>	true if ϕ is true for some values of variable x
$[\alpha]\phi$	$[\cdot]$ <i>modality / box</i>	true if ϕ is true after all runs of HP α
$\langle\alpha\rangle\phi$	$\langle\cdot\rangle$ <i>modality / diamond</i>	true if ϕ is true after at least one run of HP α

14 Summary

This lecture introduced differential dynamic logic ($\text{d}\mathcal{L}$), whose operators and their informal meaning is summarized in Note 14.

This lecture also featured first reasoning aspects for CPS. But reasoning for CPS will be investigated systematically in subsequent lectures, one operator at a time, which establishes separate reasoning principles for each operator, rather than proceeding in the more ad-hoc style of this lecture along an example. For future lectures, we should keep the bouncing ball example and its surprising subtleties in mind, though.

Exercises

Exercise 1. Show that (15) is valid. It is okay to focus only on this case, even though the argument is more general, because the following $\text{d}\mathcal{L}$ formula is valid for any hybrid program α :

$$[\alpha]F \wedge [\alpha]G \leftrightarrow [\alpha](F \wedge G)$$

Exercise 2. Let A, B be $\text{d}\mathcal{L}$ formulas. Suppose $A \leftrightarrow B$ is valid and A is valid. Is B valid? Prove or disprove.

Exercise 3. Let A, B be $\text{d}\mathcal{L}$ formulas. Suppose $A \leftrightarrow B$ is true in state ν and A is true in state ν . That is, $\nu \models A \leftrightarrow B$ and $\nu \models A$. Is B true in state ν ? Prove or disprove. Is B valid? Prove or disprove.

Exercise 4. Let α be an HP. Let ν be a state with $\nu \not\models \phi$. Does $\nu \not\models [\alpha^*]\phi$ hold? Prove or disprove.

Exercise 5. Let α be an HP. Let ν be a state with $\nu \models \phi$. Does $\nu \models [\alpha^*]\phi$ hold? Prove or disprove.

Exercise 6. Let α be an HP. Let ν be a state with $\nu \models \phi$. Does $\nu \models \langle \alpha^* \rangle \phi$ hold? Prove or disprove.

Exercise 7. Suppose you have a HP α with a CPS contract using multiple preconditions A_1, \dots, A_n and multiple postconditions B_1, \dots, B_m :

```

@requires( $A_1$ )
@requires( $A_2$ )
:
@requires( $A_n$ )
@ensures( $B_1$ )
@ensures( $B_2$ )
:
@ensures( $B_m$ )
 $\alpha$ 

```

How can this CPS contract be expressed in a $\text{d}\mathcal{L}$ formula? If there are multiple alternatives on how to express it, discuss the advantages and disadvantages of each option.

Exercise 8. For each of the following $\text{d}\mathcal{L}$ formulas, determine if they are valid, satisfiable, and/or unsatisfiable:

1. $[?x \geq 0]x \geq 0$.
2. $[?x \geq 0]x \leq 0$.
3. $[?x \geq 0]x < 0$.
4. $[?true]true$.
5. $[?true]false$.
6. $[?false]true$.
7. $[?false]false$.
8. $[x' = 1 \ \& \ true]true$.
9. $[x' = 1 \ \& \ true]false$.
10. $[x' = 1 \ \& \ false]true$.
11. $[x' = 1 \ \& \ false]false$.
12. $[(x' = 1 \ \& \ true)^*]true$.
13. $[(x' = 1 \ \& \ true)^*]false$.

14. $[(x' = 1 \ \& \ \text{false})^*] \text{true}.$

15. $[(x' = 1 \ \& \ \text{false})^*] \text{false}.$

Exercise 9. For each of the following dL formulas, determine if they are valid, satisfiable, and/or unsatisfiable:

1. $x > 0 \rightarrow [x' = 1]x > 0$
2. $x > 0 \wedge \langle x' = 1 \rangle x < 0$
3. $x > 0 \rightarrow [x' = -1]x < 0$
4. $x > 0 \rightarrow [x' = -1]x \geq 0$
5. $x > 0 \rightarrow [(x := x + 1)^*]x > 0$
6. $x > 0 \rightarrow [(x := x + 1)^*]x > 1$
7. $[x := x^2 + 1; x' = 1]x > 0.$
8. $[(x := x^2 + 1; x' = 1)^*]x > 0.$
9. $[(x := x + 1; x' = -1)^*]; ?x > 0; x' = 2]x > 0$

Exercise 10. For each $j, k \in \{\text{satisfiable}, \text{unsatisfiable}, \text{valid}\}$ answer whether there is a formula that is j but not k . Also answer for each such j, k whether there is a formula that is j but its negation is not k . Briefly justify each answer.

Exercise 11. What would happen with the bouncing ball if $c < 0$? Consider a variation of the arguments in Sect. 10 where instead of the assumption in (21), you assume $c < 0$. Is the formula valid? What would happen with a bouncing ball of damping $c = 1$?

Exercise 12. We went from (23) to (24) by removing an if-then-else. Explain how this works and justify why it is okay to do this transformation. It is okay to focus only on this case, even though the argument is more general.

*Exercise 13 (**).* Sect. 11 used a mix of a systematic and ad-hoc approach for producing an intermediate condition that was based on solving and combining differential equations. Can you think of a more systematic rephrasing?

Exercise 14 ().* Find an intermediate condition for proving the first formula in (31). The proof of the resulting formulas is complicated significantly by the fact that assignments have not yet been discussed in this lecture. Can you find a way of proving the resulting formulas before the next lecture develops how to handle assignments?

*Exercise 15 (***)*. Before looking at subsequent lectures: How could you prove the formula (29), which involves a differential equation?

References

- [Asi42] Isaac Asimov. Runaround, 1942.
- [DLTT13] Patricia Derler, Edward A. Lee, Stavros Tripakis, and Martin Törngren. Cyber-physical system design contracts. In Chenyang Lu, P. R. Kumar, and Radu Stoleru, editors, *ICCPs*, pages 109–118. ACM, 2013.
- [Flo67] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*, volume 19, pages 19–32, Providence, 1967. AMS.
- [Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Log11] Francesco Logozzo. Practical verification for the working programmer with codecontracts and abstract interpretation - (invited talk). In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *LNCS*, pages 19–22. Springer, 2011. doi:[10.1007/978-3-642-18275-4_3](https://doi.org/10.1007/978-3-642-18275-4_3).
- [Mey92] Bertrand Meyer. Applying “design by contract”. *Computer*, 25(10):40–51, October 1992.
- [MP14] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *RV*, volume 8734 of *LNCS*, pages 199–214. Springer, 2014. doi:[10.1007/978-3-319-11164-3_17](https://doi.org/10.1007/978-3-319-11164-3_17).
- [PCL11] Frank Pfenning, Thomas J. Cortina, and William Lovas. Teaching imperative programming with contracts at the freshmen level. 2011.
- [Pla07] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007. doi:[10.1007/978-3-540-73099-6_17](https://doi.org/10.1007/978-3-540-73099-6_17).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In *LICS [LIC12]*, pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:[1205.4788](https://arxiv.org/abs/1205.4788).
- [Pla12c] André Platzer. Logics of dynamical systems. In *LICS [LIC12]*, pages 13–24. doi:[10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [Pla13] André Platzer. Teaching CPS foundations with contracts. In *CPS-Ed*, pages 7–10, 2013.

- [PQ08] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. [doi:10.1007/978-3-540-71070-7_15](https://doi.org/10.1007/978-3-540-71070-7_15).
- [Pra76] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In *FOCS*, pages 109–121. IEEE, 1976.
- [XJC09] Dana N. Xu, Simon L. Peyton Jones, and Koen Claessen. Static contract checking for haskell. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, pages 41–52. ACM, 2009. [doi:10.1145/1480881.1480889](https://doi.org/10.1145/1480881.1480889).

Lecture Notes on Dynamical Systems & Dynamic Axioms

[André Platzer](#)

Carnegie Mellon University
Lecture 5

1 Introduction

[Lecture 4 on Safety & Contracts](#) demonstrated how useful and crucial CPS contracts are for CPS. Their role and understanding goes beyond dynamic testing, though. In CPS, proven CPS contracts are infinitely more valuable than dynamically tested contracts, because dynamical tests of contracts at runtime of a CPS generally leave open very little flexibility for reacting to them in any safe way. After all, the failure of a contract indicates that some safety condition that was expected to hold is not longer true. Unless provably sufficient safety margins and fallback plans remain, the system is already in trouble then.¹

Consequently, CPS contracts really shine in relation to how they are proved for CPS. Understanding how to prove CPS contracts requires us to understand the dynamical effects of hybrid programs in more detail. This deeper understanding of the effects of hybrid program statements is not only useful for conducting proofs but also for developing and sharpening our intuition about hybrid programs for CPS. This phenomenon illustrates a more general point that proof and effect (and/or meaning) are intimately linked and that truly understanding effect is ultimately the same as, as well as a prerequisite to, understanding how to prove properties of that effect [[Pla12c](#), [Pla12a](#), [Pla10](#)]. You may have seen this point demonstrated already in other courses from the Principles of Programming Languages group at CMU, but it will shine in today's lecture.

The route that we choose to get to this level of understanding is one that involves a closer look at dynamical systems and Kripke models, or rather, the effect that hybrid programs have on them. This will enable us to devise authoritative proof principles for

¹Although, in combination with formal verification, the Simplex architecture exploits the relationship of dynamic contracts for safety purposes [[SKSC98](#)]. ModelPlex, which is based on differential dynamic logic, lifts this observation to a fully verified link from verified models to CPS executions [[MP14](#)].

differential dynamic logic and hybrid programs [Pla12c, Pla12a, Pla10, Pla08]. While there are many more interesting things to say about dynamical systems and Kripke structures, this lecture will limit information to the truly essential parts that are crucial right now and leave more elaboration for later lectures.

More information can be found in [Pla12b, Pla12c] as well as [Pla10, Chapter 2.3].

The focus of today's lecture is on a systematic development of the basic reasoning principles for cyber-physical systems. The goal is to cover all cyber-physical systems by identifying one fundamental reasoning principle for each of the operators of differential dynamic logic and, specifically, its hybrid programs. Once we have a (suitably complete) reasoning principle for each of the operators, the basic idea is that any arbitrary cyber-physical system can be analyzed by just combining the various reasoning principles with one another, compositionally, by inspecting one operator at a time. With enough understanding, this guiding principle will, indeed, ultimately succeed [Pla12a, Pla14]. It will, however, take significantly more than one lecture to get there. So, today's lecture will settle for a systematic development of the reasoning principles for the more elementary operators in hybrid programs, leaving a detailed development of the others to later lectures.

This lecture is of central significance for the Foundations of Cyber-Physical Systems. The most important learning goals of this lecture are:

Modeling and Control: We will understand the core principles behind CPS by understanding analytically and semantically how cyber and physical aspects are integrated and interact in CPS. This lecture will also begin to explicitly relate discrete and continuous systems, which will ultimately lead to a fascinating view on understanding hybridness [Pla12a].

Computational Thinking: This lecture is devoted to the core aspects of reasoning rigorously about CPS models, which is critical to getting CPS right. CPS designs can be flawed for very subtle reasons. Without sufficient rigor in their analysis it can be impossible to spot the flaws, and even more challenging to say for sure whether and why a design is no longer faulty. This lecture systematically develops one reasoning principle for each of the operators of hybrid programs. This lecture begins an *axiomatization* of differential dynamic logic $d\mathcal{L}$ [Pla12c, Pla12a] to lift $d\mathcal{L}$ from a specification language to a verification language for CPS.

CPS Skills: We will develop a deep understanding of the semantics of CPS models by carefully relating their semantics to their reasoning principles and aligning them in perfect unison. This understanding will also enable us to develop a better intuition for the operational effects involved in CPS.

Note 1 (Logical trinity). *The concepts developed in this lecture illustrate the more general relation of syntax (which is notation), semantics (what carries meaning), and axiomat-ics (which internalizes semantic relations into universal syntactic transformations). These concepts and their relations jointly form the significant logical trinity of syntax, seman-tics, and axiomat-ics.*

2 A Proof of Choice (Continued)

Recall the bouncing ball from [Lecture 4 on Safety & Contracts](#), with repetition removed to simplify the subsequent discussion for illustration purposes:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow [x' = v, v' = -g \ \& \ x \geq 0; (?x = 0; v := -cv \cup ?x \neq 0)] (0 \leq x \wedge x \leq H) \quad (1)$$

In order to try to prove the above formula, we have convinced ourselves with a num-ber of steps of argumentation that we should try to prove the following two formulas (and many others):

$$\begin{aligned} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow [x' = v, v' = -g \ \& \ x \geq 0] (2gx = 2gH - v^2 \wedge g > 0) \\ 2gx = 2gH - v^2 \wedge g > 0 \rightarrow [?x = 0; v := -cv \cup ?x \neq 0] (0 \leq x \wedge x \leq H) \end{aligned} \quad (2)$$

In our attempt of proving the latter formula, we used the following principle:

Note 2 (Proving choices). *For a HP that is a nondeterministic choice $\alpha \cup \beta$, we can prove*

$$A \rightarrow [\alpha \cup \beta]B \quad (3)$$

by proving the following dL formulas:

$$A \rightarrow [\alpha]B \quad \text{and} \quad A \rightarrow [\beta]B$$

Note 3 (Proving choices: proof-rule style). *Note 2 is captured more concisely in the following proof rule:*

$$(R1) \frac{A \rightarrow [\alpha]B \quad A \rightarrow [\beta]B}{A \rightarrow [\alpha \cup \beta]B}$$

If we can prove all premises (above rule bar) of a proof rule, then that proof rule infers the conclusion (below rule bar).

Alas, the way we have been using proof rules so far is the other way around. We had been looking at a formula such as the second formula of (2) that has the shape of the conclusion of a rule such as R1. And then we went on trying to prove the premises of that proof rule instead. This conclusion-to-premise style of using our proof rules is perfectly acceptable and useful as well. Should we ever succeed in proving the premises of R1, then that proof rule would allow us to infer its conclusion too. In this way, proof rules are even useful in directing us at which formulas we should try to prove next: the premises of the instantiation of that rule.

Using these thoughts on the second formula of (2), we could prove that formula using proof rule R1 if we would manage to prove both of its premises, which, in this instance, are the following dL formulas:

$$\begin{aligned} 2gx = 2gH - v^2 \wedge g > 0 &\rightarrow [?x = 0; v := -cv] (0 \leq x \wedge x \leq H) \\ 2gx = 2gH - v^2 \wedge g > 0 &\rightarrow [?x \neq 0] (0 \leq x \wedge x \leq H) \end{aligned} \quad (4)$$

Before proceeding with proofs of (4), revisit the reasoning that led to the principle in Note 3. We said that (3) can be justified by proving that, when assuming A , all runs of α lead to states satisfying B and all runs of β lead to B states. Is that argument reflected directly in Note 3?

Kind of, but not quite, because there is a minor difference. Our informal argument assumed A once and concluded both $[\alpha]B$ and $[\beta]B$ from A . The principle captured in Note 3 assumes A to prove $[\alpha]B$ and then, separately, assumes A again to prove $[\beta]B$. These two arguments are clearly closely related, but still slightly different. Can we formalize and follow the original argument directly somehow? Or is Note 3 our only chance?

Following the original argument, we would argue that (3) holds by proving

$$A \rightarrow ([\alpha]B \wedge [\beta]B)$$

or, since the parentheses are superfluous according to the usual precedence rules:

$$A \rightarrow [\alpha]B \wedge [\beta]B \quad (5)$$

Is there a direct way how we can justify going from (3) to (5)? Preferably one that simultaneously justifies going from (3) to the formulas identified in Note 3 as well.

These considerations will take us to a more general and more elegant proof principle than R1, to a more refined understanding of the behavior of nondeterministic choices, and to a way of justifying proof rules as being sound, without which we should never be using them in the first place.

3 Dynamic Axioms for Nondeterministic Choices

Recall the semantics of nondeterministic choices from [Lecture 3 on Choice & Control](#):

$$\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta) \quad (6)$$

Remember that $\rho(\alpha)$ is a reachability relation on states, where $(\nu, \omega) \in \rho(\alpha)$ iff HP α can run from state ν to state ω . Let us illustrate graphically what (6) means:

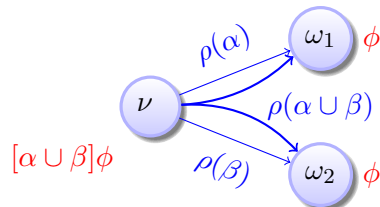


Figure 1: Illustration of the transition semantics of $\alpha \cup \beta$, which allows $\alpha \cup \beta$ to make any transitions that either α or that β could do on their own.

According to $\rho(\alpha)$, a number of states ω_i are reachable by running HP α from some initial state ν .² According to $\rho(\beta)$, a number of (possibly other) states ω_i are reachable by running HP β from the same initial state ν . By the semantic equation (6), running $\alpha \cup \beta$ from ν can give us any of those possible outcomes. And there was nothing special about the initial state ν . The same principle holds for all other states as well.

Note 4 (\cup). The nondeterministic choice $\alpha \cup \beta$ can lead to exactly the states to which either α could take us or to which β could take us or to which both could lead. The dynamic effect of a nondeterministic choice $\alpha \cup \beta$ is that running it at any time either results in a behavior of α or of β , nondeterministically. So both the behaviors of α and β are possible when running $\alpha \cup \beta$.

If we want to understand whether and where dL formula $[\alpha \cup \beta]\phi$ is true, we need to understand which states the modality $[\alpha \cup \beta]$ refers to. In which states does ϕ have to be true so that $[\alpha \cup \beta]\phi$ is true in state ν ?

By definition of the semantics, ϕ needs to be true in all states that $\alpha \cup \beta$ can reach according to $\rho(\alpha \cup \beta)$ from ν for $[\alpha \cup \beta]\phi$ to be true in ν . Referring to semantics (6) or looking at Fig. 1, shows us that this includes exactly all states that α can reach from ν according to $\rho(\alpha)$, hence $[\alpha]\phi$ has to be true in ν . And that it also includes all states that β can reach from ν , hence $[\beta]\phi$ has to be true in ν .

Consequently,

$$\nu \models [\alpha]\phi \quad \text{and} \quad \nu \models [\beta]\phi \quad (7)$$

are necessary conditions for

$$\nu \models [\alpha \cup \beta]\phi \quad (8)$$

²Fig. 1 only illustrates one such state ω_1 for visual conciseness. But ω_1 should be thought of as a generic representative for any such state that α can reach from the initial state ν in these figures.

That is, unless (7) holds, (8) cannot possibly hold. So (7) is necessary for (8). Are there any states missing? Are there any states that (8) would require to satisfy ϕ , which (7) does not already ensure to satisfy ϕ ? No, because, by (6), $\alpha \cup \beta$ does not admit any behavior that neither α nor β can exhibit. Hence (7) is also sufficient for (8), i.e. (7) implies (8). So (7) and (8) are equivalent.

Thus, when adopting a more logical language again, this justifies:

$$\nu \models [\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

This reasoning did not depend on the particular state ν but holds for all ν . Therefore,

$$\models [\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

Exciting! We have just proved our first axiom to be sound:

Lemma 1 ($[\cup]$ soundness). *The axiom of choice is sound, i.e. all its instances are valid:*

$$([\cup]) \quad [\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

Nondeterministic choices split into their alternatives in axiom $[\cup]$. From right to left: If all α runs lead to states satisfying ϕ (i.e., $[\alpha]\phi$ holds) and all β runs lead to states satisfying ϕ (i.e., $[\beta]\phi$ holds), then all runs of HP $\alpha \cup \beta$, which may choose between following α and following β , also lead to states satisfying ϕ (i.e., $[\alpha \cup \beta]\phi$ holds). The converse implication from left to right holds, because $\alpha \cup \beta$ can run all runs of α and all runs of β , so all runs of α (and of β) lead to states satisfying ϕ if that holds for all runs of $[\beta]\phi$.

Armed with this axiom $[\cup]$ at our disposal, we can now easily do a proof step from (3) to (5) just by invoking the equivalence that $[\cup]$ justifies. Let's elaborate. We want to prove:

$$A \rightarrow [\alpha \cup \beta]B \tag{3}$$

By $[\cup]$, or rather an instance of $[\cup]$ formed by using B for ϕ , we know:

$$[\alpha \cup \beta]B \leftrightarrow [\alpha]B \wedge [\beta]B \tag{9}$$

Since (9) is a valid equivalence, replacing the place where the left-hand side of (9) occurs in (3) by the right-hand side of (9) gives us a formula that is equivalent to (3):

$$A \rightarrow [\alpha]B \wedge [\beta]B \tag{5}$$

After all, according to the valid equivalence (9) justified by axiom $[\cup]$, (5) can be obtained from (3) just by replacing a formula with one that is equivalent.

Actually, stepping back, the same argument can be made to go from (5) to (3) instead of from (3) to (5). Both ways of using $[\cup]$ are perfectly fine. Although the direction that gets rid of the \cup operator tends to be much more useful, because it made progress

(getting rid of an HP operator). Yet axiom $[U]$ can also be useful in many more situations than rule R1. For example, if want to prove a $d\mathcal{L}$ formula

$$[\alpha \cup \beta]A \rightarrow B$$

where $[\alpha \cup \beta]$ is on the left-hand side of an implication, then axiom $[U]$ justifies that it is enough to prove the following $d\mathcal{L}$ formula instead:

$$[\alpha]A \wedge [\beta]A \rightarrow B$$

This inference cannot be justified with proof rule R1, but would need a separate proof rule such as

$$(R3) \frac{[\alpha]A \wedge [\beta]A \rightarrow B}{[\alpha \cup \beta]A \rightarrow B}$$

Yet, axiom $[U]$ simultaneously justifies both rules R1 and R3 and many other uses of splitting a boxed choice into a conjunction. Axiom $[U]$ is, thus, more fundamental.

A general principle behind the $d\mathcal{L}$ axioms is most noticeable in axiom $[U]$. All equivalence axioms of $d\mathcal{L}$ are primarily intended to be used by reducing the formula on the left to the (structurally simpler) formula on the right. Such a reduction symbolically decomposes a property of a more complicated system into separate properties of easier fragments α and β . While we might end up with more subproperties (like we do in the case of axiom $[U]$), each of them is structurally simpler, because it involves less program operators. This decomposition of systems into their fragments makes the problem tractable and is good for scalability purposes, because it reduces the study of complex systems successively to a study of many but smaller subsystems of which there are only finitely many. For these symbolic structural decompositions, it is very helpful that $d\mathcal{L}$ is a full logic that is closed under all logical operators, including disjunction and conjunction, for then both sides in $[U]$ are $d\mathcal{L}$ formulas again (unlike in Hoare logic [Hoa69]). This also turns out to be an advantage for computing invariants [PC08, PC09, Pla10, GP14], which will be discussed much later in this course.

4 Soundness

The definition of soundness in Lemma 1 was not specific to axiom $[U]$, but applies to all $d\mathcal{L}$ axioms.

Definition 2 (Soundness). An axiom is *sound* iff all its instances are valid.

From now on, every time we see a formula of the form $[\alpha \cup \beta]\phi$, we can remember that axiom $[U]$ knows a formula, namely $[\alpha]\phi \wedge [\beta]\phi$ that is equivalent to it. Of course, whenever we find a formula of the form $[\gamma \cup \delta]\psi$, we also remember that axiom $[U]$ knows a formula, namely $[\gamma]\psi \wedge [\delta]\psi$ that is equivalent to it, just by instantiation of axiom $[U]$. And the fact that axiom $[U]$ is sound ensures that we do not need to worry about whether such reasoning is correct every time we need it. Every instance of $[U]$ is

sound. Once we know that $[\cup]$ is sound, we can treat it syntactically and mechanically and apply it as needed, like a machine would.

But because soundness is such a big deal (a *conditio sine qua non* in logic, i.e., something without which logic could not be), we will prove soundness of $[\cup]$ carefully, even if we almost already did in our informal argument above.

Proof of Lemma 1. The fact that axiom $[\cup]$ is sound can be proved as follows. Since $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$, we have that $(\nu, \omega) \in \rho(\alpha \cup \beta)$ iff $(\nu, \omega) \in \rho(\alpha)$ or $(\nu, \omega) \in \rho(\beta)$. Thus, $\nu \models [\alpha \cup \beta]\phi$ iff $\nu \models [\alpha]\phi$ and $\nu \models [\beta]\phi$. \square

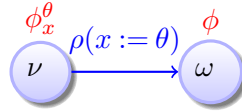
Why is soundness so critical? Well, because, without it, we could accidentally declare a system safe that is not in fact safe, which would defeat the whole purpose of verification and possibly put human lives in jeopardy when they are trusting their lives on an unsafe CPS. Unfortunately, soundness is actually not granted in all verification techniques for hybrid systems. But we will make it a point in this course to only ever use sound reasoning and scrutinizing all verification for soundness right away. Soundness is something that is comparably easy to establish in logic and proof approaches, because it localizes into the separate study of soundness of each of its axioms.

5 Dynamic Axioms for Assignments

Axiom $[\cup]$ allows us to understand and handle $[\alpha \cup \beta]$ properties. If we find similar axioms for all the other operators of hybrid programs, then we have a way of handling all other hybrid programs, too [Pla12a].

Consider discrete assignments. Recall from [Lecture 4 on Safety & Contracts](#) that:

$$\rho(x := \theta) = \{(\nu, \omega) : \omega = \nu \text{ except that } \llbracket x \rrbracket_\omega = \llbracket \theta \rrbracket_\nu\}$$



Lemma 3 ($[\text{:=}]$ soundness). *The assignment axiom is sound:*

$$([\text{:=}]) \quad [x := \theta]\phi(x) \leftrightarrow \phi(\theta)$$

Axiom $[\text{:=}]$ is Hoare's assignment rule. It uses substitutions to axiomatize discrete assignments. To show that $\phi(x)$ is true after a discrete assignment, axiom $[\text{:=}]$ shows that it has been true before, when substituting the affected variable x with its new value θ . That is, axiom $[\text{:=}]$ expresses that $\phi(x)$ is true after changing x around to the new value θ iff $\phi(\theta)$ was true before such change. Formula $\phi(\theta)$ is obtained from $\phi(x)$ by *substituting* θ for x at all occurrences of x (provided x does not occur in the scope of a quantifier or modality binding x or a variable of θ).

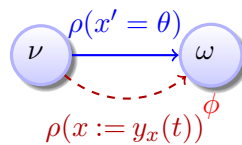
Note 8 (Bound variables). A modality containing $x :=$ or x' (outside the scope of tests $?H$ or evolution domain constraints) binds x , because it may change the value of x . A quantifier $\forall x$ or $\exists x$ also binds variable x .

Substitutions are defined as usual [Pla10, Chapter 2.5.1].

6 Dynamic Axioms for Differential Equations

Recall from [Lecture 3 on Choice & Control](#) that

$$\rho(x' = \theta \& H) = \{(\varphi(0), \varphi(r)) : \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \text{ for a solution } \varphi : [0, r] \rightarrow \mathcal{S} \text{ of any duration } r\}$$



One possible approach of proving properties of differential equations is to work with a solution if one is available (and expressible in the logic).

Lemma 4 (['] soundness). The solution axiom is sound:

$$([']) [x' = \theta]\phi \leftrightarrow \forall t \geq 0 [x := y(t)]\phi \quad (y'(t) = \theta)$$

In axiom ['], $y(\cdot)$ is the solution of the symbolic initial-value problem $y'(t) = \theta, y(0) = x$. Solution $y(\cdot)$ is unique since θ is smooth ([Lecture 2](#)). Given such a solution $y(\cdot)$, continuous evolution along differential equation $x' = \theta$ can be replaced by a discrete assignment $x := y(t)$ with an additional quantifier for the evolution time t . It goes without saying that variables like t are fresh in ['] and other axioms and proof rules. Notice that conventional initial-value problems are numerical with concrete numbers $x \in \mathbb{R}^n$ as initial values, not symbols x [Wal98]. This would not be enough for our purpose, because we need to consider all states in which the system could start, which may be uncountably many. That is why axiom ['] solves one symbolic initial-value problem, instead, because we could hardly solve uncountable many numerical initial-value problems.

Note 10 (Discrete vs. continuous dynamics). Notice something rather intriguing and peculiar about axiom [']. It relates a property of a continuous system to a property of a discrete system. The HP on the left-hand side describes a smoothly changing continuous process, while the right-hand side describes an abruptly, instantaneously changing discrete process. Still, their respective properties coincide, thanks to the time quantifier. This is the beginning of an astonishingly intimate relationship of discrete and continuous dynamics [Pla12a].

What we have so far about the dynamics of differential equations does not yet help us prove properties of differential equations with evolution domain constraints (a.k.a. continuous programs) $x' = \theta \ \& \ H$. It also does not yet tell us what to do if we cannot solve the differential equation or if the solution is too complicated. We will get to those matters in more detail in later lectures. Just briefly note that evolution domain constraints can be handled as well by adding a condition checking that the evolution domain was always true until the point in time of interest:

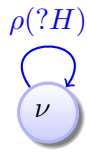
$$([']) \ [x' = \theta \ \& \ H]\phi \leftrightarrow \forall t \geq 0 \ ((\forall 0 \leq s \leq t \ [x := y(s)]H) \rightarrow [x := y(t)]\phi)$$

The effect of the additional constraint on H is to restrict the continuous evolution such that its solution $y(s)$ remains in the evolution domain H at all intermediate times $s \leq t$. This constraint simplifies to *true* if the evolution domain H is *true*, which makes sense, because there are no special constraints on the evolution (other than the differential equations) if the evolution domain is described by *true*, hence the full state space.

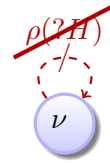
7 Dynamic Axioms for Tests

Recall from [Lecture 3 on Choice & Control](#) that

$$\rho(?H) = \{(\nu, \nu) : \nu \models H\}$$



if $\nu \models H$



if $\nu \not\models H$

Lemma 5 ([\[?\]](#) soundness). *The test axiom is sound:*

$$([?]) \ [?H]\phi \leftrightarrow (H \rightarrow \phi)$$

Tests in $[?H]\phi$ are proven by assuming that the test succeeds with an implication in axiom [\[?\]](#), because test $?H$ can only make a transition when condition H actually holds true. In states where test H fails, no transition is possible and the failed attempt to run the system is discarded. If no transition exists for an HP α , there is nothing to show for $[\alpha]\phi$ formulas, because their semantics requires ϕ to hold in all states reachable by running α , which is vacuously true if no states are reachable. From left to right, axiom [\[?\]](#) for dL formula $[?H]\phi$ assumes that formula H holds true (otherwise there is no transition and thus nothing to show) and shows that ϕ holds after the resulting no-op. The converse implication from right to left is by case distinction. Either H is false, then $?H$ cannot make a transition and there is nothing to show. Or H is true, but then also ϕ is true according to the implication.

8 Dynamic Axioms for Sequential Compositions

For sequential compositions $\alpha; \beta$, [Lecture 4 on Safety & Contracts](#) proposed the use of an intermediate condition E characterizing all intermediate states between α and β by way of the following proof rule:

Note 12 (Intermediate conditions as contracts for sequential compositions: proof-rule style). *Intermediate condition contracts for sequential compositions are captured more concisely in the following proof rule:*

$$(R8) \frac{A \rightarrow [\alpha]E \quad E \rightarrow [\beta]B}{A \rightarrow [\alpha; \beta]B}$$

This proof rule is useful, but it has one blatant annoyance compared to rule [R1](#) or let alone the simplicity and elegance of axiom [\[U\]](#). When using rule [R8](#) from the desired conclusion to the premises, it does not say how to choose the intermediate condition E . Using [R8](#) successfully requires us to find the right intermediate condition E , for if we don't, the proof won't succeed as we have seen in [Lecture 4](#). That is a bit much if we have to invent a useful intermediate condition E for every single sequential composition in a CPS.

Fortunately, differential dynamic logic provides a much better way that we also identify by investigating the dynamical system resulting from $\alpha; \beta$ and its induced Kripke structure. Recall from [Lecture 3 on Choice & Control](#) that

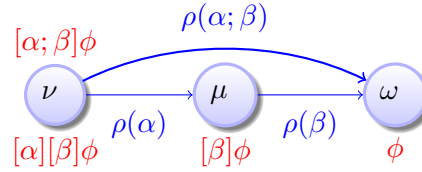
$$\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha) \stackrel{\text{def}}{=} \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\} \quad (10)$$

By its semantics, the dL formula $[\alpha; \beta]\phi$ is true in a state ν iff ϕ is true in all states that $\alpha; \beta$ can reach according to $\rho(\alpha; \beta)$ from ν , i.e. all those states for which $(\nu, \omega) \in \rho(\alpha; \beta)$. Which states are those? And how do they relate to the states reachable by α or by β alone? They do not relate to those in a way that is as direct as for axiom [\[U\]](#). But they still relate, and they do so by way of [\(10\)](#).

Postcondition ϕ has to be true in all states reachable by $\alpha; \beta$ from ν for $[\alpha; \beta]\phi$ to be true at ν . By [\(10\)](#), those are exactly the states ω to which we can get by running β from an intermediate state μ to which we have gotten from ν by running α . Thus, for $[\alpha; \beta]\phi$ to be true at ν it is necessary that ϕ holds in all states ω to which we can get by running β from an intermediate state μ to which we can get by running β from ν . Consequently, $[\alpha; \beta]\phi$ is only true at ν if $[\beta]\phi$ holds in all those intermediate states μ to which we can get from ν by running α . How do we characterize those states? And how can we then express these thoughts in a single logical formula of dL ?

Before you read on, see if you can find the answer for yourself.

If we want to express that $[\beta]\phi$ holds in all states μ to which we can get to from ν by running α , then that is exactly what truth of \mathbf{dL} formula $[\alpha][\beta]\phi$ at ν means, because this is the semantics of the modality $[\beta]$.



Consequently,

$$\nu \models [\alpha][\beta]\phi \rightarrow [\alpha; \beta]\phi$$

Reexamining our argument backwards, we see that the converse implication also holds

$$\nu \models [\alpha; \beta]\phi \rightarrow [\alpha][\beta]\phi$$

The same argument works for all ν , so both implications are even valid.

Lemma 6 ($[\cdot; \cdot]$ soundness). *The composition axiom is sound:*

$$([\cdot; \cdot]) \quad [\alpha; \beta]\phi \leftrightarrow [\alpha][\beta]\phi$$

Proof. Since $\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha)$, we have that $(\nu, \omega) \in \rho(\alpha; \beta)$ iff $(\nu, \mu) \in \rho(\alpha)$ and $(\mu, \omega) \in \rho(\beta)$ for some intermediate state μ . Hence, $\nu \models [\alpha; \beta]\phi$ iff $\mu \models [\beta]\phi$ for all μ with $(\nu, \mu) \in \rho(\alpha)$. That is $\nu \models [\alpha; \beta]\phi$ iff $\nu \models [\alpha][\beta]\phi$. \square

Sequential compositions are proven using nested modalities in axiom $[\cdot; \cdot]$. From right to left: If, after all α -runs, it is the case that all β -runs lead to states satisfying ϕ (i.e., $[\alpha][\beta]\phi$ holds), then all runs of the sequential composition $\alpha; \beta$ lead to states satisfying ϕ (i.e., $[\alpha; \beta]\phi$ holds), because $\alpha; \beta$ cannot go anywhere but following α through some intermediate state to running β . The converse implication uses the fact that if after all α -runs all β -runs lead to ϕ (i.e., $[\alpha][\beta]\phi$), then all runs of $\alpha; \beta$ lead to ϕ (that is, $[\alpha; \beta]\phi$), because the runs of $\alpha; \beta$ are exactly those that first do any α -run, followed by any β -run. Again, it is crucial that \mathbf{dL} is a full logic that considers reachability statements as modal operators, which can be nested, for then both sides in axiom $[\cdot; \cdot]$ are \mathbf{dL} formulas.

Axiom $[\cdot; \cdot]$ directly explains sequential composition $\alpha; \beta$ in terms of a structurally simpler formula, one with nested modal operators but simpler hybrid programs. Again, using axiom $[\cdot; \cdot]$ by reducing occurrences of its left-hand side to its right-hand side decomposes the formula into structurally simpler pieces, thereby making progress. One of the many ways of using axiom $[\cdot; \cdot]$ is, therefore, captured in the following proof rule:

$$(R10) \quad \frac{A \rightarrow [\alpha][\beta]B}{A \rightarrow [\alpha; \beta]B}$$

Comparing rule [R10](#) to rule [R8](#), the new rule [R10](#) is much easier to apply, because it does not require us to first identify and provide an intermediate condition E like rule [R8](#) would. It also does not branch into two premises, which helps keeping the proof lean. Is there a way of reuniting [R10](#) with [R8](#) by using the expressive power of $d\mathcal{L}$?

Before you read on, see if you can find the answer for yourself.

Yes, indeed, there is a very smart choice for the intermediate condition E that makes [R8](#) behave almost as the more efficient [R10](#) would. The clever choice $E \stackrel{\text{def}}{=} [\beta]B$:

$$\frac{A \rightarrow [\alpha][\beta]B \quad [\beta]B \rightarrow [\beta]B}{A \rightarrow [\alpha; \beta]B}$$

which trivializes the right premise and makes the left premise identical to that of [R10](#). Consequently, differential dynamic logic internalizes ways of expressing necessary and possible properties of hybrid programs and makes both first-class citizens in the logic. That cuts down on the amount of input that is needed when conducting proofs.

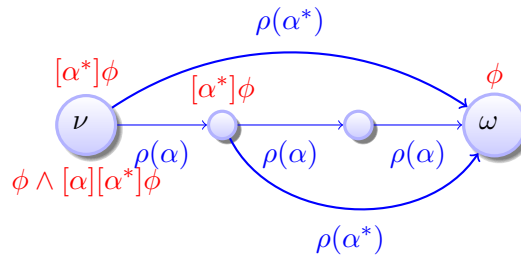
9 Unwinding Axioms for Loops

Recall from [Lecture 3 on Choice & Control](#) that

$$\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \quad \text{with} \quad \alpha^{n+1} \equiv \alpha^n; \alpha \text{ and } \alpha^0 \equiv ?\text{true}$$

How could we prove properties of loops such as $[\alpha^*]\phi$? Is there a way of reducing properties of loops to properties of simpler systems in similar ways as the other axioms of differential dynamic logic?

Before you read on, see if you can find the answer for yourself.



Lemma 7 ($[*]$ soundness). *The iteration axiom is sound:*

$$([*]) \quad [\alpha^*]\phi \leftrightarrow \phi \wedge [\alpha][\alpha^*]\phi$$

Axiom $[*]$ is the iteration axiom, which partially unwinds loops. It uses the fact that ϕ always holds after repeating α (i.e., $[\alpha^*]\phi$), if ϕ holds at the beginning (for ϕ holds after zero repetitions then), and if, after one run of α , ϕ holds after every number of repetitions of α , including zero repetitions (i.e., $[\alpha][\alpha^*]\phi$). So axiom $[*]$ expresses that $[\alpha^*]\phi$ holds iff ϕ holds immediately and after one or more repetitions of α .

The same axiom $[*]$ can be used to unwind loops $N \in \mathbb{N}$ times, which corresponds to Bounded Model Checking [CBRZ01]. If the formula is not valid, a bug has been found, otherwise N increases. An obvious issue with this simple approach is that we can never stop increasing N if the formula is actually valid, because we can never find a bug then. A later lecture will discuss proof techniques for repetitions based on loop invariants that are not subject to this issue. In particular, axiom $[*]$ is characteristically different from the other axioms discussed in this lecture. Unlike the other axioms, $[*]$ does not exactly get rid of the formula on the left-hand side. It just puts it in a different syntactic place, which does not sound like much progress.³

³ With a much more subtle and tricky analysis, it is possible to prove that $[*]$ still makes sufficient progress [Pla14]. But this is out of scope for our course.

10 A Proof of a Short Bouncing Ball

Now that we have understood so many axioms and proof rules, let us use them to prove the (single-hop) bouncing ball (1):

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\ [x' = v, v' = -g \ \& \ h \geq 0; (?x = 0; v := -cv \cup ?x \neq 0)] (0 \leq h \wedge h \leq H) \quad (1)$$

Before proceeding, let's modify the hybrid program ever so subtly in two ways so that there's no more evolution domains, because we have not (yet) understood how to prove differential equations with evolution domains:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\ [x' = v, v' = -g; (?x = 0; v := -cv \cup ?x \geq 0)] (0 \leq x \wedge x \leq H) \quad (11)$$

To fit things on the page easily, abbreviate

$$\begin{aligned} A_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \\ B_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H \\ (x'' = -g) &\stackrel{\text{def}}{=} (x' = v, v' = -g) \end{aligned}$$

With these abbreviations, (11) turns into

$$A_{x,v} \rightarrow [x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0)] B_{x,v}$$

Let there be proof for bouncing balls:

$$\begin{array}{l} A_{x,v} \rightarrow \forall t \geq 0 \left((H - \frac{g}{2}t^2 = 0 \rightarrow B_{H-\frac{g}{2}t^2, -c(-gt)}) \wedge (H - \frac{g}{2}t^2 \geq 0 \rightarrow B_{H-\frac{g}{2}t^2, -gt}) \right) \\ \text{[:=]} \frac{A_{x,v} \rightarrow \forall t \geq 0 [x := H - \frac{g}{2}t^2] ((x = 0 \rightarrow B_{x, -c(-gt)}) \wedge (x \geq 0 \rightarrow B_{x, -gt}))}{A_{x,v} \rightarrow \forall t \geq 0 [x := H - \frac{g}{2}t^2] [v := -gt] ((x = 0 \rightarrow B_{x, -cv}) \wedge (x \geq 0 \rightarrow B_{x, v}))} \\ \text{[:=]} \frac{A_{x,v} \rightarrow \forall t \geq 0 [x := H - \frac{g}{2}t^2; v := -gt] ((x = 0 \rightarrow B_{x, -cv}) \wedge (x \geq 0 \rightarrow B_{x, v}))}{A_{x,v} \rightarrow [x'' = -g] ((x = 0 \rightarrow B_{x, -cv}) \wedge (x \geq 0 \rightarrow B_{x, v}))} \\ \text{[?]} \frac{A_{x,v} \rightarrow [x'' = -g] ((x = 0 \rightarrow [v := -cv] B_{x, v}) \wedge (x \geq 0 \rightarrow B_{x, v}))}{A_{x,v} \rightarrow [x'' = -g] ([?x = 0] [v := -cv] B_{x, v} \wedge [?x \geq 0] B_{x, v})} \\ \text{[?], [?]} \frac{A_{x,v} \rightarrow [x'' = -g] ([?x = 0] [v := -cv] B_{x, v} \wedge [?x \geq 0] B_{x, v})}{A_{x,v} \rightarrow [x'' = -g] ([?x = 0; v := -cv] B_{x, v} \wedge [?x \geq 0] B_{x, v})} \\ \text{[?]} \frac{A_{x,v} \rightarrow [x'' = -g] ([?x = 0; v := -cv] B_{x, v} \wedge [?x \geq 0] B_{x, v})}{A_{x,v} \rightarrow [x'' = -g] [?x = 0; v := -cv \cup ?x \geq 0] B_{x, v}} \\ \text{[?]} \frac{A_{x,v} \rightarrow [x'' = -g] [?x = 0; v := -cv \cup ?x \geq 0] B_{x, v}}{A_{x,v} \rightarrow [x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0)] B_{x, v}} \end{array}$$

The dL axioms indicated on the left justify that the dL formulas in the two adjacent rows are equivalent. Since each step in this proof is justified by using a dL axiom, the conclusion at the very bottom of this derivation is proved if the premise at the very top can be proved, because truth then inherits from the top to the bottom. That premise

$$A_{x,v} \rightarrow \forall t \geq 0 \left((H - \frac{g}{2}t^2 = 0 \rightarrow B_{H-\frac{g}{2}t^2, -c(-gt)}) \wedge (H - \frac{g}{2}t^2 \geq 0 \rightarrow B_{H-\frac{g}{2}t^2, -gt}) \right)$$

expands out to a formula of first-order real arithmetic by expanding the abbreviations:

$$\begin{aligned} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\ \forall t \geq 0 \left(\left(H - \frac{g}{2}t^2 = 0 \rightarrow 0 \leq H - \frac{g}{2}t^2 \wedge H - \frac{g}{2}t^2 \leq H \right) \right. \\ \left. \wedge \left(H - \frac{g}{2}t^2 \geq 0 \rightarrow 0 \leq H - \frac{g}{2}t^2 \wedge H - \frac{g}{2}t^2 \leq H \right) \right) \end{aligned}$$

In this case, this remaining premise can be easily seen to be valid. The assumption $H - \frac{g}{2}t^2 = 0 \rightarrow \dots$ in the middle line directly implies the first conjunct that appears in its respective right-hand side

$$0 \leq H - \frac{g}{2}t^2 \wedge H - \frac{g}{2}t^2 \leq H$$

and reduces the remaining second conjunct to $0 \leq H$, which the assumption in the first line assumed ($0 \leq x = H$). Similarly, the assumption $H - \frac{g}{2}t^2 \geq 0$ of the last line implies the first conjunct of its right-hand side

$$0 \leq H - \frac{g}{2}t^2 \wedge H - \frac{g}{2}t^2 \leq H$$

and its second conjunct holds by assumption $g > 0$ from the first line and the real arithmetic fact that $t^2 \geq 0$.

How exactly first-order logic and first-order real arithmetic formulas such as this one can be proved in general, however, is an interesting topic for a later lecture. For now, we are happy to report that we have just formally verified our very first CPS. We have found a proof of (11). Exciting!

Okay, admittedly, the CPS we just verified was only a bouncing ball. And all we know about it now is that it won't fall through the cracks in the ground nor jump high up to the moon. But most big steps for mankind start with a small step by someone.

Yet, before we get too carried away in the excitement, we still need to remember that (11) is just a single-hop bouncing ball. So there's still an argument to be made about what happens if the bouncing ball repeats. And a rather crucial argument too, because bouncing balls let loose in the air tend not to jump any higher anyhow without hitting the ground first, which is where the model (11) stops prematurely, because it is missing a repetition. So let's put worrying about loops on the agenda for an upcoming lecture.

Yet, there's one more issue with the proof for the bouncing ball that we derived. It works in a somewhat undisciplined chaotic way, by using dL axioms all over the place. This liberal proof style can be useful for manual proofs and creative shortcuts. Albeit, since the dL axioms are sound, even such a liberal proof is a still proof. And liberal proofs could even be very creative. But liberal proofs are also somewhat unfocused and non-systematic, which makes them unreasonable for automation purposes and also tends to get people lost if the problems at hand are more complex than the single-hop bouncing ball. That is the reason why we will investigate more focused, more systematic, and more algorithmic proofs in the next lecture.

The other thing to observe is that the above proof, however liberal it might have been, already had more structure to it than we made explicit. This structure will be uncovered in the next lecture.

11 Summary

The differential dynamic logic axioms that we have seen in this lecture are summarized in Note 15. There are further axioms and proof rules of differential dynamic logic that later lectures will examine [Pla12c, Pla12a], but the reasoning principles and axioms identified here are fundamental and we will carry them with us throughout the whole course.

Note 15 (Summary of differential dynamic logic axioms from this lecture). *The following axioms of \mathbf{dL} are sound, i.e., all their instances are valid:*

$$[:=] \ [x := \theta]\phi(x) \leftrightarrow \phi(\theta)$$

$$[?] \ [?H]\phi \leftrightarrow (H \rightarrow \phi)$$

$$['] \ [x' = \theta]\phi \leftrightarrow \forall t \geq 0 \ [x := y(t)]\phi \quad (y'(t) = \theta)$$

$$[\cup] \ [\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

$$[:] \ [\alpha; \beta]\phi \leftrightarrow [\alpha][\beta]\phi$$

$$[*] \ [\alpha^*]\phi \leftrightarrow \phi \wedge [\alpha][\alpha^*]\phi$$

Exercises

Exercise 1. Explain why the subtle transformation from (1) to (11) was okay in this case.

Exercise 2. Identify which of the assumptions of (11) are actually required for the proof of (11). Which formulas could we have dropped from $0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0$ and still be able to prove

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow [x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0)](0 \leq x \wedge x \leq H)$$

Exercise 3. Develop possible axioms for differential equations with evolution domains similar to $[']$. That is, develop an axiom for $[x' = \theta \ \& \ H]\phi$. As in $[']$, you can assume to have a unique solution for the corresponding symbolic initial-value problem.

Exercise 4. Would the following be a sound axiom? Proof or disprove.

$$[x' = \theta \ \& \ H]\phi \leftrightarrow \forall t \geq 0 \forall 0 \leq s \leq t \ ([x := y(s)]H \rightarrow [x := y(t)]\phi)$$

Exercise 5. All axioms need to be proved to be sound. These lecture notes only did a proper proof for $[\cup]$ and $[:]$. Turn the informal arguments for the other axioms into proper soundness proofs using the semantics of \mathbf{dL} formulas.

Exercise 6. Would the following be a useful replacement for the $[*]$ axiom?

$$[\alpha^*]\phi \leftrightarrow \phi \wedge [\alpha^*]\phi$$

Exercise 7. This lecture identified axioms for all formulas of the form $[\alpha]\phi$ but none for formulas of the form $\langle\alpha\rangle\phi$. Identify and justify these missing axioms. Explain how they relate to the ones given in Note 15.

References

- [CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, 2001.
- [GP14] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. doi:[10.1007/978-3-642-54862-8_19](https://doi.org/10.1007/978-3-642-54862-8_19).
- [Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [MP14] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *RV*, volume 8734 of *LNCS*, pages 199–214. Springer, 2014. doi:[10.1007/978-3-319-11164-3_17](https://doi.org/10.1007/978-3-319-11164-3_17).
- [PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. doi:[10.1007/978-3-540-70545-1_17](https://doi.org/10.1007/978-3-540-70545-1_17).
- [PC09] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. Special issue for selected papers from CAV’08. doi:[10.1007/s10703-009-0079-8](https://doi.org/10.1007/s10703-009-0079-8).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:[1205.4788](https://arxiv.org/abs/1205.4788).

- [Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. [doi:10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [Pla14] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014. [arXiv:1408.1980](https://arxiv.org/abs/1408.1980).
- [SKSC98] Danbing Seto, Bruce Krogh, Lui Sha, and Alongkrit Chutinan. The Simplex architecture for safe online control system upgrades. In *ACC*, volume 6, pages 3504–3508, 1998.
- [Wal98] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1998.

Lecture Notes on Truth & Proof

André Platzer

Carnegie Mellon University
Lecture 6

1 Introduction¹

[Lecture 5 on Dynamical Systems & Dynamic Axioms](#) investigated dynamic axioms for dynamical systems, i.e. axioms in differential dynamic logic ($\text{d}\mathcal{L}$) that characterize operators of the dynamical systems that $\text{d}\mathcal{L}$ describes by hybrid programs in terms of structurally simpler $\text{d}\mathcal{L}$ formulas. All it takes to understand the bigger system, thus, is to apply the axiom and investigate the smaller remainders. That lecture did not quite show all important axioms yet, but it still revealed enough to prove a property of a bouncing ball. Yet, there's more to proofs than just axioms. Proofs also have proof rules for combining fragments of arguments into a bigger proof by proof steps. Proofs, thus, are defined by the glue that holds axioms together into a single cohesive argument justifying its conclusion.

Recall that our proof about the (single-hop) bouncing ball from the previous lecture still suffered from at least two issues. While it was a sound proof and an interesting proof, the way we had come up with it was somewhat undisciplined. We just applied axioms seemingly at random at all kinds of places all over the logical formulas. After we see such a proof, that is not a concern, because we can just follow its justifications and appreciate the simplicity and elegance of the steps it took to justify the conclusion.² But better structuring would certainly help us find proofs more constructively in the first place. The second issue was that the axioms for the dynamics that [Lecture 5](#) showed us did not actually help in proving the propositional logic and arithmetic

¹By both sheer coincidence and by higher reason, the title of this lecture turns out to be closely related to the subtitle of a well-known book on mathematical logic [[And02](#)], which summarizes the philosophy we pursue here in a way that is impossible to improve upon any further: *To truth through proof*.

²Indeed, the proof in [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) was creative in that it used axioms quite carefully in an order that minimizes the notational complexity. But it is not easy to come up with such (nonsystematic) shortcut proofs.

parts. So we were left with informal justifications of the resulting arithmetic at the end, which leaves plenty of room for subtle mistakes in correctness arguments.

The lecture today addresses both issues by imposing more structure on proofs and, as part of that, handle the operators of first-order logic that differential dynamic logic inherits (propositional connectives such as \wedge , \vee , \rightarrow) and quantifiers (\forall , \exists). As part of the structuring, we will make ample and crucial use of the dynamic axioms from [Lecture 5](#). Yet, they will be used in a more structured way than so far. In a way that focuses their use on the top level of the formula and in the direction that actually simplifies the formulas.

These notes are based on [[Pla08](#), [Pla10](#), Chapter 2.5.2], where more information can be found in addition to more information in [[Pla10](#), Appendix A]. Sequent calculus is discussed in more detail also in the handbook of proof theory [[Bus98](#)]. More resources and background material on first-order logic is also listed on the [course web page](#).

While the previous [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) laid down the most fundamental cornerstones of the Foundations of Cyber-Physical Systems and their rigorous reasoning principles, today's lecture revisits these fundamental principles and shapes them into a systematic proof approach. The most important learning goals of this lecture are:

Modeling and Control: This lecture deepens our understanding from the previous lecture on how discrete and continuous systems relate to one another in the presence of evolution domain constraints, a topic that the previous lecture only touched upon briefly.

Computational Thinking: Based on the core rigorous reasoning principles for CPS developed in the [previous lecture](#), today's lecture is devoted to reasoning rigorously *and systematically* about CPS models. Systematic ways of reasoning rigorously about CPS are, of course, critical to getting more complex CPS right. The difference between the axiomatic way of reasoning rigorously about CPS [[Pla12b](#)] as put forth in the previous lecture and the systematic way [[Pla08](#), [Pla10](#)] developed in today's lecture is not a big difference conceptually, but more a difference in pragmatics. That does not make it less important, though, and the occasion to revisit gives us a way of deepening our understanding of systematic CPS analysis principles. Today's lecture also explains ways of developing CPS proofs and logic proofs systematically and is an important ingredient for verifying CPS models of appropriate scale. This lecture also adds a fourth leg to the logical trinity of syntax, semantics, and axiomatics considered in [Lecture 5](#). Today's lecture adds pragmatics, by which we mean the question of how to use axiomatics to justify the syntactic renditions of the semantical concepts of interest. That is, how to best go about conducting a proof to justify truth of a CPS conjecture.

CPS Skills: This lecture is mostly devoted to sharpening our analytic skills for CPS. We will also develop a slightly better intuition for the operational effects involved in CPS in that we understand in which order we should worry about operational effects and whether that has an impact on the overall understanding.

2 Truth and Proof

Truth is defined by the semantics of logical formulas. The semantics gives a mathematical meaning to formulas that, in theory, could be used to establish truth of a logical formula. In practice, this is usually less feasible, for one thing, because quantifiers of differential dynamic logic quantify over real numbers (after all their variables may represent real quantities like velocities and positions). Yet, there are (uncountably) infinitely many of those, so determining the truth value of a universally quantified logical formula directly by working with its semantics is challenging since that'd require instantiating it with infinitely many real numbers, which would keep us busy for a while. The same matter is even more difficult for the hybrid system dynamics involved in modalities of differential dynamic logic formulas, because hybrid systems have so many possible behaviors and are highly nondeterministic. Literally following all possible behaviors to check all reachable states hardly sounds like a way that would ever enable us to stop and conclude the system would be safe. Except, of course, if we happen to be lucky and found a bug during just one execution, because that would be enough to falsify the formula.

Yet, we are still interested in establishing whether a logical formula is true. Or, actually, whether the formula is valid, since truth of a logical formula depends on the state (cf. definition of $\nu \models \phi$ in [Lecture 4 on Safety & Contracts](#)) whereas validity of a logical formula is independent of the state (cf. definition of $\models \phi$), because validity means truth in all states. And validity of formulas is what we ultimately care about, because we want our safety analysis to hold in all permitted initial states of the CPS, not just one particular initial state ν . In that sense, valid logical formulas are the most valuable ones. We should devote all of our efforts to finding out what is valid, because that will allow us to draw conclusions about all states, including the real world state as well.

While exhaustive enumeration and simulation is hardly an option for systems as challenging as CPS, the validity of logical formulas can be established by other means, namely by producing a proof of that formula. Like the formula itself, but unlike its semantics, a proof is a syntactical object that is amenable, e.g., to representation and manipulation in a computer. The finite syntactical argument represented in a proof witnesses the validity of the logical formula that it concludes. Proofs can be produced in a machine. They can be stored to be recalled as witnesses and evidence for the validity of their conclusion. And they can be checked by humans or machines for correctness. They can also be inspected for analytic insights about the reasons for the validity of a formula, which goes beyond the factual statement of validity. A proof justifies the judgment that a logical formula is valid, which, without such a proof as evidence, is no more than an empty claim. And empty claims would hardly be useful foundations for building any cyber-physical systems on.

Truth and proof should be related intimately, however, because we would only want to accept proofs that actually imply truth, i.e. proofs that imply their consequences to be valid if their premises are. That is, proof systems should be *sound* in order to allow us to draw reliable conclusions from the existence of a proof. And, in fact, this course will exercise great care to identify sound reasoning principles. The converse and equally

intriguing question is that of completeness, i.e. whether all true formulas (again in the sense of valid) can be proved, which turns out to be much more subtle [Pla12a] and won't concern us until much later in this course.

3 Sequents

The proof built from axioms in [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) to justify a safety property of a bouncing ball was creative and insightful, but also somewhat spontaneous and disorganized. In fact, it has not even quite become particularly obvious what exactly a proof was, except that it is somehow supposed to glue axioms together into a single cohesive argument.³ But that is not a definition of a proof.

In order to have a chance to conduct more complex proofs, we need a way of structuring the proofs and keeping track of all questions that come up while working on a proof. But despite all the lamenting about the proof from [Lecture 5](#), it has, secretly, been much more systematic than we were aware of. Even if it went in a non-systematic order as far as the application order of the proof rules is concerned, we still structured the proof quite well (unlike the ad-hoc arguments in [Lecture 4 on Safety & Contracts](#)). So part of what this lecture needs to establish is to turn this coincidence into an intentional principle. Rather than just coincidentally structuring the proof well, we want to structure all proofs well and make them all systematic by design.

Throughout this course, we will use *sequents*, which give us a structuring mechanism for conjectures and proofs. Sequent calculus was originally developed by Gerhard Gentzen [[Gen35a](#), [Gen35b](#)] for studying properties of natural deduction calculi, but sequent calculi have been used very successfully for numerous other purposes since.

In a nutshell, sequents are essentially a standard form for logical formulas that is convenient for proving purposes, because, intuitively, it neatly aligns all available assumptions on the left and gathers what needs to be shown on the right.

Definition 1 (Sequent). A *sequent* is of the form $\Gamma \vdash \Delta$, where the *antecedent* Γ and *succedent* Δ are finite sets of formulas. The semantics of $\Gamma \vdash \Delta$ is that of the formula $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$.

The antecedent Γ can be thought of as the formulas we assume to be true, whereas the succedent Δ can be understood as formulas for which we want to show that at least one of them is true assuming all formulas of Γ are true. So for proving a sequent $\Gamma \vdash \Delta$, we assume all Γ and want to show that one of the Δ is true. For some simple sequents like $\Gamma, \phi \vdash \phi, \Delta$, we directly know that they are valid, because we can certainly show ϕ if we assume ϕ (in fact, we will use this as a way of finishing a proof). For other sequents, it is more difficult to see whether they are valid (true under all circumstances) and it is the purpose of a proof calculus to provide a means to find out.

³It would have been very easy to define, though, by inductively defining formulas to be provable if they are either instances of axioms or follow from provable formulas using modus ponens [[Pla12b](#)].

The basic idea in sequent calculus is to successively transform all formulas such that Γ forms a list of all assumptions and Δ the set of formulas that we would like to conclude from Γ (or, to be precise, the set Δ whose disjunction we would like to conclude from the conjunction of all formulas in Γ). So one way of understanding sequent calculus is to interpret $\Gamma \vdash \Delta$ as the task of proving one of the formulas in the succedent Δ from all of the formulas in the antecedent Γ . But since \mathbf{dL} is a classical logic, not an intuitionistic logic, we need to keep in mind that it is actually enough for proving a sequent $\Gamma \vdash \Delta$ to just prove the disjunction of all formulas in Δ from the conjunction of all formulas in Γ . For the proof rules of real arithmetic, we will later make use of this fact by considering sequent $\Gamma \vdash \Delta$ as an abbreviation for the formula $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$, because both have the same semantics in \mathbf{dL} .

Empty conjunctions $\bigwedge_{\phi \in \emptyset} \phi$ are equivalent to *true*. Empty disjunctions $\bigvee_{\phi \in \emptyset} \phi$ are equivalent to *false*.⁴ Hence, the sequent $\vdash A$ means the same as the formula A . The empty sequent \vdash means the same as the formula *false*.

Note 2 (Nonempty trouble with empty sequents). *If you ever reduce a conjecture about your CPS to proving the empty sequent \vdash , then you are in trouble, because it is rather hard to prove false, since false isn't ever true. In that case, either you have taken a wrong turn in your proof, e.g., by discarding an assumption that was actually required for the conjecture to be true, or your CPS might take the wrong turn, because its controller can make a move that is actually unsafe.*

4 Structural Proof Rules

Before discussing any particular proof rules of \mathbf{dL} , let us first understand some common properties of most sequent calculi. The antecedent and succedent of a sequent are considered as sets. So the order of formulas is irrelevant, and we implicitly adopt what is called the *exchange rule* and do not distinguish between the following two sequents

$$\Gamma, A, B \vdash \Delta \quad \text{and} \quad \Gamma, B, A \vdash \Delta$$

ultimately since $A \wedge B$ and $B \wedge A$ are equivalent anyhow, nor do we distinguish between

$$\Gamma \vdash C, D, \Delta \quad \text{and} \quad \Gamma \vdash D, C, \Delta$$

ultimately since $C \vee D$ and $D \vee C$ are equivalent. Antecedent and succedent are considered as sets, not multisets, so we implicitly adopt what is called the *contraction rule* and do not distinguish between the following two sequents

$$\Gamma, A, A \vdash \Delta \quad \text{and} \quad \Gamma, A \vdash \Delta$$

⁴Note that *true* is the neutral element for the operation \wedge and *false* the neutral element for the operation \vee . That is $A \wedge \text{true}$ is equivalent to A for any A and $A \vee \text{false}$ is equivalent to A . So *true* plays the same role that 1 plays for multiplication. And *false* plays the role that 0 plays for addition. Another aspect of sequents $\Gamma \vdash \Delta$ that is worth mentioning is that other notations such as $\Gamma \Longrightarrow \Delta$ or $\Gamma \longrightarrow \Delta$ are also sometimes used in other contexts.

because $A \wedge A$ and A are equivalent, nor do we distinguish between

$$\Gamma \vdash C, C, \Delta \quad \text{and} \quad \Gamma \vdash C, \Delta$$

because $C \vee C$ and C are equivalent.

The only structural rule of sequent calculus that we will find reason to use explicitly in practice is the *weakening* proof rule (alias *hide* rule) that can be used to remove or hide formulas from the antecedent (Wl) or succedent (Wr), respectively:

$$(Wr) \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta}$$

$$(Wl) \frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta}$$

Weakening rules are sound, since it is fine in (structural) logics to prove a sequent with more formulas in the antecedent or succedent by a proof that uses only some of those formulas. This is different in substructural logics such as linear logic. Proof rule Wl proves the conclusion $\Gamma, \phi \vdash \Delta$ from the premise $\Gamma \vdash \Delta$, which dropped the assumption ϕ . Surely, if premise $\Gamma \vdash \Delta$ is valid, then conclusion $\Gamma, \phi \vdash \Delta$ is valid as well, because it even has one more (unused) assumption available (ϕ). Proof rule Wr proves the conclusion $\Gamma \vdash \phi, \Delta$ from the premise $\Gamma \vdash \Delta$, which is fine because $\Gamma \vdash \Delta$ just has one less (disjunctive) option in its succedent. For this, recall that succedents have a disjunctive meaning.

At first sight, weakening may sound like a stupid thing to do in any proof, because rule Wl discards available assumptions (ϕ) and rule Wr discards available options (ϕ) for proving the statement. This seems to make it harder to prove the statement after using a weakening rule. But weakening is actually useful for managing computational and conceptual proof complexity by enabling us to throw away irrelevant assumptions. These assumptions may have been crucial for another part of the proof, but have just become irrelevant for the particular sequent at hand, which can, thus, be simplified to $\Gamma \vdash \Delta$. Weakening, thus, streamlines proofs, which can, e.g., also help speed up arithmetic immensely (Sect. 11).

5 Propositional Proof Rules

The first logical operators encountered during proofs are usually propositional logical connectives, because many \mathbf{dL} formulas use forms such as $A \rightarrow [\alpha]B$ to express that all behavior of HP α leads to safe states satisfying B when starting the system in initial states satisfying A . For propositional logic, \mathbf{dL} uses the standard propositional rules with the cut rule, which are listed in Fig. 1. Each of these propositional rules decompose the propositional structure of formulas and neatly divides everything up into assumptions (which will ultimately be moved to the antecedent) and what needs to be shown (which will be moved to the succedent). The rules will be developed one at a time in the order that is most conducive to their intuitive understanding.

Proof rule $\wedge I$ is for handling conjunctions ($\phi \wedge \psi$) in the antecedent. It expresses that if a conjunction $\phi \wedge \psi$ is among the list of available assumptions in the antecedent, then

$$\begin{array}{lll}
(\neg r) \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \neg \phi, \Delta} & (\vee r) \frac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \vee \psi, \Delta} & (\wedge r) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \\
(\neg l) \frac{\Gamma \vdash \phi, \Delta}{\Gamma, \neg \phi \vdash \Delta} & (\vee l) \frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \vee \psi \vdash \Delta} & (\wedge l) \frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta} \\
(\rightarrow r) \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta} & (ax) \frac{}{\Gamma, \phi \vdash \phi, \Delta} & \\
(\rightarrow l) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \rightarrow \psi \vdash \Delta} & (cut) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta} &
\end{array}$$

Figure 1: Propositional proof rules of sequent calculus

we might just as well assume both conjuncts (ϕ and ψ , respectively) separately. If we set out to prove a sequent of the form in the conclusion $(\Gamma, \phi \wedge \psi \vdash \Delta)$, then we can justify this sequent by instead proving the sequent in the premise $(\Gamma, \phi, \psi \vdash \Delta)$, where the only difference is that the two assumptions ϕ and ψ are now assumed separately in the premise rather than jointly as a single conjunction as in the conclusion. If we just keep on using proof rule $\wedge l$ often enough, then all conjunctions in the antecedent will ultimately have been split into their pieces. Recall that the order of formulas in a sequent $\Gamma \vdash \Delta$ is irrelevant because Γ and Δ are sets, so we can always pretend that the formula that we want to apply the $\wedge l$ rule to is last in the antecedent. So $\wedge l$ takes care of all conjunctions that appear as top-level operators in antecedents. But there are other logical operators to worry about as well.

Proof rule $\vee r$ is similar to $\wedge l$ but for handling disjunctions in the succedent. If we set out to prove the sequent $\Gamma \vdash \phi \vee \psi, \Delta$ in the conclusion with a disjunction $\phi \vee \psi$ in the succedent, then we might as well split the disjunction into its two disjuncts and prove the premise $\Gamma \vdash \phi, \psi, \Delta$ instead, since the succedent has a disjunctive meaning anyhow.

Proof rule $\wedge r$ works differently, because if we are trying to prove a sequent $\Gamma \vdash \phi \wedge \psi, \Delta$ with a conjunction $\phi \wedge \psi$ in its succedent, it would not be enough at all to just prove $\Gamma \vdash \phi, \psi, \Delta$, because, as in rule $\vee r$, this would only enable us to conclude $\Gamma \vdash \phi \vee \psi, \Delta$. Instead, proving a conjunction in the succedent as in the conclusion of $\wedge r$ requires proving both conjuncts, so a proof of $\Gamma \vdash \phi, \Delta$ and a proof of $\Gamma \vdash \psi, \Delta$. This is why rule $\wedge r$ splits the proof into two branches, one for proving $\Gamma \vdash \phi, \Delta$ and one for proving $\Gamma \vdash \psi, \Delta$. Indeed, if both premises of rule $\wedge r$ are valid then so is its conclusion. To see this, it is easier to first consider the case where Δ is empty and then argue by cases, once for the case where the disjunction corresponding to Δ is true and once where it is false.

Similarly, proof rule $\vee l$ handles a disjunction in the antecedent. When the assumptions listed in the antecedent of a sequent contain a disjunction $\phi \vee \psi$, then there is no way of knowing which of the two can be assumed only that at least one of them can be assumed to be true. Rule $\vee l$, thus, splits the proof into cases. The left premise considers the case where the assumption $\phi \vee \psi$ held because ϕ was true. The right premise considers the case where assumption $\phi \vee \psi$ held because ψ was true. If both premises are valid

(because we can find a proof for them), then, either way, the conclusion $\Gamma, \phi \vee \psi \vdash \Delta$ will be valid no matter which of the two cases applies.

Proof rule $\rightarrow r$ handles implications in the succedent by using the implicational meaning of sequents. The way to understand it is to recall how we would go about proving an implication. In order to prove an implication $\phi \rightarrow \psi$, we would assume the left-hand side ϕ (which $\rightarrow r$ pushes into the assumptions listed in the antecedent) and try to prove its right-hand side ψ (which $\rightarrow r$ thus leaves in the succedent).

Proof rule $\rightarrow l$ is more involved. And one way to understand it is to recall that classical logic obeys the equivalence $(\phi \rightarrow \psi) \equiv (\neg\phi \vee \psi)$. A direct argument explaining $\rightarrow l$ uses that when assuming an implication $\phi \rightarrow \psi$, we can only assume its right-hand side ψ after we have shown its respective assumption ϕ on its left-hand side.

Proof rule $\neg r$ proves a negation $\neg\phi$ by, instead, assuming ϕ . Again, the easiest way of understanding this is for an empty Δ in which case rule $\neg r$ expresses that the way of proving a negation $\neg\phi$ in the succedent of the conclusion is to instead assume ϕ in the antecedent in the premise and then proving a contradiction in the form of the empty succedent, which is *false*. Alternatively, rule $\neg r$ can be understood using the semantics of sequents, since a conjunct ϕ on the left-hand side of an implication is semantically equivalent to a disjunct $\neg\phi$ on the right-hand side.

Proof rule $\neg l$ handles a negation $\neg\phi$ among the assumptions in the antecedent of the conclusion by, instead, pushing ϕ into the succedent of the premise. Indeed, for the case of empty Δ , if ϕ were shown to hold assuming Γ , then Γ and $\neg\phi$ imply a contradiction in the form of the empty sequent, which is *false*. Again, a semantic argument using the semantics of sequents also justifies $\neg l$ directly.

All these propositional rules make progress by splitting operators. And that will ultimately lead to atomic formulas, i.e. those formulas without any logical operators. But there is no way to ever properly stop the proof yet. That is what the axiom rule ax is meant for (not to be confused with the axioms from [Lecture 5](#)). The axiom rule ax closes a goal (there are no further subgoals, which we sometimes mark by a $*$ explicitly), because assumption ϕ in the antecedent trivially entails ϕ in the succedent (the sequent $\Gamma, \phi \vdash \phi, \Delta$ is a simple syntactic tautology). If, in our proving activities, we ever find a sequent of the form $\Gamma, \phi \vdash \phi, \Delta$, for any formula ϕ , we can immediately use the axiom rule ax to close this part of the prove.

Rule cut is Gentzen's *cut* rule [[Gen35a](#), [Gen35b](#)] that can be used for case distinctions: The right subgoal assumes any additional formula ϕ in the antecedent that the left subgoal shows in the succedent. Dually: regardless of whether ϕ is actually true or false, both cases are covered by proof branches. Alternatively, and maybe more intuitively, the cut rule is fundamentally a lemma rule. The left premise proves an auxiliary lemma ϕ in its succedent, which the right premise then assumes in its antecedent (again consider the case of empty Δ first). We only use cuts in an orderly fashion to derive simple rule dualities and to simplify meta-proofs. In practical applications, cuts are not needed in theory. But in practice, complex practical applications make use of cuts for efficiency reasons. Cuts can be used, for example, to simplify arithmetic, or to first prove lemmas and then make ample use of them, in a number of places in the remaining proof.

Even though we write sequent rules as if the principal formula (like $\phi \wedge \psi$ in $\wedge r, \wedge l$)

were at the end of the antecedent or at the beginning of the succedent, respectively, the sequent proof rules can be applied to other formulas in the antecedent or succedent, respectively, because we consider their order to be irrelevant (aset).

6 Proofs

The $\text{d}\mathcal{L}$ calculus has further proof rules beyond the structural and propositional rules. But before investigating those additional rules, let us first understand what exactly a proof is, what it means to prove a logical formula, and how we know whether a proof rule is sound. The same notions of proof, provability and soundness work for propositional logic as for differential dynamic logic, except that the latter has more proof rules.⁵ The soundness notion that will be sufficient for our purposes in this course is the following.

Definition 2 (Global Soundness). A sequent calculus proof rule of the form

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \dots \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$$

is *sound* iff the validity of all premises implies the validity of the conclusion, i.e.

$$\models (\Gamma_1 \vdash \Delta_1) \dots \text{ and } \models (\Gamma_n \vdash \Delta_n) \text{ implies } \models (\Gamma \vdash \Delta)$$

Recall from Def. 1 that the meaning of a sequent $\Gamma \vdash \Delta$ is $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$, so that $\models (\Gamma \vdash \Delta)$ stands for $\models \left(\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi \right)$.

A formula ϕ is provable or derivable (in the $\text{d}\mathcal{L}$ calculus) if we can find a $\text{d}\mathcal{L}$ proof for it that starts with axioms (rule [ax](#)) at all its leaves and ends with a sequent $\vdash \phi$ at the bottom and that has only used $\text{d}\mathcal{L}$ proof rules in between to go from their premises to their conclusion. The shape of a $\text{d}\mathcal{L}$ proof, thus, is a tree with the axioms at the top leaves and the formula that the proof proves at the bottom root. While constructing proofs, however, we would start with the desired goal $\vdash \phi$ at the bottom that we want as the eventual conclusion of the proof and we work our way backwards to the subgoals until they can be proven to be valid as axioms ([ax](#)). Once all subgoals have been proven to be valid axioms, they entail their respective conclusion, which, recursively, entail the original goal $\vdash \phi$. This property of preserving truth or preserving entailment is called soundness. Thus, while constructing proofs, we work bottom-up from the goal and apply all proof rules from the desired conclusion to the required premises. When we have found a proof, we justify formulas conversely from the axioms top-down to the original goal, because validity transfers from the premises to the conclusion when using sound proof rules.

⁵ There is one subtlety with the interaction of \forall and \exists for automation purposes and how that leads to a more general notion of soundness [[Pla08](#)]. But this generalization is not needed for the purposes of this course.

$$\text{construct proofs upwards} \quad \uparrow \quad (\wedge r) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \quad \downarrow \quad \text{validity transfers downwards}$$

We write $\vdash_{d\mathcal{L}} \phi$ iff $d\mathcal{L}$ formula ϕ can be *proved* with $d\mathcal{L}$ rules from $d\mathcal{L}$ axioms. That is, a $d\mathcal{L}$ formula is inductively defined to be *provable* in the $d\mathcal{L}$ sequent calculus if it is the conclusion (below the rule bar) of an instance of one of the $d\mathcal{L}$ sequent proof rules, whose premises (above the rule bar) are all provable. A formula ψ is *provable* from a set Φ of formulas, denoted by $\Phi \vdash_{d\mathcal{L}} \psi$, iff there is a finite subset $\Phi_0 \subseteq \Phi$ of formulas for which the sequent $\Phi_0 \vdash \psi$ is provable.

Example 3. A very simple (in fact propositional) proof of the formula

$$v^2 \leq 10 \wedge b > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq 10) \quad (1)$$

is shown in Fig. 2. The proof starts with the desired proof goal as a sequent at the bottom:

$$\vdash v^2 \leq 10 \wedge b > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq 10).$$

and proceeds by applying suitable sequent proof rules upwards.

$$\begin{array}{c} \begin{array}{c} * \\ \hline \text{ax} \frac{}{v^2 \leq 10, b > 0 \vdash b > 0} \\ \hline \wedge l \frac{}{v^2 \leq 10 \wedge b > 0 \vdash b > 0} \end{array} \quad \begin{array}{c} * \\ \hline \text{ax} \frac{}{v^2 \leq 10, b > 0 \vdash \neg(v \geq 0), v^2 \leq 10} \\ \hline \wedge l \frac{}{v^2 \leq 10 \wedge b > 0 \vdash \neg(v \geq 0), v^2 \leq 10} \\ \hline \vee r \frac{}{v^2 \leq 10 \wedge b > 0 \vdash \neg(v \geq 0) \vee v^2 \leq 10} \end{array} \\ \hline \wedge r \frac{}{v^2 \leq 10 \wedge b > 0 \vdash b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq 10)} \\ \hline \rightarrow r \frac{}{\vdash v^2 \leq 10 \wedge b > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq 10)} \end{array}$$

Figure 2: A simple propositional example proof in sequent calculus

The first (i.e., bottom most) proof step applies proof rule $\rightarrow r$ to turn the implication (\rightarrow) to the sequent level by moving its left-hand side into the assumptions tracked in the antecedent. The next proof step applies rule $\wedge r$ to split the proof into the left branch for showing that conjunct $b > 0$ follows from the assumptions in the antecedent and into the right branch for showing that conjunct $\neg(v \geq 0) \vee v^2 \leq 10$ follows from the antecedent also. On the left branch, the proof closes with an axiom ax after splitting the conjunction \wedge in the antecedent into its conjuncts with rule $\wedge l$. We mark closed proof goals with $*$, to indicate that we did not just stopped writing but that a subgoal is actually proved successfully. It makes sense that the left branch closes by the axiom rule ax , because its assumption $b > 0$ in the antecedent trivially implies the formula $b > 0$ in the succedent, because both formulas are identical. The right branch closes with an axiom ax after splitting the disjunction (\vee) in the succedent with rule $\vee r$ and then splitting the conjunction (\wedge) in the antecedent with rule $\wedge l$. On the right branch,

the first assumption formula $v^2 \leq 10$ in the antecedent trivially implies the last formula in the succedent $v^2 \leq 10$, because both are identical, so the axiom rule [ax](#) applies. Now that all branches of the proof have closed (with [ax](#) and marked by *), we know that all leaves at the top are valid, and, hence, since the premises are valid, each application of a proof rule ensures that their respective conclusions are valid also, by soundness. By recursively following this proof from the leaves at the top to the original root at the bottom, we conclude that the original goal at the bottom is valid and formula (1) is, indeed, true under all circumstances (valid). And that is what we set out to prove, that formula (1) is valid, which the proof in Fig. 2 justifies.

While this proof does not prove any particularly exciting formula, it still shows how a proof can be built systematically in the dL calculus and gives an intuition as to how validity is inherited from the premises to the conclusions. Note that the proof has been entirely systematic. All we did to come up with it was successively inspect the top-level operator in one of the logical formulas in the sequent and apply its corresponding propositional proof rule to find the resulting subgoals. All the while we were doing this, we carefully watched to see if the same formula shows up in the antecedent and succedent, for then the axiom rule [ax](#) closes that subgoal. There would be no point in proceeding with any other proof rule if the [ax](#) rule closes a subgoal.

Most interesting formulas will not be provable with the sequent proof rules we have seen so far, because those were only propositional and structural rules. Next, we, thus, set out to find sequent proof rules for the other operators of dL .

First, though, notice that the sequent proof rules are sound. We consider only one of the proof rules to show how soundness works. Soundness is crucial, however, so you are invited to prove soundness for the other rules (Exercise 3).

Proof. The proof rule [∧r](#) is sound. For this, consider any instance for which both premises $\Gamma \vdash \phi, \Delta$ and $\Gamma \vdash \psi, \Delta$ are valid and show that the conclusion $\Gamma \vdash \phi \wedge \psi, \Delta$ is valid. To show the latter, consider any state ν . If there is a formula $F \in \Gamma$ in the antecedent that is not true in ν (i.e. $\nu \not\models F$) there is nothing to show, because $\nu \models (\Gamma \vdash \phi \wedge \psi, \Delta)$ then holds trivially, because not all assumptions in Γ are satisfied in ν . Likewise, if there is a formula $G \in \Delta$ in the succedent that is true in ν (i.e. $\nu \models G$) there is nothing to show, because $\nu \models (\Gamma \vdash \phi \wedge \psi, \Delta)$ then holds trivially, because one of the formulas in the succedent is already satisfied in ν . Hence, the only interesting case to consider is the case where all formulas in $F \in \Gamma$ are true in ν and all formulas $G \in \Delta$ are false. In that case, since both premises were assumed to be valid, and Γ is true in ν but Δ false in ν , the left premise implies that $\nu \models \phi$ and the right premise implies that $\nu \models \psi$. Consequently, $\nu \models \phi \wedge \psi$ by the semantics of \wedge . Thus, $\nu \models (\Gamma \vdash \phi \wedge \psi, \Delta)$. As the state ν was arbitrary, this implies $\models (\Gamma \vdash \phi \wedge \psi, \Delta)$, i.e. the conclusion of the considered instance of [∧r](#) is valid. \square

7 Dynamic Proof Rules

When making the proof for the bouncing ball from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) systematic by turning it into a sequent calculus proof, the first propositional step succeeds to turn \rightarrow into a sequent, but the rest of the proof involves modalities referring to the behavior of hybrid programs. The next set of sequent rules for $\text{d}\mathcal{L}$ handles the hybrid programs in the modalities. [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) has already shown and (at least informally) justified axioms for dynamical systems that correspond to each of the operators of hybrid programs in $[\cdot]$ modalities of differential dynamic logic [\[Pla12b\]](#). These were equivalence axioms which represent schemata of valid formulas such as

$$([\cup]) \quad [\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$$

How can such valid equivalences be used in the context of a sequent calculus? There is more than one productive way to do that.

The $\text{d}\mathcal{L}$ axioms such as axiom [\[U\]](#) are primarily meant to be used for replacing the left-hand side $[\alpha \cup \beta]\phi$ by the structurally simpler right-hand side $[\alpha]\phi \wedge [\beta]\phi$, because that direction of use assigns meaning to $[\alpha \cup \beta]\phi$ in logically simpler terms, i.e. as a structurally simpler logical formula. Thus, whenever there is an occurrence of a formula of the form $[\alpha \cup \beta]\phi$, the equivalence axiom [\[U\]](#) ought to be used from left to right to get rid of $[\alpha \cup \beta]\phi$ and replace it by the structurally simpler right-hand side of the axiom. The following two sequent proof rules allow replacements in that direction for formulas in the antecedent ([\[U\]l](#)) and succedent ([\[U\]r](#)), respectively.

$$([\cup]r) \quad \frac{\Gamma \vdash [\alpha]\phi \wedge [\beta]\phi, \Delta}{\Gamma \vdash [\alpha \cup \beta]\phi, \Delta}$$

$$([\cup]l) \quad \frac{\Gamma, [\alpha]\phi \wedge [\beta]\phi \vdash \Delta}{\Gamma, [\alpha \cup \beta]\phi \vdash \Delta}$$

The sequent proof rules [\[U\]r](#), [\[U\]l](#) are more systematic in that they orient the use of the axiom [\[U\]](#) in the direction that makes formulas structurally simpler. Without such direction, proofs could apply axiom [\[U\]](#) from left to right and then from right to left and from left to right again forever without making any actual progress. That does not happen with [\[U\]r](#), [\[U\]l](#), because they cannot simply go back.⁶ Furthermore, the sequent rules [\[U\]r](#), [\[U\]l](#) focus the application of axiom [\[U\]](#) to the top level of sequents. That is, [\[U\]r](#), [\[U\]l](#) can only be used for formulas of the succedent or antecedent, respectively, that are of the form $[\alpha \cup \beta]\phi$, not to any subformulas within those formulas that happen to be of this form. Abiding both of those restrictions imposes more structure on the proof, compared to the proof we produced in [Lecture 5](#). In particular, there is exactly one sequent proof rule that can be applied to a formula of the form $[\alpha \cup \beta]\phi$ in a sequent.⁷

⁶Albeit, going back is still possible indirectly when using a reasonably creative *cut*. But that requires an intentional extra effort to do so, hence, does not happen accidentally during proof search.

⁷With the exception of differential equations and, to a lesser extent, loops, the whole $\text{d}\mathcal{L}$ sequent calculus singles out exactly one proof rule to apply, just depending on the top-level logical operators in the formula. Differential equations are slightly more complicated, because there will eventually be more options for proving differential equations.

Reconsidering the contract-type rules from [Lecture 4 on Safety & Contracts](#), we could have turned axiom $[\cup]$ from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) into the following two sequent proof rules instead of into the two sequent rules $[\cup]r, [\cup]l$:

$$(R14) \frac{\Gamma \vdash [\alpha]\phi, \Delta \quad \Gamma \vdash [\beta]\phi, \Delta}{\Gamma \vdash [\alpha \cup \beta]\phi, \Delta}$$

$$(R15) \frac{\Gamma, [\alpha]\phi, [\beta]\phi \vdash \Delta}{\Gamma, [\alpha \cup \beta]\phi \vdash \Delta}$$

These rules [R14, R15](#) already split into separate subgoals ([R14](#)) or separate formulas ([R15](#)), respectively. It would be fine to use sequent rules [R14, R15](#) instead of $[\cup]r, [\cup]l$, and, in fact, earlier versions of KeYmaera did. The disadvantage of rules [R14, R15](#) compared to $[\cup]r, [\cup]l$ is that rules [R14, R15](#) have a less obvious relation to axiom $[\cup]$ and that they are asymmetric (they both look surprisingly different). This nuisance is overcome in the rules $[\cup]r, [\cup]l$, from which rules [R14, R15](#) follow immediately with just one more application of rules $\wedge r$ or $\wedge l$, respectively. Thus, $[\cup]r, [\cup]l$ are more elementary and more atomic in that they isolate the proof-theoretical meaning of $[\alpha \cup \beta]\phi$, as opposed to already incorporating parts of the meaning of \wedge as well, which, after all, is what the propositional rules $\wedge r, \wedge l$ are supposed to capture. Consequently, while the result of alternative rules [R14, R15](#) is what will ultimately happen after applying sequent rules $[\cup]r, [\cup]l$ regardless, we decide against the alternatives [R14, R15](#), because they blur the essence of the meaning of $[\alpha \cup \beta]\phi$ unnecessarily and make the symmetry of the antecedent and succedent use more apparent.

The other $d\mathcal{L}$ axioms from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) translate into sequent calculus proof rules in exactly the same way. The equivalences that the $d\mathcal{L}$ axioms identify are lifted to the sequent level by introducing a pair of sequent rules, one for the antecedent and one for the succedent, and orienting the equivalence such that formulas always get structurally simpler when applying the sequent rules. Thus, all dynamic modality rules of the $d\mathcal{L}$ sequent calculus transform a hybrid program into structurally simpler logical formulas by symbolic decomposition or symbolic execution.

In order to simplify notation, we adopt a convention that exhibits the symmetry of antecedent and succedent rules. Instead of rules $[\cup]r, [\cup]l$, [Fig. 3](#) shows a single *symmetric rule* $[\cup]$ that does not mention the sequent sign \vdash :

$$\frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi}$$

This is abbreviated notation to say that the same rule from a conclusion with a formula $[\alpha \cup \beta]$ in either antecedent or succedent can be proved from a premise with formula $[\alpha]\phi \wedge [\beta]\phi$ in the antecedent or succedent, respectively. That is, we consider the symmetric rule $[\cup]$ as an abbreviation for the two rules $[\cup]r, [\cup]l$. [Fig. 3](#) lists a single symmetric rule $[\cup]$ but we pretend it had both rules $[\cup]r, [\cup]l$. The same applies to the other symmetric rules in [Fig. 3](#), which each have a version of the rule for the antecedent and a version of the rule for the succedent. The antecedent version of $[\cdot]$ is called $[\cdot]l$, its succedent version is called $[\cdot]r$. The antecedent version of $[\cdot]$ is called $[\cdot]l$, its succedent

version is called $\langle \cdot \rangle r$ and so on (a full list is in Fig. 8 for reference). Furthermore, Fig. 3 lists rules both for formulas of the form $[\alpha]\phi$ and for formulas of the form $\langle \alpha \rangle \phi$.

$$\begin{array}{lll}
(\langle \cdot \rangle) \frac{\langle \alpha \rangle \langle \beta \rangle \phi}{\langle \alpha; \beta \rangle \phi} & (\langle *n \rangle) \frac{\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi}{\langle \alpha^* \rangle \phi} & (\langle := \rangle) \frac{\phi_x^\theta}{\langle x := \theta \rangle \phi} \\
([\cdot]) \frac{[\alpha][\beta]\phi}{[\alpha; \beta]\phi} & ([*n]) \frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi} & ([:=]) \frac{\phi_x^\theta}{[x := \theta]\phi} \\
(\langle \cup \rangle) \frac{\langle \alpha \rangle \phi \vee \langle \beta \rangle \phi}{\langle \alpha \cup \beta \rangle \phi} & (\langle ? \rangle) \frac{H \wedge \psi}{\langle ?H \rangle \psi} & (\langle ' \rangle) \frac{\exists t \geq 0 ((\forall 0 \leq \tilde{t} \leq t \langle x := y(\tilde{t}) \rangle H) \wedge \langle x := y(t) \rangle \phi)}{\langle x' = \theta \& H \rangle \phi} \quad 1 \\
([\cup]) \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} & ([?]) \frac{H \rightarrow \psi}{[?H]\psi} & ([']) \frac{\forall t \geq 0 ((\forall 0 \leq \tilde{t} \leq t [x := y(\tilde{t})] H) \rightarrow [x := y(t)] \phi)}{[x' = \theta \& H]\phi} \quad 1
\end{array}$$

¹ t and \tilde{t} are fresh logical variables and $\langle x := y(t) \rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value.

Figure 3: Dynamic proof rules of \mathbf{dL} sequent calculus

Nondeterministic choices split into their alternatives $(\langle \cup \rangle, [\cup])$. For rule $[\cup]$: If all α transitions lead to states satisfying ϕ (i.e., $[\alpha]\phi$ holds) and all β transitions lead to states satisfying ϕ (i.e., $[\beta]\phi$ holds), then, all transitions of program $\alpha \cup \beta$ that choose between following α and following β also lead to states satisfying ϕ (i.e., $[\alpha \cup \beta]\phi$ holds). Dually for rule $\langle \cup \rangle$, if there is an α transition to a ϕ state ($\langle \alpha \rangle \phi$) or a β -transition to a ϕ state ($\langle \beta \rangle \phi$), then, in either case, there is a transition of $\alpha \cup \beta$ to ϕ ($\langle \alpha \cup \beta \rangle \phi$ holds), because $\alpha \cup \beta$ can choose which of those transitions to follow. A general principle behind the \mathbf{dL} proof rules that is most noticeable in $\langle \cup \rangle, [\cup]$ is that these proof rules symbolically decompose the reasoning into two separate parts and analyse the fragments α and β separately, which is good for scalability. For these symbolic structural decompositions, it is very helpful that \mathbf{dL} is a full logic that is closed under all logical operators, including disjunction and conjunction, for then the premises in $[\cup], \langle \cup \rangle$ are \mathbf{dL} formulas again (unlike in Hoare logic [Hoa69]).

Sequential compositions are proven using nested modalities $(\langle \cdot \rangle, [\cdot])$. For rule $[\cdot]$: If after all α -transitions, all β -transitions lead to states satisfying ϕ (i.e., $[\alpha][\beta]\phi$ holds), then also all transitions of the sequential composition $\alpha; \beta$ lead to states satisfying ϕ (i.e., $[\alpha; \beta]\phi$ holds). The dual rule $\langle \cdot \rangle$ uses the fact that if there is an α -transition, after which there is a β -transition leading to ϕ (i.e., $\langle \alpha \rangle \langle \beta \rangle \phi$), then there is a transition of $\alpha; \beta$ leading to ϕ (that is, $\langle \alpha; \beta \rangle \phi$), because the transitions of $\alpha; \beta$ are just those that first do any α -transition, followed by any β -transition.

Rules $\langle *n \rangle, [*n]$ are the usual iteration rules, which partially unwind loops. Rule $\langle *n \rangle$ uses the fact that ϕ holds after repeating α (i.e., $\langle \alpha^* \rangle \phi$), if ϕ holds at the beginning (for ϕ holds after zero repetitions then), or if, after one execution of α , ϕ holds after any number of repetitions of α , including zero repetitions (i.e., $\langle \alpha \rangle \langle \alpha^* \rangle \phi$). So rule $\langle *n \rangle$ expresses that for $\langle \alpha^* \rangle \phi$ to hold, ϕ must hold either immediately or after one or more repetitions of α . Rule $[*n]$ is the dual rule expressing that ϕ must hold after all of those

combinations for $[\alpha^*]\phi$ to hold.

Tests are proven by showing (with a conjunction in rule $\langle ? \rangle$) or assuming (with an implication in rule $[?]$) that the test succeeds, because test $?H$ can only make a transition when condition H actually holds true. Thus, for dL formula $\langle ?H \rangle \phi$, rule $\langle ? \rangle$ is used to prove that H holds true (otherwise there is no transition and thus the reachability property is false) and that ϕ holds after the resulting no-op. Rule $[?]$ for dL formula $[?H]\phi$, in contrast, assumes that H holds true (otherwise there is no transition and thus nothing to show) and shows that ϕ holds after the resulting no-op.

Given first-order definable flows for their differential equations, proof rules $\langle \rangle, []$ handle continuous evolutions. These flows are combined in the discrete jump set $x := y(t)$. Given a solution $x := y(t)$ for the differential equation system with symbolic initial values x_1, \dots, x_n , continuous evolution along differential equations can be replaced by a discrete jump $\langle x := y(t) \rangle$ with an additional quantifier for the evolution time t . The effect of the constraint on H is to restrict the continuous evolution such that its solution $x := y(\tilde{t})$ remains in the evolution domain H at all intermediate times $\tilde{t} \leq t$. This constraint simplifies to *true* if the evolution domain restriction H is *true*, which makes sense, because there are no special constraints on the evolution (other than the differential equations) if the evolution domain region is described by *true*, hence the full space \mathbb{R}^n . A notable special case of rules $[]$ and $\langle \rangle$ is when the evolution domain H is *true*:

$$\frac{\forall t \geq 0 [x := y(t)]\phi}{[x' = \theta]\phi} \qquad \frac{\exists t \geq 0 \langle x := y(t) \rangle \phi}{\langle x' = \theta \rangle \phi} \quad (2)$$

Finally note that rules $[]$ and $\langle \rangle$ apply in similar ways to the case of differential equation systems [Pla08, Pla12b] (Exercise 5):

$$x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H$$

For a very simple example of a proof, see Fig. 4. This proof is still not very interesting.

$$\begin{array}{c} \vdash v^2 \leq 10 \wedge -(-b) > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq 10) \\ \hline [:=]r \vdash [c := 10](v^2 \leq 10 \wedge -(-b) > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq c)) \\ \hline [:=]r \vdash [a := -b][c := 10](v^2 \leq 10 \wedge -a > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq c)) \\ \hline [;]r \vdash [a := -b; c := 10](v^2 \leq 10 \wedge -a > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq c)) \end{array}$$

Figure 4: A simple dynamic example proof in sequent calculus

Incidentally, the proof in Fig. 4 ends with a premise at the top that is identical to the (provable) conclusion at the bottom of Fig. 2. So gluing both proofs together leads to a proof of the conclusion at the bottom of Fig. 4:

$$[a := -b; c := 10](v^2 \leq 10 \wedge -a > 0 \rightarrow b > 0 \wedge (\neg(v \geq 0) \vee v^2 \leq c))$$

Notice how substitutions are applied in the proof shown in Fig. 4 when using proof rule $[:=]r$, which requires quite some care. Observe another important subtlety. The proof

in Fig. 4 ends in a formula mentioning $-(-b) > 0$ while the proof in Fig. 2 starts with a formula mentioning $b > 0$ in the same place. Both formulas are, of course, equivalent, but, in order to glue both proofs, we still need to add a proof rule for this arithmetic transformation. We could add the following proof rule for such a purpose (Exercise 1), but will ultimately decide on adding a more powerful proof rule instead:

$$\frac{\Gamma, \theta > 0 \vdash \Delta}{\Gamma, -(-\theta) > 0 \vdash \Delta}$$

8 Quantifier Proof Rules

When trying to make the proof for the bouncing ball from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) systematic by turning it into a sequent calculus proof, the first propositional step succeeds, then a couple of steps succeed for splitting the hybrid program, but, ultimately, proof rule $[?]r$ produces a quantifier that needs to be handled. And, of course, a mere inspection of the syntax of \mathbf{dL} shows that there are logical operators that have no proof rules yet.

The proof rules for quantifiers come in two sets. The first set is standard in first-order logic. The second set is more unique to first-order logic of real arithmetic (but would also work for other decidable theories). Rules $\exists r, \forall l, \forall r, \exists l$ are standard proof rules for first-order logic, listed in Fig. 5. For explaining these quantifier proof rules, let us first assume for a moment there are no (existential) free variables X_1, \dots, X_n (i.e. $n = 0$, so with $\phi(s)$ instead of $\phi(s(X_1, \dots, X_n))$) and use what is known as the ground calculus for \mathbf{dL} .

$$\begin{array}{ll} (\exists r) \frac{\Gamma \vdash \phi(\theta), \exists x \phi(x), \Delta}{\Gamma \vdash \exists x \phi(x), \Delta} {}_1 & (\forall r) \frac{\Gamma \vdash \phi(s(X_1, \dots, X_n)), \Delta}{\Gamma \vdash \forall x \phi(x), \Delta} {}_2 \\ (\forall l) \frac{\Gamma, \phi(\theta), \forall x \phi(x) \vdash \Delta}{\Gamma, \forall x \phi(x) \vdash \Delta} {}_1 & (\exists l) \frac{\Gamma, \phi(s(X_1, \dots, X_n)) \vdash \Delta}{\Gamma, \exists x \phi(x) \vdash \Delta} {}_2 \end{array}$$

¹ θ is an arbitrary term, often a new (existential) logical variable X .

² s is a new (Skolem) function and X_1, \dots, X_n are all (existential) free logical variables of $\forall x \phi(x)$.

Figure 5: Proof rules for first-order quantifiers

The quantifier proof rules work much as in mathematics. Consider the proof rule $\forall r$, where we want to show a universally quantified property. When a mathematician wants to show a universally quantified property $\forall x \phi(x)$ to hold, he could choose a fresh symbol s (known as a *Skolem function symbol* or Herbrand function symbol in logic) and set out to prove that $\phi(s)$ holds (for s). Once he found a proof for $\phi(s)$, the mathematician would remember that s was arbitrary and his proof did not assume anything special about the value of s . So he would conclude that $\phi(s)$ must indeed hold for all s , and that, hence, $\forall x \phi(x)$ holds true. For example, to show that the square of all

numbers is nonnegative, a mathematician could start out by saying “let s be an arbitrary number”, prove $s^2 \geq 0$ for s , and then conclude $\forall x (x^2 \geq 0)$, since s was arbitrary. Proof rule $\forall r$ essentially makes this reasoning formal. It chooses a *new* (function) symbol s and replaces the universally quantified formula in the succedent by a formula for s (with all free logical variables X_1, \dots, X_n added as arguments, as we explain below, for now, think of $n = 0$ so no arguments). Notice, of course, that it is important to choose a new symbol s that has not been used (in the sequent) before. Otherwise, we would assume special properties about s in Γ, Δ that would not be justified to assume.

Consider proof rule $\exists r$, where we want to show an existentially quantified property. When a mathematician proves $\exists x \phi(x)$, he could directly produce any witness θ for this existential property and prove that, indeed, $\phi(\theta)$, for then he would have shown $\exists x \phi(x)$ with this witness. For example, to show that there is a number whose cube is less than its square, a mathematician could start by saying “let me choose 0.5 and show the property for 0.5”. Then he could prove $0.5^3 < 0.5^2$, because $0.125 < 0.25$, and conclude that there, thus, is such a number, i.e., $\exists x (x^3 < x^2)$, because 0.5 was a perfectly good witness for that. Proof rule $\exists r$ does that. It allows the choice of *any* term θ for x and accepts a proof of $\phi(\theta)$ as a proof of $\exists x \phi(x)$. However note that the claim “ θ is a witness” may turn out to be wrong, for example, the choice 2 for x would have been a pretty bad start for attempting to show $\exists x (x^3 < x^2)$. Consequently, proof rule $\exists r$ keeps both options $\phi(\theta)$ and $\exists x \phi(x)$ in the succedent.⁸ If the proof with θ is successful, the sequent is valid and the part of the proof can be closed successfully. If the proof with θ later turns out to be unsuccessful, another attempt can be used to prove $\exists x \phi(x)$, e.g., by applying rule $\exists r$ again to the same formula $\exists x \phi(x)$ that is still in the succedent, just with another attempt for a different witness θ_2 .

This approach already hints at a practical problem. If we are very smart about our choice of the witness θ , rule $\exists r$ leads to very short and elegant proofs. If not, we may end up going in circles without much progress in the proof. That is why KeYmaera allows you to specify a witness if you can find one (and you should if you can, because that gives much faster proofs) but also allows you to keep going without a witness, as detailed in Sect. 13.

Rules $\forall l, \exists l$ are dual to $\exists r, \forall l$. Consider proof rule $\forall l$, where we have a universally quantified formula in the assumptions (antecedent) that we can use, and not in the succedent, which we want to show. In mathematics, when we know a universal fact, we can use this knowledge for any particular instance. If we know that all positive numbers have a square root, then we can also use the fact that 5 has a square root, because 5 is a positive number. Hence from assumption $\forall x (x > 0 \rightarrow \text{hasSqrt}(x))$ in the antecedent, we can also assume the particular instance $5 > 0 \rightarrow \text{hasSqrt}(5)$ that uses 5 for x . Rule $\forall l$ can produce an instance $\phi(\theta)$ of the assumption $\forall x \phi(x)$ for an arbitrary term θ . Since we may need the universal fact $\forall x \phi(x)$ for multiple instantiations with

⁸KeYmaera does not actually keep $\exists x \phi(x)$ around in the succedent for rule $\exists r$ and, for a fundamental reason [Pla08], does not have to. The same holds for rule $\forall l$, where KeYmaera does not keep $\forall x \phi(x)$ around in the antecedent, because it does not have to. That means, however, that if you conjecture θ to produce the right instance, and your conjecture turns out wrong during the proof, then you have to go back in the proof and undo your instantiation with θ .

is not exactly an instance of the ax rule. So even here we need simple arithmetic to conclude that $0 \leq r \leq r$ is equivalent to $r \geq 0$ by reflexivity and flipping sides, at which point the left premise turns into a formula that can be closed by the ax rule:

$$\frac{ax}{A_{x,v}, r \geq 0 \vdash r \geq 0} *$$

A full formal proof and a KeYmaera proof, thus, need an extra proof step of arithmetic in the left premise. In paper proofs, we will frequently accept such minor steps as abbreviations but always take care to write down the reason. In the above example, we might, for example remark the arithmetic reason “by reflexivity of \leq and by flipping $0 \leq r$ to $r \geq 0$ ”.

The right premise is

$$A_{x,v}, r \geq 0, H - \frac{g}{2}s^2 \geq 0 \vdash B_{H - \frac{g}{2}r^2, -gt}$$

which, when resolving abbreviations turns into

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0, r \geq 0, H - \frac{g}{2}s^2 \geq 0 \vdash 0 \leq H - \frac{g}{2}r^2 \wedge H - \frac{g}{2}r^2 \leq H$$

This sequent proves using $\wedge r$ plus simple arithmetic for the left branch

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0, r \geq 0, H - \frac{g}{2}s^2 \geq 0 \vdash 0 \leq H - \frac{g}{2}r^2$$

resulting from $\wedge r$. We should again remark the arithmetic reason as “by flipping $0 \leq H - \frac{g}{2}r^2$ to $H - \frac{g}{2}r^2 \geq 0$ ”. Some more arithmetic is needed on the right branch resulting from $\wedge r$:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0, r \geq 0, H - \frac{g}{2}s^2 \geq 0 \vdash H - \frac{g}{2}r^2 \leq H$$

where we should remark the arithmetic reason “ $g > 0$ and $r^2 \geq 0$ ”. Finishing the above sequent proof up as discussed for the right premise, thus, shows that $d\mathcal{L}$ formula (3) at the conclusion of the proof is provable.

Throughout this course, you are strongly advised to write down such arithmetic reasons in your paper proofs to justify that the arithmetic is valid. KeYmaera provides a number of ways for proving arithmetic that will be discussed next.

10 Instantiating Real Arithmetic

Real arithmetic can be very challenging. That does not come as a surprise, because cyber-physical systems and the behavior of dynamical systems themselves is challenging. It is amazing that differential dynamic logic reduces challenging questions about CPS to just plain real arithmetic. Of course, that means that you may be left with challenging arithmetic, of quite noticeable computational complexity. This is one part

where you can use your creativity to master challenging verification questions by helping KeYmaera figure them out. While there will soon be more tricks in your toolbox to overcome the challenges of arithmetic, we discuss some of them in this lecture.

Providing instantiations for quantifier rules $\exists r, \forall l$ can significantly speed up real arithmetic decision procedures. The proof in Sect. 9 instantiated the universal quantifier $\forall s$ for an evolution domain constraint by the end point r of the time interval using quantifier proof rule $\forall l$. This is a very common simplification that usually speeds up arithmetic significantly (Note 4). It does not always work, because the instance one guesses may not always be the right one. Even worse, there may not always be a single instance that is sufficient for the proof, but that is a phenomenon that later lectures will examine.

Note 4 (Extreme instantiation). *The proof rule $\forall l$ for universal quantifiers in the antecedent as well as the rule $\exists r$ for existential quantifiers in the succedent allow instantiation of the quantified variable x with any term θ .*

$$(\forall l) \frac{\phi(\theta), \forall x \phi(x) \vdash}{\forall x \phi(x) \vdash}^a$$

The way this rule is used in KeYmaera is with a direct use of weakening rule Wl to hide the quantified formula:

$$(\forall l) \frac{\phi(\theta) \vdash}{\forall x \phi(x) \vdash}^b$$

This instantiation is very helpful if only a single instance θ is important for the argument. Often, an extremal value for x is all it takes for the proof.

This happens often for quantifiers coming from the handling of evolution domains in proof rule $[?]r$. The proof steps that often help then is instantiation of intermediate time s by the end time t :

$$\begin{array}{c} \frac{\frac{\frac{\frac{\frac{\Gamma, t \geq 0 \vdash 0 \leq t \leq t, [x := y(t)]\phi}{\rightarrow l} \quad \Gamma, t \geq 0, [x := y(t)]H \vdash [x := y(t)]\phi}{\forall l} \quad \Gamma, t \geq 0, 0 \leq t \leq t \rightarrow [x := y(t)]H \vdash [x := y(t)]\phi}{\rightarrow r} \quad \Gamma, t \geq 0, \forall 0 \leq s \leq t [x := y(s)]H \vdash [x := y(t)]\phi}{\rightarrow r} \quad \Gamma, t \geq 0 \vdash (\forall 0 \leq s \leq t [x := y(s)]H) \rightarrow [x := y(t)]\phi}{\rightarrow r} \quad \Gamma \vdash t \geq 0 \rightarrow ((\forall 0 \leq s \leq t [x := y(s)]H) \rightarrow [x := y(t)]\phi)}{\forall r} \quad \Gamma \vdash \forall t \geq 0 ((\forall 0 \leq s \leq t [x := y(s)]H) \rightarrow [x := y(t)]\phi) \end{array}$$

Similar instantiations can simplify arithmetic in other cases as well.

^a θ is an arbitrary term, often a new (existential) logical variable X .

^b θ is an arbitrary term, often a new (existential) logical variable X .

11 Weakening Real Arithmetic

Weakening rules Wl, Wr can be useful to hide irrelevant parts of a sequent to make sure they do not be a distraction for real arithmetic decision procedures.

In the proof in Sect. 9, the left premise was

$$A_{x,v}, r \geq 0 \vdash 0 \leq r \leq r$$

The proof of this sequent did not make use of $A_{x,v}$ at all. Here, the proof worked easily. But if $A_{x,v}$ were a very complicated formula, then proving the same sequent might have been very difficult, because our proving attempts could have been distracted by the presence of $A_{x,v}$ and all the lovely assumptions it provides. We might have applied lots of proof rules to $A_{x,v}$ before finally realizing that the sequent proves because of $r \geq 0 \vdash 0 \leq r \leq r$ alone.

The same kind of distraction can happen in decision procedures for real arithmetic, sometimes shockingly so [Pla10, Chapter 5]. Consequently, it often saves a lot of proof effort to simplify irrelevant assumptions away as soon as they have become unnecessary. Fortunately, there already is a proof rule for that purpose called weakening, which we can use on our example from the left premise in the proof of Sect. 9:

$$\text{wl} \frac{r \geq 0 \vdash 0 \leq r \leq r}{A_{x,v}, r \geq 0 \vdash 0 \leq r \leq r}$$

You are generally advised to get rid of assumptions that you no longer need. This will help you manage the relevant facts about your CPS and will also help the arithmetic in KeYmaera to succeed much quicker. Just be careful not to hide an assumption that you still need. But if you accidentally do, that can also be a valuable insight, because you found out what the safety of your system critically depends on.

12 Real Arithmetic

What, in general, can be done to prove real arithmetic? We managed to convince ourselves with ad-hoc arithmetic reasons that the simple arithmetic in the above proofs was fine. But that is neither a proper proof rule nor should we expect to get away with such simple arithmetic arguments for the full complexity of CPS.

Later lectures will discuss the handling of real arithmetic in much more detail. For now, the focus is on the most crucial elements for proving CPS. Differential dynamic logic and KeYmaera make use of a fascinating miracle: the fact that first-order logic of real arithmetic, however challenging it might sound, is perfectly decidable [Tar51]. In a nutshell, the notation $\text{QE}(\phi)$ denotes the use of real arithmetic reasoning on formula ϕ . For a formula ϕ of first-order real arithmetic, $\text{QE}(\phi)$ is a logical formula that is equivalent to ϕ but simpler, because $\text{QE}(\phi)$ is quantifier-free.

Definition 4 (Quantifier elimination). A first-order theory admits *quantifier elimination* if, with each formula ϕ , a quantifier-free formula $\text{QE}(\phi)$ can be associated effectively that is equivalent, i.e. $\phi \leftrightarrow \text{QE}(\phi)$ is valid (in that theory).

Theorem 5 (Tarski [Tar51]). *The first-order logic of real arithmetic admits quantifier elimination and is, thus, decidable.*

The operation QE is further assumed to evaluate ground formulas (i.e., without variables), yielding a decision procedure for closed formulas of this theory (i.e., formulas without free variables). For a closed formula ϕ , all it takes is to compute its quantifier-free equivalent $\text{QE}(\phi)$ by quantifier elimination. The closed formula ϕ is closed, so has no free variables or other free symbols, and neither will $\text{QE}(\phi)$. Hence, ϕ as well as its equivalent $\text{QE}(\phi)$ are either equivalent to *true* or to *false*. Yet, $\text{QE}(\phi)$ is quantifier-free, so which one it is can be found out simply by evaluating the (variable-free) concrete arithmetic in $\text{QE}(\phi)$.

Example 6. Quantifier elimination yields, e.g., the following equivalence by real arithmetic:

$$\text{QE}(\exists x (ax + b = 0)) \equiv (a \neq 0 \vee b = 0).$$

Both sides are easily seen to be equivalent, i.e.

$$\models \exists x (ax + b = 0) \leftrightarrow (a \neq 0 \vee b = 0)$$

because a linear equation with nonzero inhomogeneous part has a solution iff its linear part is nonzero as well. The left-hand side of the equivalence may be hard to evaluate, because it conjectures the existence of an x and it is not clear how we might get such an x . The right-hand side, instead, is trivial to evaluate, because it is quantifier-free and directly says to compare the values of a and b to zero and that an x such that $ax + b = 0$ will exist if and only if $a \neq 0$ or $b = 0$. This is easy to check at least if a, b are either concrete numbers or fixed parameters for your CPS. Then all you need to do is make sure they satisfy these constraints.

Now, if we have quantifiers, QE can remove them for us. But we first need quantifiers. Rules $\forall r, \exists r, \forall l, \exists l$ went through a lot of trouble to get rid of the quantifiers in the first place. Oh my! That makes it kind of hard to eliminate them equivalently later on. Certainly the proof rules in Fig. 5 have not been particularly careful about eliminating quantifiers equivalently. Just think of what might happen if we did try to use $\exists r$ with the wrong witness and then weaken the $\exists x \phi(x)$ away. That is cheaper than quantifier elimination, but hardly as precise and useful.

But if we misplaced a quantifier using the rules from Fig. 5, then all we need to do is to dream it up again and we are in business for eliminating quantifiers by QE. The key to understanding how that works is to recall that the Skolem function symbols were originally universal (and existential logical variables were originally existential).

With the rule $i\forall$, we can reintroduce a universal quantifier for a Skolem term $s(X_1, \dots, X_n)$, which corresponds to a previously universally quantified variable in the succedent or a previously existentially quantified variable in the antecedent. The point of reintroducing the quantifier is that this makes sense when the remaining formulas are first-order in the quantified variable so that they can be handled equivalently by quantifier elimination in real-closed fields. When we have proven the subgoal (with for all X) then

this entails the goal for the particular $s(X_1, \dots, X_n)$. In particular, when we remove a quantifier with $\forall r, \exists l$ to obtain a Skolem term, we can continue with other proof rules to handle the dynamic modalities and then reintroduce the quantifier for the Skolem term with $i\forall$ once quantifier elimination for real arithmetic becomes applicable.

The dual rule $i\exists$ can reintroduce an existential quantifier for a free logical variable that was previously existentially quantified in the succedent or previously universally quantified in the antecedent. Again, this makes sense when the resulting formula in the premise is first-order in the quantified variable X so that quantifier elimination can eliminate the quantifier equivalently. When we remove a quantifier with $\exists r, \forall l$ to obtain a free logical variable, we can continue using other proof rules to handle the dynamic modalities and then reintroduce the quantifier for the free logical variable with $i\exists$ once quantifier elimination is applicable.

$$(i\forall) \frac{\vdash \text{QE}(\forall X (\Phi(X) \vdash \Psi(X)))}{\Phi(s(X_1, \dots, X_n)) \vdash \Psi(s(X_1, \dots, X_n))} \quad 1 \qquad (i\exists) \frac{\vdash \text{QE}(\exists X \bigwedge_i (\Phi_i \vdash \Psi_i))}{\Phi_1 \vdash \Psi_1 \quad \dots \quad \Phi_n \vdash \Psi_n} \quad 2$$

¹ X is a new logical variable. Further, QE needs to be defined for the formula in the premise.

²Among all open branches, free logical variable X only occurs in the branches $\Phi_i \vdash \Psi_i$. Further, QE needs to be defined for the formula in the premise, especially, no Skolem dependencies on X can occur.

In the grand scheme of things, it may not be so particularly apparent why it was a good idea to get rid of the quantifiers in the first place. But if there is a way of getting the quantifiers back and then eliminating them, once and for all, equivalently by QE, then that is pretty reasonable. Can you speculate on the reason why we would first want to mumble quantifiers away with the slightly impoverished rules in Fig. 5 before ultimately coming back with the big steamroller in the form of QE?

Before you read on, see if you can find the answer for yourself.

It is useful to get quantifiers out of the way first using the rules $\forall r, \exists r, \forall l, \exists l$, because further sequent rules only work in the top-level, so quantifiers need to get out of the way before any other proof rules could be applied. And if the formula underneath the quantifier contains modalities with hybrid programs, then that is a bit much to ask from quantifier elimination to solve them for us as well. So the key is to first get rid of quantifiers by using extra Skolem functions or existential variables, work out the proof arguments for the remaining hybrid program modalities and then reintroduce quantifiers by $\forall i, \exists i$ to ask quantifier elimination for the answer to the arithmetic.

Real arithmetic equivalences can be used in differential dynamic logic to eliminate quantifiers as indicated in the proof rules $\forall i, \exists i$. But real arithmetic can also be used to otherwise simplify arithmetic (formally because other symbols can simply be considered as if they were Skolem constants).

13 Quantifier Proof Rules with Free Variables (Expert Reading)

There are two ways of using the proof rules in Fig. 5. One way is as we have explained in Sect. 8 by avoiding free variables X_i altogether and only choose ground terms without variables for instantiations θ in $\exists r, \forall l$. In that case, all Skolem functions used in $\forall r, \exists l$ will have $n = 0$ free logical variables X_1, \dots, X_n as arguments (which is why they are also called Skolem constants in that case). This case is called a *ground calculus*, because free variables are never used and all term instantiations are ground (no free variables).

The other way is to leverage free variables and always use some fresh (existential) logical variable X for instantiation of θ every time $\exists r, \forall l$ are used. This is a free-variable calculus [HS94, Fit96, FM99] where $\exists r, \forall l$ are also called γ -rules and $\forall r, \exists l$ are called δ^+ -rules [HS94], which is an improvement of what is known as the δ -rule [Fit96, FM99]. This case is called a *free-variable calculus*, because instantiations are with free variables. Later in the proof, these free variables can be requantified [Pla08]. The free variables X_1, \dots, X_n in the Skolem terms keep track of the dependencies of symbols and prevent instantiations where we instantiate X_1 by a term such as $s(X_1, \dots, X_n)$ which already depends on X_1 . The ground calculus and free-variable calculus uses of Fig. 5 can also be mixed, which can be a good idea in practice when you can guess values for some witnesses but are unsure about others. These dependency conditions, for example prevent the wrong of reintroducing quantifiers in Fig. 13 and only allow the right way Fig. 6.

14 Creatively Cutting Real Arithmetic

Weakening is not the only propositional proof rule that can help speed your arithmetic. The *cut* rule is not just a curiosity, but can be very helpful in practice. It can speed up real arithmetic a lot when using a cut to replace a difficult arithmetic formula by a simpler one that is sufficient for the proof.

$ \begin{array}{c} \text{i}\forall \text{ is not applicable} \\ \hline \vdash \text{QE}(\exists X (2X + 1 < s(X))) \\ \text{i}\exists \vdash 2X + 1 < s(X) \\ \text{<:=} \vdash \langle x := 2X + 1 \rangle (x < s(X)) \\ \forall r \vdash \forall y \langle x := 2X + 1 \rangle (x < y) \\ \exists r \vdash \exists x \forall y \langle x := 2x + 1 \rangle (x < y) \end{array} $	$ \begin{array}{c} \text{false} \\ \vdash \overline{\text{QE}(\exists X \text{QE}(\forall s (2X + 1 < s)))} \\ \text{i}\exists \vdash \text{QE}(\forall s (2X + 1 < s)) \\ \text{i}\forall \vdash 2X + 1 < s(X) \\ \text{<:=} \vdash \langle x := 2X + 1 \rangle (x < s(X)) \\ \forall r \vdash \forall y \langle x := 2X + 1 \rangle (x < y) \\ \exists r \vdash \exists x \forall y \langle x := 2x + 1 \rangle (x < y) \end{array} $
--	---

Figure 6: **a** Wrong rearrangement attempt for quantifiers**6b:** Correct reintroduction order

For example, suppose $\psi(x)$ is a big and very complicated formula of first-order real arithmetic. Then proving the following formula

$$(x - y)^2 \leq 0 \wedge \psi(x) \rightarrow \psi(y)$$

by just real arithmetic will turn out to be surprisingly difficult and can take ages (even if it ultimately terminates). Yet, thinking about it, $(x - y)^2 \leq 0$ implies that $y = x$, which should make the rest of the proof easy since, $\psi(x)$ should easily imply $\psi(y)$ if $y = x$. How do we exhibit a proof based on these thoughts?

The critical idea to make such a proof work is to use *cut* for a creative cut with the suitable arithmetic. So we choose $y = x$ as the cut formula ϕ in *cut* and proceed as follows:

$ \begin{array}{c} (x - y)^2 \leq 0 \vdash y = x \\ \text{Wr} \vdash (x - y)^2 \leq 0 \vdash y = x, \psi(y) \\ \text{Wl} \vdash (x - y)^2 \leq 0, \psi(x) \vdash y = x, \psi(y) \\ \text{cut} \vdash (x - y)^2 \leq 0, \psi(x) \vdash \psi(y) \\ \wedge l \vdash (x - y)^2 \leq 0 \wedge \psi(x) \vdash \psi(y) \\ \rightarrow r \vdash (x - y)^2 \leq 0 \wedge \psi(x) \rightarrow \psi(y) \end{array} $	$ \begin{array}{c} * \\ ax \vdash \psi(x), y = x \vdash \psi(x) \\ =r \vdash \psi(x), y = x \vdash \psi(y) \\ \text{Wl} \vdash (x - y)^2 \leq 0, \psi(x), y = x \vdash \psi(y) \end{array} $
---	--

Indeed, the left premise proves easily using real arithmetic. The right premise proves comparably easily as well. This proof uses proof rule *=r* that is discussed next. Observe that proofs like this one benefit a lot from weakening to get rid of superfluous assumptions to simplify the resulting arithmetic.

15 Applying Equations by Substitution

The above cut proof uses the following proof rule for applying an equation to a formula ϕ by substituting the left-hand side x of an equation by its right-hand side θ . This substitution is sound, because x is assumed to be equal to θ in the antecedent. The same rule works applies to formulas ϕ that are in the antecedent (*=l*) as well as in the

succedent ($=r$). Obviously, the assumed equality $x = \theta$ has to be in the antecedent for the rule to be sound.

$$(\text{=r}) \frac{\Gamma, x = \theta \vdash \phi(\theta), \Delta}{\Gamma, x = \theta \vdash \phi(x), \Delta} \quad (\text{=l}) \frac{\Gamma, x = \theta, \phi(\theta) \vdash \Delta}{\Gamma, x = \theta, \phi(x) \vdash \Delta}$$

It would be okay to use the equation in the other direction for replacing all occurrences of θ by x , because the equation $\theta = x$ is equivalent to $x = \theta$. Both proof rules, $=r$ and $=l$ apply an equation $x = \theta$ from the antecedent to an occurrence of x in the antecedent or succedent to substitute θ for x . By using the proof rule sufficiently often, multiple occurrences of x in Γ and Δ can be substituted.

Again, quantifier elimination would have been able to prove the same fact, but with significantly more time and effort. So you are advised to exploit these proof shortcuts whenever you find them.

16 Summary

The differential dynamic logic sequent proof rules that we have seen in this lecture are summarized in Fig. 7. They are sound [Pla08]. There are further proof rules of differential dynamic logic that later lectures will examine [Pla08, Pla12b].

Exercises

Exercise 1. Prove soundness of the following special purpose proof rule and use it to continue the proof in Fig. 4 similar to the proof in Fig. 2:

$$(\text{R19}) \frac{\Gamma, \theta > 0 \vdash \Delta}{\Gamma, -(-\theta) > 0 \vdash \Delta}$$

Exercise 2 ()*. Since we are not adding proof rule R19 to the $d\mathcal{L}$ proof calculus, show how you can derive the same proof step using a creative combination of rule $i\forall$ and the other proof rules.

Exercise 3. Prove soundness for the structural and propositional sequent proof rules considered in this lecture.

Exercise 4. Prove soundness for the dynamic sequent proof rules considered in this lecture. You can use a general argument how soundness of the dynamic sequent proof rules follows from soundness of the $d\mathcal{L}$ axioms considered in Lecture 5 on Dynamical Systems & Dynamic Axioms, but first need to prove soundness of those $d\mathcal{L}$ axioms.

Exercise 5 ()*. Generalize the proof rules $[']$ and $[']$ to the case of differential equation systems:

$$x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ H$$

First consider the easier case where $H \equiv \text{true}$.

Note 7.

$$\begin{array}{lll}
(\neg r) \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \neg \phi, \Delta} & (\vee r) \frac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \vee \psi, \Delta} & (\wedge r) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \\
(\neg l) \frac{\Gamma \vdash \phi, \Delta}{\Gamma, \neg \phi \vdash \Delta} & (\vee l) \frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \vee \psi \vdash \Delta} & (\wedge l) \frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta} \\
(\rightarrow r) \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta} & (ax) \frac{}{\Gamma, \phi \vdash \phi, \Delta} & (Wr) \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \\
(\rightarrow l) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \rightarrow \psi \vdash \Delta} & (cut) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta} & (Wl) \frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta} \\
(\langle ; \rangle) \frac{\langle \alpha \rangle \langle \beta \rangle \phi}{\langle \alpha; \beta \rangle \phi} & (\langle *n \rangle) \frac{\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi}{\langle \alpha^* \rangle \phi} & (\langle := \rangle) \frac{\phi_x^\theta}{\langle x := \theta \rangle \phi} \\
([\cdot]) \frac{[\alpha][\beta]\phi}{[\alpha; \beta]\phi} & ([*n]) \frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi} & ([:=]) \frac{\phi_x^\theta}{[x := \theta]\phi} \\
(\langle \cup \rangle) \frac{\langle \alpha \rangle \phi \vee \langle \beta \rangle \phi}{\langle \alpha \cup \beta \rangle \phi} & (\langle ? \rangle) \frac{H \wedge \psi}{\langle ?H \rangle \psi} & (\langle ' \rangle) \frac{\exists t \geq 0 ((\forall 0 \leq \tilde{t} \leq t \langle x := y(\tilde{t}) \rangle H) \wedge \langle x := y(t) \rangle \phi)}{\langle x' = \theta \& H \rangle \phi} \quad 1 \\
([\cup]) \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} & ([?]) \frac{H \rightarrow \psi}{[?H]\psi} & ([']) \frac{\forall t \geq 0 ((\forall 0 \leq \tilde{t} \leq t [x := y(\tilde{t})]H) \rightarrow [x := y(t)]\phi)}{[x' = \theta \& H]\phi} \quad 1 \\
(\exists r) \frac{\Gamma \vdash \phi(\theta), \exists x \phi(x), \Delta}{\Gamma \vdash \exists x \phi(x), \Delta} \quad 2 & (\forall r) \frac{\Gamma \vdash \phi(s(X_1, \dots, X_n)), \Delta}{\Gamma \vdash \forall x \phi(x), \Delta} \quad 3 & \\
(\forall l) \frac{\Gamma, \phi(\theta), \forall x \phi(x) \vdash \Delta}{\Gamma, \forall x \phi(x) \vdash \Delta} \quad 2 & (\exists l) \frac{\Gamma, \phi(s(X_1, \dots, X_n)) \vdash \Delta}{\Gamma, \exists x \phi(x) \vdash \Delta} \quad 3 & \\
(i\forall) \frac{\Gamma \vdash QE(\forall X (\Phi(X) \vdash \Psi(X))), \Delta}{\Gamma, \Phi(s(X_1, \dots, X_n)) \vdash \Psi(s(X_1, \dots, X_n)), \Delta} \quad 4 & (i\exists) \frac{\Gamma \vdash QE(\exists X \bigwedge_i (\Phi_i \vdash \Psi_i)), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \dots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta} \quad 5
\end{array}$$

¹ t and \tilde{t} are fresh logical variables and $\langle x := y(t) \rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value.

² θ is an arbitrary term, often a new (existential) logical variable X .

³ s is a new (Skolem) function and X_1, \dots, X_n are all (existential) free logical variables of $\forall x \phi(x)$.

⁴ X is a new logical variable. Further, QE needs to be defined for the formula in the premise.

⁵Among all open branches, free logical variable X only occurs in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$. Further, QE needs to be defined for the formula in the premise, especially, no Skolem dependencies on X can occur.

Figure 7: Most proof rules of the $d\mathcal{L}$ sequent calculus

$$\begin{array}{lll}
(\langle ; \rangle r) \frac{\Gamma \vdash \langle \alpha \rangle \langle \beta \rangle \phi, \Delta}{\Gamma \vdash \langle \alpha; \beta \rangle \phi, \Delta} & (\langle *^n \rangle r) \frac{\Gamma \vdash \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi, \Delta}{\Gamma \vdash \langle \alpha^* \rangle \phi, \Delta} & (\langle := \rangle r) \frac{\Gamma \vdash \phi_x^\theta, \Delta}{\Gamma \vdash \langle x := \theta \rangle \phi, \Delta} \\
([\cdot] r) \frac{\Gamma \vdash [\alpha][\beta] \phi, \Delta}{\Gamma \vdash [\alpha; \beta] \phi, \Delta} & ([*^n] r) \frac{\Gamma \vdash \phi \wedge [\alpha][\alpha^*] \phi, \Delta}{\Gamma \vdash [\alpha^*] \phi, \Delta} & ([:=] r) \frac{\Gamma \vdash \phi_x^\theta, \Delta}{\Gamma \vdash [x := \theta] \phi, \Delta} \\
(\langle \cup \rangle r) \frac{\Gamma \vdash \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi, \Delta}{\Gamma \vdash \langle \alpha \cup \beta \rangle \phi, \Delta} & (\langle ? \rangle r) \frac{\Gamma \vdash H \wedge \psi, \Delta}{\Gamma \vdash \langle ?H \rangle \psi, \Delta} & \\
([\cup] r) \frac{\Gamma \vdash [\alpha] \phi \wedge [\beta] \phi, \Delta}{\Gamma \vdash [\alpha \cup \beta] \phi, \Delta} & ([?] r) \frac{\Gamma \vdash H \rightarrow \psi, \Delta}{\Gamma \vdash [?H] \psi, \Delta} & \\
(\langle ' \rangle r) \frac{\Gamma \vdash \exists t \geq 0 ((\forall 0 \leq \tilde{t} \leq t \langle x := y(\tilde{t}) \rangle H) \wedge \langle x := y(t) \rangle \phi), \Delta}{\Gamma \vdash \langle x' = \theta \& H \rangle \phi, \Delta} 1 & & \\
([\prime] r) \frac{\Gamma \vdash \forall t \geq 0 ((\forall 0 \leq \tilde{t} \leq t [x := y(\tilde{t})] H) \rightarrow [x := y(t)] \phi), \Delta}{\Gamma \vdash [x' = \theta \& H] \phi, \Delta} 1 & & \\
(\langle ; \rangle l) \frac{\Gamma, \langle \alpha \rangle \langle \beta \rangle \phi \vdash \Delta}{\Gamma, \langle \alpha; \beta \rangle \phi \vdash \Delta} & (\langle *^n \rangle l) \frac{\Gamma, \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi \vdash \Delta}{\Gamma, \langle \alpha^* \rangle \phi \vdash \Delta} & (\langle := \rangle l) \frac{\Gamma, \phi_x^\theta \vdash \Delta}{\Gamma, \langle x := \theta \rangle \phi \vdash \Delta} \\
([\cdot] l) \frac{\Gamma, [\alpha][\beta] \phi \vdash \Delta}{\Gamma, [\alpha; \beta] \phi \vdash \Delta} & ([*^n] l) \frac{\Gamma, \phi \wedge [\alpha][\alpha^*] \phi \vdash \Delta}{\Gamma, [\alpha^*] \phi \vdash \Delta} & ([:=] l) \frac{\Gamma, \phi_x^\theta \vdash \Delta}{\Gamma, [x := \theta] \phi \vdash \Delta} \\
(\langle \cup \rangle l) \frac{\Gamma, \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi \vdash \Delta}{\Gamma, \langle \alpha \cup \beta \rangle \phi \vdash \Delta} & (\langle ? \rangle l) \frac{\Gamma, H \wedge \psi \vdash \Delta}{\Gamma, \langle ?H \rangle \psi \vdash \Delta} & \\
([\cup] l) \frac{\Gamma, [\alpha] \phi \wedge [\beta] \phi \vdash \Delta}{\Gamma, [\alpha \cup \beta] \phi \vdash \Delta} & ([?] l) \frac{\Gamma, H \rightarrow \psi \vdash \Delta}{\Gamma, [?H] \psi \vdash \Delta} & \\
(\langle ' \rangle l) \frac{\Gamma, \exists t \geq 0 ((\forall 0 \leq \tilde{t} \leq t \langle x := y(\tilde{t}) \rangle H) \wedge \langle x := y(t) \rangle \phi) \vdash \Delta}{\Gamma, \langle x' = \theta \& H \rangle \phi \vdash \Delta} 1 & & \\
([\prime] l) \frac{\Gamma, \forall t \geq 0 ((\forall 0 \leq \tilde{t} \leq t [x := y(\tilde{t})] H) \rightarrow [x := y(t)] \phi) \vdash \Delta}{\Gamma, [x' = \theta \& H] \phi \vdash \Delta} 1 & &
\end{array}$$

¹ t and \tilde{t} are fresh logical variables and $\langle x := y(t) \rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value.

Figure 8: Dynamic proof rules of $\text{d}\mathcal{L}$ sequent calculus (left and right rules corresponding to the symmetric rules in Fig. 3)

References

- [And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer, 2nd edition, 2002.
- [Bus98] Samuel R. Buss. An introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, chapter 1, pages 1–78. Elsevier, 1998.
- [Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, 2nd edition, 1996.
- [FM99] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic*. Kluwer, Norwell, MA, USA, 1999.
- [Gen35a] Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Math. Zeit.*, 39(2):176–210, 1935.
- [Gen35b] Gerhard Gentzen. Untersuchungen über das logische Schließen. II. *Math. Zeit.*, 39(3):405–431, 1935.
- [Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
- [HS94] Reiner Hähnle and Peter H. Schmitt. The liberalized δ -rule in free variable semantic tableaux. *J. Autom. Reasoning*, 13(2):211–221, 1994.
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:[10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.

Lecture Notes on Control Loops & Invariants

André Platzer

Carnegie Mellon University
Lecture 7

1 Introduction

[Lecture 3 on Choice & Control](#) demonstrated how important control is in CPS and that control loops are a very important feature for making this control happen. Without loops, CPS controllers are limited to short finite sequences of control actions, which are rarely sufficient to get our CPS anywhere. With loops, CPS controllers shine, because they can inspect the current state of the system, take action to control the system, let the physics evolve, and then repeat these steps in a loop over and over again to slowly get the state where the controller wants the system to be. Loops truly make feedback happen, by enabling a CPS to sense state and act in response to that over and over again. Think of programming a robot to drive on a highway. Would you be able to do that without some means of repetition or iteration as in repeated control? Probably not, because you'll need to write a CPS program that monitors the traffic situation frequently and reacts in response to what the other cars do on the highway. There's no way of telling ahead of time, how often the robot will need to change its mind when its driving a car on a highway.

Hybrid programs' way of exercising repetitive control actions is the repetition operator $*$ that can be applied to any hybrid program α . The resulting hybrid program α^* repeats α any number of times, nondeterministically. That may be zero times or 1 time or 10 times or

Now, the flip side of the fact that control loops are responsible for a lot of the power of CPS is that they can also be tricky to analyze and fully understand. After all, what a system does in just one step is easier to get a handle on than to understand what it will do in the long run when the CPS is running for any arbitrary amount of time. This is the CPS analogue of the fact that ultra-short-term predictions are often much easier

than long-term predictions. It is easy to predict the weather a second into the future but much harder to predict next week's weather.

The main insight behind the analysis of loops in CPS is to reduce the (complicated) analysis of their long-term global behavior to a simpler analysis of their local behavior for one control cycle. This principle significantly reduces the analytic complexity of loops in CPS. It leverages invariants, i.e. aspects of the system behavior that do not change as time progresses, so that our analysis can rely on them no matter how long the system already evolved. Invariants turn out to also lead to an important design principle for CPS, even more so than in programs [PCL11]. The significance of invariants in understanding CPS is not a coincidence, because the study of invariants (just of other mathematical structures) is also central to a large body of mathematics.

More information can be found in [Pla12b, Pla12a] as well as [Pla10, Chapter 2.5.2, 2.5.4].

The most important learning goals of this lecture are:

Modeling and Control: We develop a deeper understanding of control loops as a core principle behind CPS that is ultimately underlying all feedback mechanisms in CPS control. This lecture also intensifies our understanding of the dynamical aspects of CPS and how discrete and continuous dynamics interact.

Computational Thinking: This lecture extends the rigorous reasoning approach from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) to systems with repetitions. This lecture is devoted to the development of rigorous reasoning techniques for CPS models with repetitive control loops or other loopy behavior, a substantially nontrivial problem in theory and practice. Without understanding loops, there is no hope of understanding the repetitive behavior of feedback control principles that are common to almost all CPS. Understanding such behavior can be tricky, because so many things can change in the system and its environment over the course of the runtime of even just a few lines of code if that program runs repeatedly to control the behavior of a CPS. That is why the study of *invariants*, i.e. properties that do not change throughout the execution of the system are crucial for their analysis. Invariants constitute the single most insightful and most important piece of information about a CPS. As soon as we understand the invariants of a CPS, we almost understand everything about it and will even be in a position to design the rest of the CPS around this invariant, a process known as design-by-invariant principle. Identifying and expressing invariants of CPS models will be a part of this lecture as well.

The first part of the lecture shows a careful and systematic development of the invariants, discussing some proof rules and proof principles of more general interest along the way. The second part of the lecture focuses on invariants.

Another aspect of today's lecture is the important concept of global proof rules, which have global premises rather than the local premises from the previous sequent proof rules.

CPS Skills: We will develop a better understanding of the semantics of CPS models by understanding the core aspects of repetition and relating its semantics to cor-

responding reasoning principles. This understanding will lead us to develop a higher level of intuition for the operational effects involved in CPS by truly understanding what control loops fundamentally amount to.

2 Control Loops

Recall the little acrophobic bouncing ball from [Lecture 4 on Safety & Contracts](#).

$$\begin{aligned}
 & @requires(0 \leq x \wedge x = H \wedge v = 0) \\
 & @requires(g > 0 \wedge 1 \geq c \geq 0) \\
 & @ensures(0 \leq x \wedge x \leq H) \\
 & (x' = v, v' = -g \ \& \ x \geq 0; \\
 & \text{if}(x = 0) v := -cv)^*
 \end{aligned} \tag{1}$$

The contracts above have been augmented with the ones that we have identified in [Lecture 4](#) by converting the initial contract specification into a logical formula in differential dynamic logic and then identifying the required assumptions to make it true in all states:

$$\begin{aligned}
 & 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\
 & \quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) v := -cv)^* \right] (0 \leq x \wedge x \leq H)
 \end{aligned} \tag{2}$$

Because we did not want to be bothered by the presence of the additional `if-then-else` operator, which is not officially part of the minimal set of operators that differential dynamic logic **dL** provides, we simplified (2) to:

$$\begin{aligned}
 & 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\
 & \quad \left[(x' = v, v' = -g \ \& \ x \geq 0; (?x = 0; v := -cv \cup ?x \neq 0))^* \right] (0 \leq x \wedge x \leq H)
 \end{aligned} \tag{3}$$

In [Lecture 4](#), we had an informal understanding why (3) is valid (true in all states), but no formal proof, albeit we proved a much simplified version of (3) in which we simply threw away the loop. Such ignorance is clearly not a correct way of understanding loops. Let's make up for that now by properly proving (3) in the **dL** calculus.

Yet, before going for a proof of this bouncing ball property, however much the bouncing ball may long for it, let us first take a step back and understand the role of loops in more general terms. Their semantics has been explored in [Lecture 3 on Choice & Control](#) and more formally in [Lecture 5 on Dynamical Systems & Dynamic Axioms](#).

The little bouncing ball had a loop in which physics and its bounce control alternated. The bouncing ball desperately needs a loop for it wouldn't know ahead of time how often it would bounce today. When falling from great heights, it bounces quite a bit. The bouncing ball also has a controller, albeit a rather impoverished one. All it could do is inspect the current height, compare it to the ground floor (at height 0) and, if $x = 0$, flip its velocity vector around after a little damping by factor c . That is not a whole lot of

flexibility for control choices, but the bouncing ball was still rather proud to serve such an important role in controlling the ball's behavior. Indeed, without the control action, the ball would never bounce back from the ground but would keep on falling forever—what a frightful thought for the acrophobic bouncing ball. On second thought, the ball would, actually, not even fall for very long without its controller, because of the evolution domain $x \geq 0$ for physics $x'' = -g \ \& \ x \geq 0$, which would only allow physics to evolve for time zero if the ball is already at height 0, because gravity would otherwise try to pull it further down, except that the $x \geq 0$ constraint won't have it. So, in summary, without the bouncing ball's control statement, it would simply fall and then lie flat on the ground without time being allowed to proceed. That would not sound very reassuring and certainly not as much fun as bouncing back up, so the bouncing ball is really quite proud of its control.

This principle is not specific to the bouncing ball, but, rather, quite common in CPS. The controller performs a crucial task, without which physics would not evolve in the way that we want it to. After all, if physics did already always do what we want it to without any input from our side, we would not need a controller for it in the first place. Hence, control is crucial and understanding and analyzing its effect on physics one of the primary responsibilities in CPS.

Before proving (3), we apply one more simplification that we have also done in [Lecture 5](#), just to save space on the page. We boldly drop the evolution domain constraint and make up for it by modifying the condition in the second test (Exercise 1):

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \rightarrow \\ \left[(x' = v, v' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^* \right] (0 \leq x \wedge x \leq H) \quad (4)$$

Hold on, why is that okay? Doesn't our previous investigation say that the ball could suddenly fall through the cracks in the floor if physics insists on evolving for hours before giving the poor bouncing ball controller a chance to react? To make sure the bouncing ball does not panic in light of this threat, solve Exercise 1 to investigate.

But, even if we took the evolution domain constraint away from it, notice what the bouncing ball in (4) proudly possess in comparison to the impoverished single-hop ball from [Lecture 5](#): it still has a repetition. And the little acrophobic bouncing ball is certainly mighty proud of its repetition, for this is the only way it could ever bounce and bounce again rather than just bounce and have the world end right after.

3 Proofs of Loops

There is a loop in the HP of the modality in (4). As we have seen, its behavior is crucial to the bouncing ball. So let's prove to understand what it does and to see whether we have to be just as nervous as the bouncing ball about losing it to the earth (if postcondition $0 \leq x$ is not ensured) or to the sky (if $x \leq H$ is not ensured).

Abbreviations have served us well in trying to keep proofs onto one page:

$$\begin{aligned} A_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \\ B_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H \\ (x'' = -g) &\stackrel{\text{def}}{=} (x' = v, v' = -g) \end{aligned}$$

With these abbreviations, the bouncing ball formula (4) turns into:

$$A_{x,v} \rightarrow [(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v} \quad (4)$$

This formula is swiftly turned into the sequent at the top using proof rule $\rightarrow r$:

$$\frac{A_{x,v} \vdash [(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}}{\rightarrow r \vdash A_{x,v} \rightarrow [(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}}$$

This leaves a loop to be worried about. Inspecting our $d\mathcal{L}$ proof rules from [Lecture 6 on Truth & Proof](#) there is exactly one that addresses loops:

$$([*n]) \frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi}$$

Using this one to continue the sequent derivation proceeds as follows:

$$\frac{\begin{array}{c} * \\ A_{x,v} \vdash B_{x,v} \end{array} \quad \frac{[*r] A_{x,v} \vdash [x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0)][(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}}{[*n] A_{x,v} \vdash B_{x,v} \wedge [x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0)][(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}}}{\wedge r \vdash A_{x,v} \vdash [(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}}$$

The left subgoal that results from using $\wedge r$ closes by very simple arithmetic. The right subgoal is more of a challenge to prove. We can solve the differential equation and proceed using $[*]r$, which will produce a quantifier that $\forall r$ can handle and leaves us with a sequent that we need to consider further to prove.

4 Loops of Proofs

After a lot of proof effort, the above sequent prove continues so that the first modalities

$$\dots [x'' = -g][?x = 0; v := -cv \cup ?x \geq 0]\psi$$

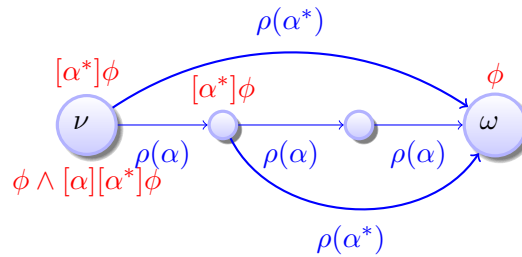
can be handled. But there is still a loop in the postcondition ψ , which is

$$\psi \equiv [(x'' = -g; (?x = 0; v := -cv \cup ?x \geq 0))^*] B_{x,v}$$

How can we prove that postcondition, then? Investigating our proof rules, there is exactly one that addresses loops: $[*n]r$ again. If we use $[*n]r$ again, what will happen?

Recall the loop semantics from [Lecture 3 on Choice & Control](#) and its unwinding axiom from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#):

$$\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \quad \text{with} \quad \alpha^{n+1} \equiv \alpha^n; \alpha \text{ and } \alpha^0 \equiv ?true$$



Lemma 1 ($[*]$ soundness). *The iteration axiom is sound:*

$$([*]) \quad [\alpha^*]\phi \leftrightarrow \phi \wedge [\alpha][\alpha^*]\phi$$

Using proof rule $[*n]r$ on the succedent of a sequent has the same effect as using axiom $[*]$ from left-hand side to right-hand side. Axiom $[*]$ can be used to turn a formula

$$A \rightarrow [\alpha^*]B \tag{5}$$

into

$$A \rightarrow B \wedge [\alpha][\alpha^*]B$$

as we did in the bouncing ball half-proof above. What happens if we use that axiom $[*]$ again?

Recall that, unlike sequent proof rules such as $[*n]r$, axioms do not say where they can be used, so we might as well use them anywhere in the middle of the formula. Hence using axiom $[*]$ on the inner loop yields:

$$A \rightarrow B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)$$

Let's do that again and use the $[*]$ axiom on the only occurrence of $[\alpha^*]B$ to obtain

$$A \rightarrow B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)) \quad (6)$$

This is all very interesting but won't exactly get us any closer to a proof, because we could keep expanding the $*$ star forever that way. How do we ever break out of this loop of never-ending proofs?

Before we get too disillusioned about our progress with $[*]$ so far, notice that (6) still allows us to learn something about α and whether it always satisfies B when repeating α . Since $[*]$ is an equivalence axiom, formula (6) still expresses the same thing as (5), i.e. that B always holds after repeating α when A was true in the beginning. Yet, (6) explicitly singles out the first 3 runs of α . Let's make this more apparent by recalling

Lemma 2 (Boxes distribute over conjunctions). *The following is a sound axiom*

$$([\Box] \wedge [\alpha])(B \wedge \psi) \leftrightarrow [\alpha]B \wedge [\alpha]\psi$$

Using this valid equivalence turns (6) into

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha](B \wedge [\alpha][\alpha^*]B)$$

Using $[\Box] \wedge$ again gives us

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha]([\alpha]B \wedge [\alpha][\alpha][\alpha^*]B)$$

Using $[\Box] \wedge$ once more gives

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha][\alpha^*]B \quad (7)$$

Looking at it this way, (7) could be more useful than the original (5), because, even though both are equivalent, (7) explicitly singles out the fact that B has to hold initially, after doing α once, after doing α twice, and that $[\alpha^*]B$ has to hold after doing α three times. Even if we are not quite sure what to make of the latter $[\alpha][\alpha][\alpha][\alpha^*]B$, because it still involves a loop, we are quite certain how to understand and handle the first three:

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \quad (8)$$

If this formula is not valid, then, certainly, neither is its equivalent (7) and, thus, neither is the original (5). Hence, if we find a counterexample to (8), we disproved (7) and (5). That can actually be rather useful.

Yet, if (8) is still valid, we do not know whether (7) and (5) are, since they involve stronger requirements (B holds after any number of repetitions of α). What can we do then? Simply unroll the loop once more by using $[*]$ on (6) to obtain

$$A \rightarrow B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))) \quad (9)$$

Or, equivalently, use axiom $[*]$ on (7) to obtain the equivalent

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha](B \wedge [\alpha][\alpha^*]B) \quad (10)$$

By sufficiently many uses of axiom $[\wedge]$, (9) and (10) are both equivalent to

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B \quad (11)$$

which we can again examine to see if we can find a counterexample to the first part

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B$$

If yes, we disproved (5), otherwise we use $[*]$ once more.

Note 3 (Bounded model checking). *This process of iteratively unrolling a loop with either axiom $[*]$ or rule $[*n]r$ and then checking the resulting (loop-free) conjuncts is called Bounded Model Checking and has been used very successfully, e.g., in the context of finite-state systems [CBRZ01]. The same principle can be useful to disprove properties of loops in differential dynamic logic by unwinding the loop, checking to see if the resulting formulas have counterexamples and, if not, unroll the loop once more.*

Suppose such a bounded model checking process has been followed to unroll the loop $N \in \mathbb{N}$ times. What can you conclude about the safety of the system?

If a counterexample is found or the formula can be disproved, then we are certain that the CPS is unsafe. If, instead, all but the last conjunct in the N th unrolling of the loop are provable then the system will be safe for $N - 1$ steps, but we cannot conclude anything about the safety of the system after $N - 1$ steps.

5 Breaking Loops for Proofs

Proving properties of loops by unwinding them forever with $[*n]r$ is not a promising strategy, unless we find that the conjecture is not valid after a number of unwindings. Or unless we do not mind being busy with the proof forever for infinitely many proof steps (which would never get our acrophobic bouncing ball off the ground either with the confidence that a safety argument provides). One way or another, we will have to find a way to break the loop apart to complete our reasoning.

Consider the formula (11) again that we got from (5) by unwinding the loop with axiom $[*]$ a number of times and then flattening the formula with the help of axiom $[\wedge]$:

$$A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B \quad (11)$$

Using the propositional sequent rules $\rightarrow r$ and $\wedge r$ on (11) leads to

$$\begin{array}{c}
 \frac{A \vdash [\alpha][\alpha]B \quad \frac{A \vdash [\alpha][\alpha][\alpha]B \quad A \vdash [\alpha][\alpha][\alpha^*]B}{A \vdash [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B}}{A \vdash [\alpha]B \quad \wedge r} \quad \wedge r \\
 \frac{A \vdash B \quad \wedge r}{A \vdash [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B} \quad \wedge r \\
 \frac{\wedge r}{A \vdash B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B} \quad \rightarrow r \\
 \vdash A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B
 \end{array}$$

Let us summarize this notationally by the following

$$\frac{A \vdash B \quad A \vdash [\alpha]B \quad A \vdash [\alpha][\alpha]B \quad A \vdash [\alpha][\alpha][\alpha]B \quad A \vdash [\alpha][\alpha][\alpha^*]B}{\vdash A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B} \rightarrow r, \wedge r, \wedge r, \wedge r, \wedge r$$

to recall that there was a derivation involving one use of $\rightarrow r$ and 4 uses of $\wedge r$ from the four premises to the single conclusion without saying which derivation it was exactly. Mentioning $\wedge r$ 4 times seems a bit repetitive, so simply abbreviate this as:

$$\frac{A \vdash B \quad A \vdash [\alpha]B \quad A \vdash [\alpha][\alpha]B \quad A \vdash [\alpha][\alpha][\alpha]B \quad A \vdash [\alpha][\alpha][\alpha^*]B}{\vdash A \rightarrow B \wedge [\alpha]B \wedge [\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha]B \wedge [\alpha][\alpha][\alpha^*]B} \rightarrow r, \wedge r$$

How could we prove the premises? Sect. 4 investigated one way, which essentially amounts to Bounded Model Checking. Can we be more clever and prove the same premises in a different way? Preferably one that is more efficient and allows us to get the proof over with after finitely many steps?

There is not all that much we can do to improve the way we prove the first premise ($A \vdash B$). We simply have to bite the bullet and do it, armed with all our knowledge of arithmetic from Lecture 6. But it's actually very easy at least for the bouncing ball. Besides, no dynamics have actually happened yet in the first premise, so if we despair in proving this one, the rest cannot become any easier either. For the second premise, there is not much that we can do either, because we will have to analyze the effect of the loop body α running once at least in order to be able to understand what happens if we run α repeatedly.

Yet, what's with the third premise $A \vdash [\alpha][\alpha]B$? We could just approach it as is and try to prove it directly using the $d\mathcal{L}$ proof rules. Alternatively, however, we could try to take advantage of the fact that it is the same hybrid program α that is running in the first and the second modality. Maybe they should have something in common that we can exploit as part of our proof?

How could that work? Can we possibly find something that is true after the first run of α and is all we need to know about the state for $[\alpha]B$ to hold? Can we characterize the intermediate state after the first α and before the second α ? Suppose we manage to do that and identify a formula E that characterizes the intermediate state in this way. How do we use this intermediate condition E to simplify our proof?

Recall the intermediate condition contract version of the sequential composition proof rule from [Lecture 4 on Safety & Contracts](#) that we briefly revisited again in [Lecture 5](#).

$$(R4) \frac{A \rightarrow [\alpha]E \quad E \rightarrow [\beta]B}{A \rightarrow [\alpha; \beta]B}$$

[Lecture 5](#) ended up dismissing the intermediate contract rule [R4](#) in favor of the more general axiom

$$([\alpha; \beta]) [\alpha; \beta]\phi \leftrightarrow [\alpha][\beta]\phi$$

But, let us revisit [R4](#) just the same and see if we can learn something from its way of using intermediate condition E . The first obstacle is that the conclusion of [R4](#) does not match the form we need for $A \vdash [\alpha][\alpha]B$. That's not a problem in principle, because we could use axiom [\[;\]](#) backwards from right-hand side to left-hand side in order to turn $A \vdash [\alpha][\alpha]B$ back into

$$A \vdash [\alpha; \alpha]B$$

and then use rule [R4](#) to generalize with an intermediate condition E in the middle. However, this is what we wanted to stay away from, because using the axioms both forwards and backwards can get our proof search into trouble because we might loop around trying to find a proof forever without making any progress by simply using axiom [\[;\]](#) forwards and then backwards and then forwards again and so on until the end of time. Such a looping proof does not strike us as useful. Instead, we'll adopt a proof rule that has some of the thoughts of [R4](#) but is more general. It is called *generalization* and allows us to prove any stronger postcondition ϕ for a modality, i.e. a postcondition that implies the original postcondition ψ .

Lemma 3 (Generalization rule). *The following is a sound proof rule*

$$([\textit{gen}']) \frac{\Gamma \vdash [\alpha]\phi, \Delta \quad \phi \vdash \psi}{\Gamma \vdash [\alpha]\psi, \Delta}$$

If we apply rule [\[gen'\]](#) on the third premise $A \vdash [\alpha][\alpha]B$ of our bounded model checking style proof attempt with the intermediate condition E for ϕ that we assume to have identified, then we end up with

$$[\textit{gen}'] \frac{A \vdash [\alpha]E \quad E \vdash [\alpha]B}{A \vdash [\alpha][\alpha]B}$$

Let us try to use this principle to see if we can find a way to prove

$$A \rightarrow B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))) \quad (9)$$

Using rules [∧r](#) and [\[gen'\]](#) a number of times for a sequence of intermediate conditions E_1, E_2, E_3 derives:

$$\begin{array}{c}
\frac{A \vdash B \quad \text{[gen']}}{A \vdash [\alpha]E_1} \quad \wedge r \quad \frac{E_1 \vdash B \quad \text{[gen']}}{E_1 \vdash [\alpha]E_2} \quad \wedge r \quad \frac{E_2 \vdash B \quad \text{[gen']}}{E_2 \vdash [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))} \\
\frac{E_2 \vdash [\alpha]E_3 \quad \wedge r \quad \frac{E_3 \vdash B \quad E_3 \vdash [\alpha][\alpha^*]B}{E_3 \vdash B \wedge [\alpha][\alpha^*]B}}{E_2 \vdash [\alpha](B \wedge [\alpha][\alpha^*]B)} \\
\frac{E_1 \vdash [\alpha]E_2 \quad \wedge r \quad \frac{E_2 \vdash [\alpha](B \wedge [\alpha][\alpha^*]B)}{E_2 \vdash B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)}}{E_1 \vdash [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))} \\
\frac{A \vdash [\alpha]E_1 \quad \wedge r \quad \frac{E_1 \vdash [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))}{E_1 \vdash B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B))}}{A \vdash [\alpha](B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)))} \\
\wedge r \quad \frac{A \vdash B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)))}{\rightarrow r \quad \vdash A \rightarrow B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha](B \wedge [\alpha][\alpha^*]B)))}
\end{array}$$

This particular derivation is still not very useful because it still has a loop in one of the premises, which is what we had originally started out with in (5) in the first place. But the derivation hints at a useful way how we could possibly shortcut proofs. To lead to a proof of the conclusion, the above derivation requires us to prove the premises

$$\begin{array}{l}
A \vdash [\alpha]E_1 \\
E_1 \vdash [\alpha]E_2 \\
E_2 \vdash [\alpha]E_3
\end{array}$$

as well as some other premises. What is an easy case to make that happen? What if all the intermediate conditions E_i were the same? Let's assume they are all the same condition E , that is, $E_1 \equiv E_2 \equiv E_3 \equiv E$. In that case, most of the resulting premises actually turn out to be one and the same premise:

$$\begin{array}{l}
E \vdash B \\
E \vdash [\alpha]E
\end{array}$$

except for the two left-most and the right-most premise. *Let us leverage this observation and develop a proof rule for which the same intermediate condition is used for all iterates of the loop.* Furthermore, we would even know the first premise

$$A \vdash [\alpha]E$$

if we could prove that the precondition A implies E :

$$A \vdash E$$

because, we already have $E \vdash [\alpha]E$ as one of the premises.

6 Invariant Proofs of Loops

The condition $E \vdash [\alpha]E$ identified in the previous section seems particularly useful, because it basically says that whenever the system α starts in a state satisfying E , it will stay in E , no matter which of the states in E it was where the system started in the first place. It sounds like the system α^* couldn't get out of E either if it starts in E since all

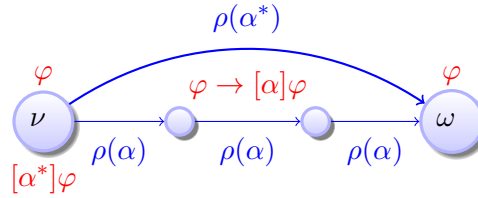
that α^* can do is to repeat α some number of times. But every time we repeat α , the sequent $E \vdash [\alpha]E$ expresses that we cannot leave E that way. So no matter how often our CPS repeats α^* , it will still reside in E .

The other condition that the previous section identified as crucial is $E \vdash B$. And, indeed, if E does not imply the postcondition B that we have been interested in in the first place, then E is a perfectly true invariant of the system, but not a very useful one as far as proving B goes.

What else could go wrong in a system that obeys $E \vdash [\alpha]E$, i.e. where this sequent is valid, because we found a proof for it? Indeed, the other thing that could happen is that E is an invariant of the system that would imply safety, but our system just does not initially start in E , then we still don't know whether it's safe.

Recall the semantics of nondeterministic repetitions from [Lecture 3 on Choice & Control](#)

$$\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \quad \text{with} \quad \alpha^{n+1} \equiv \alpha^n; \alpha \text{ and } \alpha^0 \equiv ?\text{true}$$



Lemma 4 (Induction). *The (loop) induction rule is sound:*

$$(ind') \frac{\Gamma \vdash \varphi, \Delta \quad \varphi \vdash [\alpha]\varphi \quad \varphi \vdash \psi}{\Gamma \vdash [\alpha^*]\psi, \Delta}$$

First observe that the *inductive invariant* φ (which we called E in the previous more concrete examples) occurs in all premises but not in the conclusion of *ind'*. That means, whenever we apply the induction rule *ind'* to a desired conclusion, we get to choose what invariant φ we want to use it for. Good choices of φ will lead to a successful proof of the conclusion. Bad choices of φ will stall the proof, because some of the premises cannot be proved.

The first premise of *ind'* says that the initial state, about which we assume Γ (and that Δ does not hold), satisfies the invariant φ , i.e. the invariant is initially true. The second premise of *ind'* shows that the invariant φ is *inductive*. That is, whenever φ was true before running the loop body α , then φ is always true again after running α . The third premise of *ind'* shows that the invariant φ is strong enough to imply the postcondition ψ that the conclusion was interested in.

Rule *ind'* says that ψ holds after any number of repetitions of α if an invariant φ holds initially (left premise), if invariant φ remains true after one iteration of α from any state where φ was true (middle premise), and if invariant φ finally implies the

desired postcondition ψ (right premise). If φ is true after executing α whenever φ has been true before (middle premise), then, if φ holds in the beginning (left premise), φ will continue to hold, no matter how often we repeat α in $[\alpha^*]\psi$, which is enough to imply $[\alpha^*]\psi$ if φ implies ψ (right premise).

Taking a step back, these three premises look somewhat familiar, because they correspond exactly to the proof steps that the [15-122 Principles of Imperative Computation](#) course used to show that the contract of a function with a $@requires$ contract Γ (and not Δ), $@ensures$ contract ψ , and a loop invariant φ is correct. Now, we have this reasoning in a more general and formally more precisely defined context. We no longer need to appeal to intuition to justify why such a proof rule is fine, but can evoke a soundness proof for *ind'*. We will also no longer be limited to informal arguments to justify invariance for a program but can do actual solid and rigorous formal proofs if we combine proof rule *ind'* with the other proof rules from [Lecture 6 on Truth & Proof](#).

7 A Proof of a Repetitive Bouncing Ball

Now that it understand the principles of how to prove loops in CPS, the bouncing ball is eager to put these skills to use. The ball wants to relieve itself of its acrophobic fears once and for all by conducting a proof that it won't ever have to be afraid of excess heights $> H$ again nor of falling through the cracks in the ground to heights < 0 .

The bouncing ball again use its favorite abbreviations:

$$\begin{aligned} A_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \\ B_{x,v} &\stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H \\ (x'' = \dots) &\stackrel{\text{def}}{=} (x' = v, v' = -g \ \& \ x \geq 0) \end{aligned}$$

Note the somewhat odd abbreviation for the differential equation just to simplify notation, so that the bouncing ball conjecture (3) is:

$$A_{x,v} \rightarrow [(x' = \dots; (?x = 0; v := -cv \cup ?x > 0)^*)] B_{x,v} \quad (3)$$

The first thing that the bouncing ball will need for the proof of (3) is the appropriate choice for the invariant φ to be used in the induction proof rule *ind'*. The bouncing ball will use a $d\mathcal{L}$ formula $E_{x,v}$ for the invariant when instantiating φ in the proof rule *ind'*. But the ball is still a little unsure about how exactly to define that formula $E_{x,v}$, not an unusual situation when trying to master the understanding of CPS. Can you think of a good choice for the formula $E_{x,v}$ to help the bouncing ball?

Before you read on, see if you can find the answer for yourself.

There was trouble in the induction step, because $x \leq H$ could not be proved to be inductive. So the bouncing ball could demand a little less from the invariant and use the following weaker choice for $E_{x,v}$ instead of (12):

$$E_{x,v} \stackrel{\text{def}}{=} x \geq 0 \quad (13)$$

Armed with this new choice for an invariant, the bouncing ball quickly gets to work constructing a new proof for (3). After frantically scribbling a couple of pages with sequent proofs, the bouncing ball experiences a *déjà vu* and notices that its new proof has exactly the same form as the last sequent proof it began. Just with a different choice for the logical formula $E_{x,v}$ to be used as the invariant when applying rule *ind'*. With the choice (13) rather than (12). Fortunately, the bouncing ball already worked with an abbreviation last time it started a proof, so it is actually not surprising after all to see that the proof structure stays exactly the same and that the particular choice of $E_{x,v}$ only affects the premises, not the way the proof unraveled its program statements in the modalities.

Inspecting the 5 premises of the above sequent proof attempt in light of the improved choice (13) for the invariant, the bouncing ball is delighted to find out that the inductive step works out just fine. The height stays above ground always by construction with the evolution domain constraint $x \geq 0$ and is not changed in the subsequent discrete bouncing control. The initial condition ($A_{x,v} \vdash E_{x,v}$) also works out alright, because $0 \leq x$ was among the assumptions in $A_{x,v}$. Only this time, the last premise ($E_{x,v} \vdash B_{x,v}$) falls apart, because $x \geq 0$ is not at all enough to conclude the part $x \leq H$ of the post-condition. What's a ball to do to get itself verified these days?

Before you read on, see if you can find the answer for yourself.

The bouncing ball takes the lesson from Cartesian Doubt to heart and realizes that the invariant needs to transport enough information about the state of the system to make sure the inductive step has a chance of holding true. In particular, the invariant desperately needs to preserve knowledge about the velocity, because how the height changes depends on the velocity (after all the differential equation reads $x' = v, \dots$), so it would be hard to get a handle on height x without first understanding how velocity v changes.

Fine, so the bouncing ball quickly discards the failed invariant choice from (12), which it is no longer so proud of, and also gives up on the weaker version (13), but instead shoots for a stronger invariant of which it would be sure to be inductive and strong enough to imply safety:

$$E_{x,v} \stackrel{\text{def}}{=} x = 0 \wedge v = 0 \quad (14)$$

This time, the bouncing ball learned its lesson and won't blindly set out to prove the property (3) from scratch again, but, rather, be clever about it and realize that it's still going to find the same shape of the sequent proof attempt above, just with a, once again, different choice for the invariant $E_{x,v}$. So the bouncing ball quickly jumps to conclusions and inspects its famous 5 premises of the above sequent proof attempt. This time, the postcondition works out easily and the inductive step works like a charm (no velocity, no height, no motion). But the initial condition is giving it a headache, because there is no reason to believe the ball would initially lie flat on the ground with velocity zero.

For a moment there, the bouncing ball fancied the option of simply changing the initial condition $A_{x,v}$ around to include $x = 0$, because that would make this proof attempt work out swell. But then it realized that this would mean the bouncing ball would from now on be doomed to only start the day at speed zero on the ground, which would not lead to all that much excitement for a cheerful bouncing ball. That would be safe, but a bit too much so for lack of motion.

What, then, is the bouncing ball supposed to do to finally get a proof without crippling its permitted initial conditions?

Before you read on, see if you can find the answer for yourself.

This time, the bouncing ball thinks real hard and has a smart idea. Thinking back of how the lecture notes had motivated the idea of invariants for loops, commonalities of states before and after running the loop body as well as intermediate conditions featured a prominent role in shaping the intuition for invariants. [Lecture 4 on Safety & Contracts](#) had already identified an intermediate condition for the single-hop bouncing ball. Maybe that will prove useful as an invariant, too:

$$E_{x,v} \stackrel{\text{def}}{=} 2gx = 2gH - v^2 \wedge x \geq 0 \quad (15)$$

After all, an invariant is something like a permanent intermediate condition, i.e. an intermediate condition that keeps on working out alright for all future iterations. The bouncing ball is not so sure whether this will work but it seems worth trying.

The shape of the above proof again stays exactly the same, just with a different choice of $E_{x,v}$, this time coming from (15). The remaining 5 premises are then proved easily. The first premise $A_{x,v} \vdash E_{x,v}$ proves easily using $x = H$ and $v = 0$:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \vdash 2gx = 2gH - v^2 \wedge x \geq 0$$

Recalling the usual abbreviations, the second premise $E_{x,v} \vdash [x' = \dots]E_{x,v}$ is

$$2gx = 2gH - v^2 \wedge x \geq 0 \vdash [x' = v, v' = -g \ \& \ x \geq 0](2gx = 2gH - v^2 \wedge x \geq 0)$$

a proof whose pieces we have seen in previous lectures (Exercise 2). The third premise $E_{x,v}, x = 0 \vdash E_{x,-cv}$ is

$$2gx = 2gH - v^2 \wedge x \geq 0, x = 0 \vdash 2gx = 2gH - (-cv)^2 \wedge x \geq 0$$

which would prove easily if we knew $c = 1$. Do we know $c = 1$? No, we do not know $c = 1$, because we only assumed $1 \geq c \geq 0$ in $A_{x,v}$. But we could prove this third premise easily if we would also change the definition of the initial condition $A_{x,v}$ around to include $c = 1$. That may not be the most general possible statement about bouncing balls, but let's happily settle for it. Note that even then, however, we still need to augment $E_{x,v}$ to include $c = 1$ as well, since we otherwise would have lost this knowledge before we need it in the third premise. Having lost critical pieces of knowledge is a phenomenon you may encounter when you are conducting proofs. In that case, you should trace where you lost the assumption in the first place and put it back in. But then you have also learned something valuable about your system, namely which assumptions are crucial for the correct functioning of which part of the system. The fourth premise, $E_{x,v}, x \geq 0 \vdash E_{x,v}$ proves whatever the abbreviations stand for simply using the axiom rule [ax](#). In fact, the bouncing ball could have noticed this earlier already but might have been distracted by its search for a good choice for the invariant $E_{x,v}$. This is but one indication for the fact that it may pay off to take a step back from a proving effort and critically reflect on what all the pieces of the argument rely on exactly. Finally, the fifth premise $E_{x,v} \vdash B_{x,v}$, which is

$$2gx = 2gH - v^2 \wedge x \geq 0 \vdash 0 \leq x \wedge x \leq H$$

proves easily with arithmetic as long as we know $g > 0$. This condition is already included in $A_{x,v}$. But we still managed to forget about that in our invariant $E_{x,v}$. So, again, $g > 0$ should have been included in the invariant $E_{x,v}$, which, overall, should have been defined as

$$E_{x,v} \stackrel{\text{def}}{=} 2gx = 2gH - v^2 \wedge x \geq 0 \wedge c = 1 \wedge g > 0 \quad (16)$$

This is nearly the same definition as (15) except that assumptions about the system parameter choices are carried through. The last two conjuncts are trivial, because neither c nor g changes while the little bouncing ball falls. We, unfortunately, still have to include it in the invariant. This is one of the downsides of working with intermediate condition style proofs such as what we get with rule $\llbracket \text{gen}' \rrbracket$. Later lectures investigate significant simplifications for this nuisance and will enable you to elide the trivial constant part $c = 1 \wedge g > 0$ from the invariant.

For the record, we now really have a full sequent proof of the undamped bouncing ball with repetitions. Exciting!

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 = c \rightarrow \\ \llbracket (x' = v, v' = -g \ \& \ x \geq 0; (?x = 0; v := -cv \cup ?x \geq 0))^* \rrbracket (0 \leq x \wedge x \leq H) \quad (17)$$

Since invariants are a crucial part of a CPS design, you are encouraged to always describe invariants in your hybrid programs: Let's capture the bouncing ball's contracts, which has now been verified by way of proving the corresponding dL formula (17):

$$\begin{aligned} & @requires(0 \leq x \wedge x = H \wedge v = 0) \\ & @requires(g > 0 \wedge c = 1) \\ & @ensures(0 \leq x \wedge x \leq H) \\ & (x' = v, v' = -g \ \& \ x \geq 0; \\ & (?x = 0; v := -cv \cup ?x \geq 0))^* @invariant(2gx = 2gH - v^2 \wedge x \geq 0 \wedge c = 1 \wedge g > 0) \end{aligned} \quad (18)$$

KeYmaera will also make use of the invariants annotated using the `@invariant` contract in hybrid programs to simplify your proof effort. But KeYmaera does not require a list of the constant expressions in the invariant contracts. So the following slight simplification of (18) will suffice:

$$\begin{aligned} & @requires(0 \leq x \wedge x = H \wedge v = 0) \\ & @requires(g > 0 \wedge c = 1) \\ & @ensures(0 \leq x \wedge x \leq H) \\ & (x' = v, v' = -g \ \& \ x \geq 0; \\ & (?x = 0; v := -cv \cup ?x \geq 0))^* @invariant(2gx = 2gH - v^2 \wedge x \geq 0) \end{aligned}$$

Since KeYmaera uses `@invariant` contracts whenever possible, it is a good idea to rephrase (17) by explicitly including the invariant contract as:

$$\begin{aligned}
 &0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 = c \rightarrow \\
 &\quad [(x' = v, v' = -g \ \& \ x \geq 0; \\
 &\quad \quad (?x = 0; v := -cv \cup ?x \geq 0))^* @invariant(2gx = 2gH - v^2 \wedge x \geq 0)] \\
 &\quad (0 \leq x \wedge x \leq H)
 \end{aligned} \tag{19}$$

8 Essentials of Induction & Cuts

The induction rule *ind'* is very useful in practice. But there is also a more elegant and more essential way of stating the induction principle.

Lemma 5 (Induction). *The induction rule is sound:*

$$(ind) \frac{\varphi \vdash [\alpha]\varphi}{\varphi \vdash [\alpha^*]\varphi}$$

The new rule *ind* is clearly a special case of rule *ind'*, obtained by specializing $\Gamma \stackrel{\text{def}}{=} \psi$, $\Delta = .$ (empty), and $\varphi \stackrel{\text{def}}{=} \psi$, in which case the left and right premises of *ind'* are provable directly by *ax* so that only the middle premise remains. If *ind* is a special case of *ind'*, why should we still prefer *ind* from a perspective of essentials? Obviously, *ind* is more fundamental and easier. But if this came at the cost of being less powerful, *ind'* should still be preferred. It turns out that *ind'* is actually a special case of *ind* with a little extra work. This extra work needs a bit of attention but is insightful.

Let's adopt the following variation of the generalization rule:

$$([gen]) \frac{\phi \vdash \psi}{[\alpha]\phi \vdash [\alpha]\psi}$$

For example, using a cut with $\varphi \rightarrow [\alpha^*]\varphi$, rule *ind'* can be derived from *ind* and *[gen]* as follows (using weakening *Wl,Wr* without notice):

$$\begin{array}{c}
 \frac{\varphi \vdash [\alpha]\varphi}{\text{ind} \varphi \vdash [\alpha^*]\varphi} \quad \frac{\Gamma \vdash \varphi, \Delta \quad \text{[gen]} [\alpha^*]\varphi \vdash [\alpha^*]\psi}{\rightarrow I \Gamma, \varphi \rightarrow [\alpha^*]\varphi \vdash [\alpha^*]\psi, \Delta} \\
 \hline
 \text{cut} \frac{\rightarrow R \Gamma \vdash \varphi \rightarrow [\alpha^*]\varphi, \Delta \quad \rightarrow I \Gamma, \varphi \rightarrow [\alpha^*]\varphi \vdash [\alpha^*]\psi, \Delta}{\Gamma \vdash [\alpha^*]\psi, \Delta}
 \end{array}$$

Hence *ind'* is a derived rule, because it can be derived using *ind* and some other rules. Thus, *ind'* is not necessary in theory, but still useful in practice.

Yet, now, in order to derive rule *ind'* out of the more fundamental *ind*, we had to add the revised generalization rule *[gen]*. Is that any easier? Well it is, because *[gen]* actually makes *[gen']* unnecessary by another smart argument using a *cut* with the desired formula $[\alpha]\phi$.

$$\begin{array}{c}
\frac{\Gamma \vdash [\alpha]\phi, \Delta}{\text{Wr} \Gamma \vdash [\alpha]\phi, [\alpha]\psi, \Delta} \quad \frac{\phi \vdash \psi}{\text{[gen]} \frac{[\alpha]\phi \vdash [\alpha]\psi}{\text{Wl,Wr} \Gamma, [\alpha]\phi \vdash [\alpha]\psi, \Delta}} \\
\text{cut} \frac{}{\Gamma \vdash [\alpha]\psi, \Delta}
\end{array}$$

This leaves exactly the premises of rule [gen'] , making [gen'] a derived rule. Whenever we need [gen'] , we could simply expand the proof out in the above form to reduce it just a proof involving [gen] and cut and weakening.

These are two illustrations how creative uses of cuts can suddenly make proves and concepts easier. A phenomenon that we will see in action much more often in this course.

Before you despair that you would have to derive ind' and [gen'] every time you need them: that is not the case. The theorem prover KeYmaera is very well aware of how useful both versions of the proof rules are and has them at your disposal. For theoretical investigations, however, as well as for understanding the truly fundamental reasoning steps, it is instructive to see that ind and [gen] are fundamental, while the others are mere consequences.

9 Summary

This lecture focused on developing and using the concept of invariants for CPS. Invariants enable us to prove properties of CPS with loops, a problem of ubiquitous significance, because hardly any CPS get by without repeating some operations in a control loop. Invariants constitute the single most insightful and most important piece of information about a CPS.

Note 7. This lecture discussed a number of useful proof rules, including:

$$\begin{array}{l}
(\text{ind'}) \frac{\Gamma \vdash \varphi, \Delta \quad \varphi \vdash [\alpha]\varphi \quad \varphi \vdash \psi}{\Gamma \vdash [\alpha^*]\psi, \Delta} \\
([\wedge r]) \frac{\Gamma \vdash [\alpha]B \wedge [\alpha]\psi, \Delta}{\Gamma \vdash [\alpha](B \wedge \psi), \Delta} \\
([\wedge l]) \frac{\Gamma, [\alpha]B \wedge [\alpha]\psi \vdash \Delta}{\Gamma, [\alpha](B \wedge \psi) \vdash \Delta} \\
(\text{[gen']}) \frac{\Gamma \vdash [\alpha]\phi, \Delta \quad \phi \vdash \psi}{\Gamma \vdash [\alpha]\psi, \Delta}
\end{array}$$

The development that led to invariants has some interesting further consequences especially for finding bugs in CPS by unrolling loops and disproving the resulting premises. But this bounded model checking principle is of limited use for ultimately verifying safety, because it only considers the system some finite number of steps in the future. You may find unwinding useful when you are looking for bugs in your CPS, though.

In our effort of helping the bouncing ball succeed with its proof, we saw a range of reasons why an inductive proof may not work out and what needs to be done to adapt the invariant.

Exercises

Exercise 1 (Give bouncing ball back its evolution domain). Explain why the transformation from (3) to (4) was okay in this case.

Exercise 2. Give a sequent proof for

$$2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [x' = v, v' = -g \ \& \ x \geq 0](2gx = 2gH - v^2 \wedge x \geq 0)$$

Does this property also hold if we remove the evolution domain constraint $x \geq 0$? That is, is the following formula valid?

$$2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [x' = v, v' = -g](2gx = 2gH - v^2 \wedge x \geq 0)$$

Exercise 3. To develop an inductive proof rule, we have started systematic unwinding considerations from formula (9) in Sect. 5. In lecture, we started from the form (11) instead and have seen that that takes us to the same inductive principle. Which of the two ways of proceeding is more efficient? Which one produces less premises that are distractions in the argument? Which one has less choices of different intermediate conditions E_i in the first place?

Exercise 4. Could the bouncing ball use any of these formulas as invariants to prove (3)?

$$E_{x,v} \stackrel{\text{def}}{=} (x = 0 \vee x = H) \wedge v = 0$$

$$E_{x,v} \stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H \wedge v^2 \leq 2gH$$

$$E_{x,v} \stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H \wedge v \leq 0$$

Exercise 5. Conduct a sequent proof for (17) without using the generalization rule $\llbracket gen' \rrbracket$.

References

- [CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, 2001.
- [PCL11] Frank Pfenning, Thomas J. Cortina, and William Lovas. Teaching imperative programming with contracts at the freshmen level. 2011.
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.

- [Pla12a] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. [arXiv:1205.4788](https://arxiv.org/abs/1205.4788).
- [Pla12b] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. [doi:10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).

Lecture Notes on Events & Responses

[André Platzer](#)

Carnegie Mellon University
Lecture 8

1 Introduction

[Lecture 3 on Choice & Control](#) demonstrated the importance of control and loops in CPS models, [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) presented a way of unwinding loops iteratively to relate repetition to runs of the loop body, [Lecture 6 on Truth & Proof](#) showed a corresponding way of unwinding loops in sequent calculus, and [Lecture 7 on Control Loops & Invariants](#) finally explained the central proof principle for loops based on induction using invariants.

That has been a lot of attention on loops, but there are even more things to be learned about loops. Not by coincidence, because loops are one of the difficult challenges in CPS. The other difficult challenge comes from the differential equations. If the differential equations are simple and there are no loops, CPS suddenly become easy (they are even decidable [[Pla12a](#)]).

This lecture will focus on how these two difficult parts of CPS interact: how loops interface with differential equations. That interface is ultimately the connection between the cyber and the physical part, which, as we know since [Lecture 2 on Differential Equations & Domains](#), is fundamentally represented by the evolution domain constraints that determine when physics pauses to let cyber look and act.

Today's and the next lecture focuses on two important paradigms for making cyber interface with physics to form cyber-physical systems. Both paradigms played an equally important role in classical embedded systems. One paradigm is that of *event-driven control*, where responses to events dominate the behavior of the system and an action is taken whenever one of the events is observed. The other paradigm is *time-triggered control*, which uses periodic actions to affect the behavior of the system at certain frequencies. Both paradigms follow naturally from an understanding of the hybrid

program principle for CPS. Event-driven control will be studied in this lecture, while time-triggered control will be pursued in the next lecture.

These lecture notes are loosely based on [Pla12b, Pla10].

Based on the understanding of loops from [Lecture 7 on Loops & Invariants](#), the most important learning goals of this lecture are:

Modeling and Control: Today's lecture provides a number of crucial lessons for modeling CPS. We develop an understanding of one important design paradigm for control loops in CPS: event-driven control. This lecture studies ways of developing models and controls corresponding to this feedback mechanism, which will turn out to be surprisingly subtle to model. Today's lecture focuses on CPS models assuming continuous sensing, which assumes that sensor data is available and can be checked all the time.

Computational Thinking: This lecture uses the rigorous reasoning approach from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) and particularly the rigorous reasoning approach for CPS loops from [Lecture 7 on Loops & Control](#) to study CPS models with event-driven control. A discussion of reasoning for CPS models with time-triggered control will be pursued in the next lecture. As a running example, the lecture continues to develop the bouncing ball that has served us so well for conveying subtleties of hybrid system models in an intuitive example. This time, we add control decisions to the bouncing ball, turning it into a ping pong ball, which retains the intuitive simplicity of the bouncing ball, while enabling us to develop generalizable lessons about how to design event-driven control systems correctly. The lecture will also crucially study invariants and show a development of the powerful technique of design-by-invariant in a concrete example. While the lecture could hardly claim showing how to verify CPS models of appropriate scale, the basics laid in this lecture definitely carry significance for numerous practical applications.

CPS Skills: This lecture develops an understanding for the precise semantics of event-driven control, which can often be surprisingly subtle even if superficially simple. This understanding of the semantics will also guide our intuition of the operational effects caused by event-driven control. Finally, the lecture shows a brief first glimpse of higher-level model-predictive control, even if that topic will have to be followed up on in much more detail later in the course.

2 The Need for Control

Having gotten accustomed to the little bouncing ball, this lecture will simply stick to it. Yet, the bouncing ball asks for more action, for it had so far no choice but to wait until it was at ground height $x = 0$. And when its patience paid off so that it finally observed height $x = 0$, then its only action was to make its velocity bounce back up. Frustrated by this limited menu of actions to choose from, the bouncing ball begs for a ping pong

paddle. Thrilled at the opportunities opened up by a ping pong paddle, the bouncing ball first performs some experiments and then settles on using the ping pong paddle high up in the air to push itself back down again. It had high hopes that proper control exerted by the ping pong paddle at just the right moments would allow the ball to go faster without risking the terrified moments inflicted on it by its acrophobic attitude to heights. Setting aside all Münchaussian concerns about how effective ping pong paddles can be for the ball if the ball is using the paddle on itself in light of Newton's third law about opposing forces, let us investigate this situation regardless.¹ After all, the ping-pong-crazy bouncing ball still has what it takes to make control interesting: the dynamics of a physical system and decisions on when to react and how to react to the observed status of the system.

[Lecture 7 on Loops & Invariants](#) developed a sequent proof of the undamped bouncing ball with repetitions:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 = c \rightarrow \\ [(x' = v, v' = -g \ \& \ x \geq 0; (?x = 0; v := -cv \cup ?x \geq 0))^*](0 \leq x \wedge x \leq H) \quad (1)$$

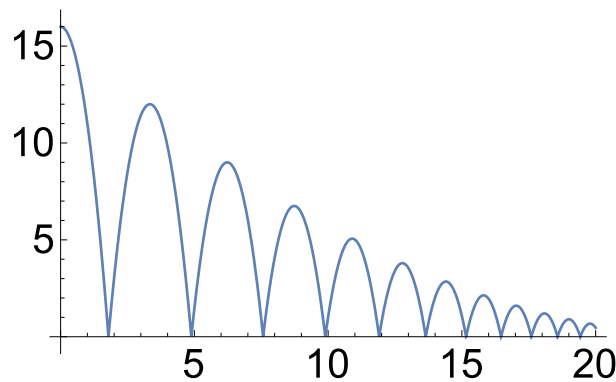


Figure 1: Sample trajectory of a bouncing ball (plotted as position over time)

With this pretty complete understanding of (undamped) bouncing balls, let's examine how to turn the simple bouncing ball into a fancy ping pong ball using clever actuation of a ping pong paddle. The bouncing ball tried to actuate the ping pong paddle in all kinds of directions. But it never knew where it was going to land if it tried the ping pong paddle sideways. So it quickly gave up the thought of using the ping pong paddle sideways. The ball probably got so accustomed to its path of going up and down

¹If you find it hard to imagine a bouncing ball that uses a ping pong paddle to pad itself on its top to propel itself down to the ground again, just step back and consider the case where the ping pong ball has a remote control to activate a device that moves the ping pong paddle. That will do as well, but is less fun. Besides, Baron Münchhausen would be horribly disappointed if we settled for such a simple explanation for the need of control.

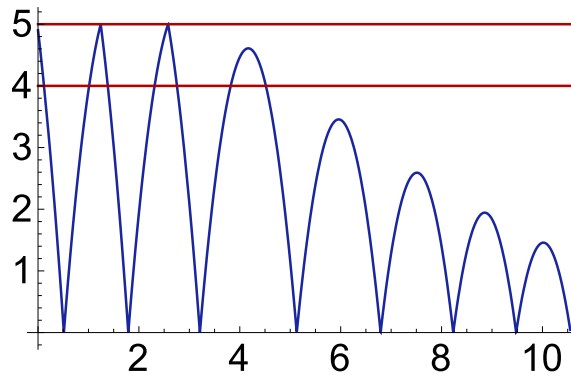


Figure 2: Sample trajectory of a ping pong ball (plotted as position over time) with the indicated ping pong paddle actuation range

on the spot that it embraced the thought of keeping it that way. With the ping pong paddle, it wanted to do the same, just faster.

By making the ping pong paddle move up and down, the bouncing ball ultimately figured out that the ball would go back down pretty fast as soon as it got a pat on the top by the paddle. It also learned that the other direction turned out to be not just difficult but also rather dangerous. Moving the ping pong paddle up when the ball was above it to give it a pat on the bottom was first of all rather tricky, but when it worked would, furthermore, make the ball fly up even higher than before. Yet, that is what the acrophobic bouncing ball did not enjoy at all, so it tries to control the ping pong paddle so that the ping pong paddle only ever bounces the ball down, never up.

As a height that the bouncing ball feels comfortable with, it chooses the magic number 5 and so it wants to establish $0 \leq x \leq 5$ to always hold as its favorite safety condition. The ball further installs the ping pong paddle at a similar height so that it can actuate somewhere between 4 and 5. It exercises great care to make sure it would ever only move the paddle downwards when the ball is underneath, never above, because that would take it frightfully high up. Thus, the effect of the ping pong paddle will only be to reverse the ball's direction. For simplicity, the ball figures that being hit by a ping pong paddle might have a similar effect as being hit by the floor, except with a possibly different bounce factor $f \geq 0$ instead of the damping coefficient c .² So the paddle actuated this way is simply assumed to have the effect $v := -fv$. Since the bouncing ball can decide to use the ping pong paddle as it sees fit (within the ping pong paddle's reach between height 4 and 5), the ping pong model is obtained from the bouncing ball model by adding this additional (nondeterministic) choice to the HP. A sample trajectory for the ping pong ball, where the ping pong paddle is used twice is illustrated in Fig. 2. Observe how the use of the ping pong paddle (here only at height $x = 5$) makes the ball bounce back faster.

Taking these thoughts into account, the ball devises an HP and conjectures safety as

²The real story is quite a bit more complicated, but the bouncing ball does not know any better.

expressed in the following $\text{d}\mathcal{L}$ formula:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \right. \\
 &\quad \left. (?x = 0; v := -cv \cup ?4 \leq x \leq 5; v := -fv \cup ?x \geq 0))^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{2}$$

Having taken the *Principle of Cartesian Doubt* from [Lecture 4 on Safety & Contracts](#) to heart, the aspiring ping-pong ball first scrutinizes conjecture (2) before setting out to prove it. What could go wrong?

For one thing, (2) allows the right control options of using the paddle by $?4 \leq x \leq 5; v := -fv$ but it also always allows the wrong choice $?x \neq 0$ when above ground. Remember that nondeterministic choices are just that: nondeterministic. So if the bouncing ball is unlucky, the HP in (2) could run so that the middle choice is never chosen and, if the ball has a large downwards velocity v initially, it will jump back up higher than 5 even if it was below 5 initially. That scenario falsifies (2) and a concrete counterexample can be constructed correspondingly, e.g., from initial state ν with

$$\nu(x) = 5, \nu(v) = -10^{10}, \nu(c) = \frac{1}{2}, \nu(f) = 1, \nu(g) = 10$$

Despite this setback in its first control attempt, the bouncing ball is thrilled by the extra prospects of a proper control decision for it to be made. So the bouncing ball “only” needs to figure out how to restrict the control decisions such that nondeterminism will only ever take one of the (possibly many) correct control choices, quite a common problem in CPS control. How can the bouncing ball fix this bug in its control and turn itself into a proper ping pong ball? The problem with the controller in (2) is that it permits too much choice, some of which are unsafe. Restricting these choices and making them more deterministic is what it takes to ensure the ping pong paddle is actuated as intended.

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \right. \\
 &\quad \left. (?x = 0; v := -cv \cup ?4 \leq x \leq 5; v := -fv \cup ?x \geq 0 \wedge x < 4 \vee x > 5))^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{3}$$

Recalling the $\text{if}(E) \alpha \text{ else } \beta$ statement, the same system can be modeled equivalently:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \right. \\
 &\quad \left. (?x = 0; v := -cv \cup ?x \neq 0; \text{if}(4 \leq x \leq 5) v := -fv))^* \right] (0 \leq x \leq 5)
 \end{aligned}$$

Or, even shorter as the equivalent

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \right. \\
 &\quad \left. \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{4}$$

Is conjecture (4) valid?

Before you read on, see if you can find the answer for yourself.

3 Events in Control

The problem with the controller in (4) is that, even though it exercises the appropriate control choice whenever the controller runs, the model does not ensure the controller would run at all when needed. The paddle control only runs after the differential equation stops, which could be almost any time. The differential equation is only guaranteed to stop when the ball bounces down to the ground ($x = 0$), because its evolution domain constraint $x \geq 0$ would not be satisfied any longer on its way down. Above ground, the differential equation model does not provide any constraints on how long it might evolve. Recall from [Lecture 2 on Differential Equations & Domains](#) that the semantics of differential equations is nondeterministic in that the system can follow a differential equation *any amount of time*, as long as it does not violate the evolution domain constraints. In particular, the HP in (4) could miss the interesting *event* $4 \leq x \leq 5$ that the ping pong ball's paddle control wanted to respond to. The system might simply skip over that region by following the differential equation $x' = v, v' = -g \ \& \ x \geq 0$ obliviously until the event $4 \leq x \leq 5$ has passed.

How can the HP from (4) be modified to make sure that the event $4 \leq x \leq 5$ will always be noticed and never missed?

Before you read on, see if you can find the answer for yourself.

The “only” way to prevent the system from following a differential equation for too long is to restrict the evolution domain constraint, which is the predominant way to make cyber and physics interact. Indeed, that is what the evolution domain constraint $\dots \& x \geq 0$ in (4) did in the first place. Even though this domain was introduced for different reasons (first principle arguments that light balls never fall through solid ground), its secondary effect was to make sure that the ground controller $?x = 0; v := -cv$ will never miss the right time to take action and reverse the direction of the ball from falling to climbing.

Note 1 (Evolution domains detect events). *Evolution domain constraints of differential equations in hybrid programs can detect events. That is, they can make sure the system evolution stops whenever an event happens on which the control wants to take action. Without such evolution domain constraints, the controller is not necessarily guaranteed to execute but may miss the event.*

Following these thoughts further indicates that the evolution domain somehow ought to be augmented with more constraints that ensure the interesting event $4 \leq x \leq 5$ will never be missed accidentally. How can this be done? Should the event be conjoined to the evolution domain as follows

$$0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \left[(x' = v, v' = -g \& x \geq 0 \wedge 4 \leq x \leq 5; \right. \\ \left. \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^* \right] (0 \leq x \leq 5)$$

Before you read on, see if you can find the answer for yourself.

Of course not! This evolution domain would be entirely counterfactual and require the ball to always be at height between 4 and 5, which is hardly the right physical model. How could the ball ever fall on the ground and bounce back, this way? It couldn't.

Yet, on second thought, the way the event $\dots \& x = 0$ got detected by the HP in the first place was not by including $x = 0$ in the evolution domain constraint, but by including the inclusive limiting constraint $\dots \& x \geq 0$, which made sure the system could perfectly well evolve before the event domain $x = 0$, but that it just couldn't miss the event rushing past the event $x = 0$. What would the inclusion of such an inclusive limiting constraint correspond to for the intended ping pong paddle event $4 \leq x \leq 5$?

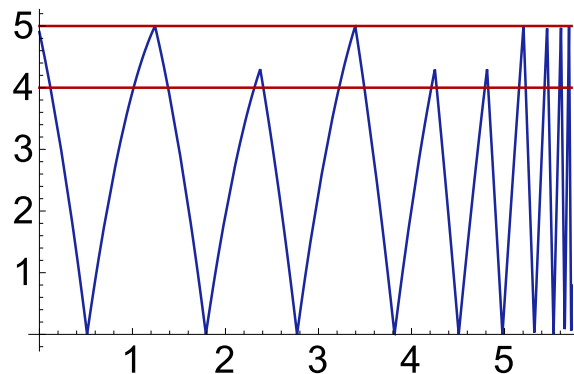
When the ball is hurled up into the sky, the last point at which action has to be taken to make sure not to miss the event $4 \leq x \leq 5$ is $x = 5$. The corresponding inclusive limiting constraint $x \leq 5$ thus should be somewhere in the evolution domain constraint.

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \& x \geq 0 \wedge x \leq 5; \right. \\
 &\quad \quad \left. \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{5}$$

Is this the right model? Is dL formula (5) valid? Will its HP ensure that the critical event $4 \leq x \leq 5$ will not be missed out on?

Before you read on, see if you can find the answer for yourself.

First, however, note that the hybrid program in (5) allows the use of the ping pong paddle anywhere in between the height range $4 \leq x \leq 5$. Its evolution domain constraint enforces that this event $4 \leq x \leq 5$ will be noticed at the latest at height $x = 5$. So when exactly the ping pong paddle is exercised in that range is nondeterministic (even if the control is written deterministically), because the duration of the differential equation is still chosen nondeterministically. This allows the ping pong paddle to be controlled at the last height $x = 5$ or before it reaches height $x = 5$ as in Fig. 3.



Notice, however, that (5) does not make sure that the critical event $4 \leq x \leq 5$ will not be missed out on in the case of a ball that is bouncing up above the lower trigger 4 but starts falling down again already before it exceeds the upper trigger 5 of the event. Such a possible behavior of the ping pong ball was already shown in Fig. 2. Yet, this is not actually problematic, because missing out on the chance of actuating the ping pong paddle in a situation where it is not needed to ensure height control is just missing an opportunity for fun, not missing a critical control choice.

But there is a much deeper problem with (5). So, formula (5) is perfectly valid. But why? Because all runs of the differential equation $x' = v, v' = -g \ \& \ x \geq 0 \wedge x \leq 5$ remain within the safety condition $0 \leq x \leq 5$ *by construction*. None of them are ever allowed to leave the region $x \geq 0 \wedge x \leq 5$, which, after all, is their evolution domain constraint. So formula (5) is trivially safe, because it says that a system that is constrained to not leave $x \leq 5$ cannot leave $x \leq 5$, which is a rather trivial insight since $5 = 5$. A more careful argument involves that, every time around the loop, the postcondition holds trivially, because the differential equation’s evolution constraint maintains it by definition, the subsequent discrete control never changes the only variable x on which the postcondition depends. Hold on, the loop does not have to run but could be skipped over by zero iterations as well. Yet, in that case, the precondition ensures the postcondition, so, indeed, (5) is valid, but only trivially so.

Note 2 (Non-negotiability of Physics). *It is a good idea to make systems safe by construction. For computer programs, that is a great idea. But we need to remember that physics is unpleasantly non-negotiable. So if the only reason why a CPS model is safe is because we forgot to model all relevant behavior of the real physical system, then correctness statements about those inadequate models are not particularly applicable to reality.*

One common cause for counterfactual models are too restrictive evolution domain constraints that rule out physically realistic behavior.

And that is what happened in (5). The bouncing ball got so carried away with trying not to miss the event $4 \leq x \leq 5$ that it forgot to include a behavior in the model that takes place after the event has happened.

Contrast this with the role of the evolution domain constraint $\dots \& x \geq 0$, which came into the system because of physics: to model the guaranteed bouncing back on the ground and to prevent the ball from falling through the ground. The constraint $x \geq 0$ is there for physical reasons. It models the physical limitations of balls which cannot fall through solid soil. The evolution domain constraint $\dots \& x \leq 5$ got added to the ping pong HP for an entirely different reason. It came into play to model what our controller does, and inaptly so, because our feeble attempt ruled out physical behavior that could actually have happened in reality. There is no reason to believe that physics would be so kind to only evolve within $x \leq 5$ just because our controller model wants to respond to an event then. Remember never to do that ever.

Note 3 (Physical constraints versus control constraints). *Some constraints of the system models are included for physical reasons, other constraints are added later to describe the controllers. Take care to ensure not to accidentally limit the behavior of physics when all you meant to do is impose a constraint on your system controller. Physics will not listen to your desire. This applies to evolution domain constraints but also other aspects of your system model such as tests. It is fine, for example, to limit the force that the ping pong paddle is exerting, because that is for the controller to decide. But it is not necessarily a good idea for a controller to limit or change the values of gravity or damping coefficients, because that is rather hard to implement without leaving the planet.*

Let's make up for this modeling mishap by developing a model that has both behaviors, the behaviors before and after the event, just in different continuous programs so that the decisive event in the middle could not accidentally have been missed.

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad [(((x' = v, v' = -g \& x \geq 0 \wedge x \leq 5) \cup (x' = v, v' = -g \& x > 5)); \quad (6) \\
 &\quad \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^*](0 \leq x \leq 5)
 \end{aligned}$$

Instead of the single differential equation with a single evolution domain constraint in (5), the HP in (6) has a (nondeterministic) choice between two differential equations,

actually both the same, with two different evolution domain constraints. The left continuous system is restricted to the lower physics space $x \geq 0 \wedge x \leq 5$, the right continuous system is restricted to the upper physics space $x > 5$. Every time the loop repeats, there is a choice of either the lower physics equation or the upper physics equation. But the system can never stay in these differential equations for too long, because, e.g., when the ball is below 5 and speeding upwards very fast, then it cannot stay in the left differential equation above height 5, so it will have to stop evolving continuously and give the subsequent controller a chance to inspect the state and respond in case the event $4 \leq x \leq 5$ happened.

Now $\text{d}\mathcal{L}$ formula (6) has a much better model of events than the ill-advised (5). Is formula (6) valid?

Before you read on, see if you can find the answer for yourself.

The model in (6) is, unfortunately, quite horribly broken. We meant to split the continuous evolution space into the regions before and after the event $4 \leq x \leq 5$. But we overdid it, because the space is now fractured into two disjoint regions, the lower physics space $x \geq 0 \wedge x \leq 5$ and the upper physics space $x > 5$. How could the ping pong ball ever transition from one to the other? Certainly, as the ball moves upwards within lower physics space $x \geq 0 \wedge x \leq 5$, it will have to stop evolving at $x = 5$ at the latest. But then even if the loop repeats, the ball still could not continue in the upper physics space $x > 5$, because it is not quite there yet. It is an infinitesimal step away from $x > 5$. Of course, the bouncing ball will only ever move continuously along a differential equation. There is no continuous motion that would take the ball from the region $x \geq 0 \wedge x \leq 5$ to the disjoint region $x > 5$. In other words, the HP in (6) has accidentally modeled that there will never ever be a transition from lower to upper physics space nor the other way around, because of an infinitesimal gap in between.

Note 4 (Connectedness & disjointness in evolution domains). *Evolution domain constraints need to be thought out carefully, because they determine the respective regions within which the system can evolve. Disjoint or unconnected evolution domain constraint regions often indicate that the model will have to be thought over again, because there cannot be any continuous transitions from one domain to the other if they are not connected.*

Let's close the infinitesimal gap between $x \geq 0 \wedge x \leq 5$ and $x > 5$ by including the boundary $x = 5$ in both domains:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad [(((x' = v, v' = -g \ \& \ x \geq 0 \wedge x \leq 5) \cup (x' = v, v' = -g \ \& \ x \geq 5)); \quad (7) \\
 &\quad \text{if}(x = 0) \ v := -cv \text{ else if}(4 \leq x \leq 5) \ v := -fv)^*](0 \leq x \leq 5)
 \end{aligned}$$

Now there is a proper separation into lower physics $x \geq 0 \wedge x \leq 5$ and upper physics $x \geq 5$ but the system can be in either physics space at the switching boundary $x = 5$. This makes it possible for the ball to pass from lower physics into upper physics or back, yet only on the boundary $x = 5$, which, in this case, is the only point that the two evolution domain constraints have in common.

Now $\text{d}\mathcal{L}$ formula (7) has a much better model of events than the ill-advised (5). Is formula (7) valid?

Before you read on, see if you can find the answer for yourself.

When the ball is jumping up from the ground, the model in (7) makes it impossible for the controller to miss the event $4 \leq x \leq 5$, because the only evolution domain constraint in the HP that applies at the ground is $x \geq 0 \wedge x \leq 5$. And that evolution domain stops being true above 5. Yet, suppose the ping pong ball was jumping up from the ground following the continuous program in the left choice and then stopped its evolution at height $x = 4.5$, which always remains perfectly within the evolution domain $x \geq 0 \wedge x \leq 5$ and is, thus, allowed. Then, after the sequential composition between the middle and last line of (7), the controller in the last line of (7) runs, notices that the formula $4 \leq x \leq 5$ for the event checking is true, and changes the velocity according to $v := -fv$, corresponding to the assumed effect of a pat with the paddle. That is actually its only choice in such a state, because the controller is deterministic, much unlike the differential equation. Consequently, the velocity has just become negative since it was positive before as the ball was climbing up. So the loop can repeat and the differential equation runs again. Yet, then the differential equation might evolve until the ball is at height $x = 4.25$, which will happen since its velocity is negative. If the differential equation stops then, the controller will run again, determine that $4 \leq x \leq 5$ is true still and so take action to change the velocity to $v := -fv$ again. That will, however, make the velocity positive again, since it was previously negative as the ball was in the process of falling. Hence, the ball will keep on climbing now, which, again, threatens the postcondition $0 \leq x \leq 5$. Will this falsify (7) or is it valid?

Before you read on, see if you can find the answer for yourself.

On second thought, that alone still will not cause the postcondition to evaluate to *false*, because the only way the bouncing ball can evolve continuously from $x = 4.25$ is still by the continuous program in the left choice of (7). And that differential equation is restricted to the evolution domain $x \geq 0 \wedge x \leq 5$, which causes the controller to run before leaving $x \leq 5$. That is, the event $4 \leq x \leq 5$ will again be noticed by the controller so that the ball is ping pong paddle pats the ball back down; see Fig. 4.

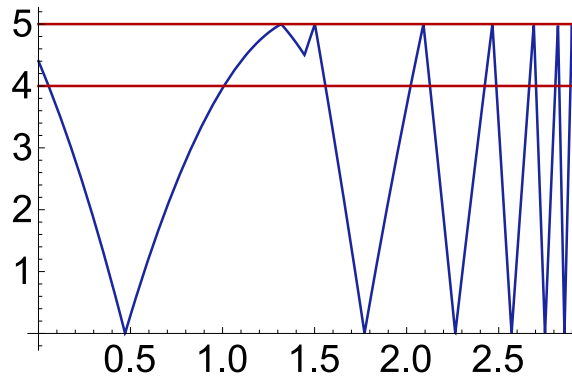


Figure 4: Sample trajectory of a ping pong ball (plotted as position over time) with the controller firing multiple times for the same event

However, the exact same reasoning applies also to the case where the ball successfully made it up to height $x = 5$, which is the height at which any climbing ball has to stop its continuous evolution, because it would otherwise violate the evolution domain $x \geq 0 \wedge x \leq 5$. As soon as that happens, the controller runs, notices that the event $4 \leq x \leq 5$ came true and responds with a ping pong paddle to cause $v := -fv$. If, now, the loop repeats, yet the continuous evolution evolves for duration zero only, which is perfectly allowed, then the condition $4 \leq x \leq 5$ will still be true so that the controller again notices this “event” and responds with ping pong paddle $v := -fv$. That will make the velocity positive, the loop can repeat, the continuous program on the right of the choice can be chosen since $x \geq 5$ holds true, and then the bouncing ball can climb and disappear into nothingness high up in the sky if only its velocity has been large enough. Such a behavior is shown in Fig. 5. The second illustration in Fig. 5 uses the artistic liberty of delaying the second ping pong paddle use just a tiny little bit to make it easier to see the two ping pong paddle uses separately, even if that is not actually quite allowed by the HP model, because such behavior would actually be reflected by a third ping pong paddle use as in Fig. 4.

Ergo, (7) is not valid. What a pity! The poor bouncing ball would still have to be afraid of heights when following the control in (7). How can this problem be resolved?

Before you read on, see if you can find the answer for yourself.

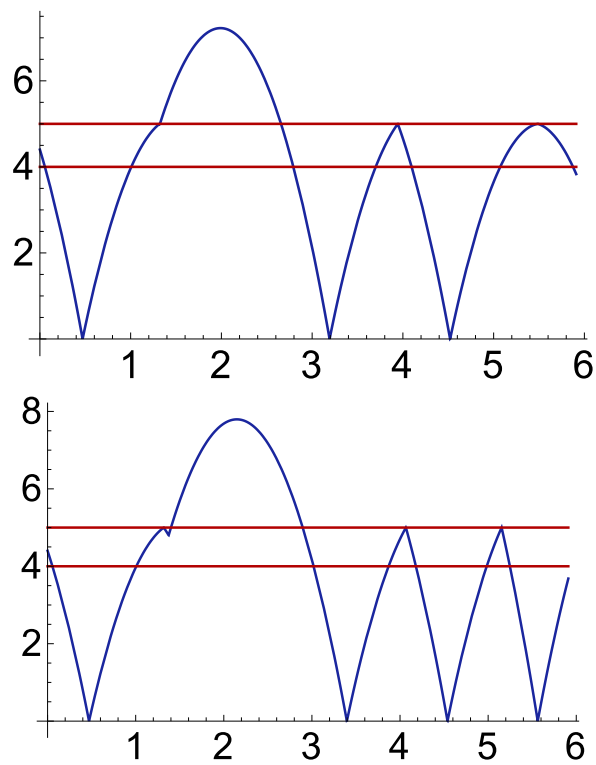


Figure 5: Sample trajectory of a ping pong ball (plotted as position over time) with the controller firing multiple times for the same event on the event boundary $x = 5$ within lower and upper physics

The problem in (7) is that its left differential equation makes sure never to miss out on the event $4 \leq x \leq 5$ but its control may respond to it multiple times. It is not even sure whether each occasion of $4 \leq x \leq 5$ should be called an event. But certainly repeated responses to the same event according to control (7) causes trouble.

Note 5 (Multi-firing of events). *In event-driven control, exercise care to ensure whether you want events to fire only once when they occur for the first time, or whether the system stays safe even if the same event is detected and responded to multiple times in a row.*

One way of solving this problem is to change the condition in the controller to make sure it only responds to the $4 \leq x \leq 5$ event when the ball is on its way up, i.e. when its velocity is not negative ($v \geq 0$). That is what the bouncing ball wanted to ensure in any case. The ping pong paddle should only be actuated downwards when the ball is flying up.

These thoughts lead to the following variation:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad [(((x' = v, v' = -g \ \& \ x \geq 0 \wedge x \leq 5) \cup (x' = v, v' = -g \ \& \ x \geq 5)); \quad (8) \\
 &\quad \text{if}(x = 0) \ v := -cv \ \text{else if}(4 \leq x \leq 5 \wedge v \geq 0) \ v := -fv)^*](0 \leq x \leq 5)
 \end{aligned}$$

Because the paddle action $v := -fv$ will disable the condition $v \geq 0$ for nonzero velocities (Exercise 1), the controller in (8) can only respond once to the event $4 \leq x \leq 5$ to turn the upwards velocity into a downwards velocity, scaled by f . Unlike in (7), this control decision cannot be reverted inadvertently by the controller.

Is dL formula (8) valid?

Before you read on, see if you can find the answer for yourself.

Yes, formula (8) is valid. Note that it is still the case in (8) that, every time around the loop, there will be a nondeterministic choice to evolve within lower physics $x \geq 0 \wedge x \leq 5$ or within upper physics $x \geq 5$. This choice is nondeterministic, so any outcome will be possible. If the left differential equation is chosen, the subsequent continuous evolution must be confined to $x \geq 0 \wedge x \leq 5$ and stop before leaving that lower physics region to give the controller a chance to check for events and respond. If the right differential equation is chosen, the subsequent continuous evolution must be limited to $x \geq 5$ and must stop before leaving that upper physics region to give the controller a chance to inspect. In fact, the only way of leaving the upper physics space is downwards (with velocity $v < 0$), which, unlike in (7), will not trigger a response from the subsequent control in (8), because that controller checks for $v \geq 0$.

How could $d\mathcal{L}$ formula (8) be proved, so that we have unquestionable evidence that it is, indeed, valid? The most critical element of a proof is finding a suitable invariant. What could be the invariant for proving (8)?

Before you read on, see if you can find the answer for yourself.

The formula

$$5 \geq x \geq 0 \tag{9}$$

is an obvious candidate for an invariant. If it is true, it trivially implies the postcondition $0 \leq x \leq 5$ and it holds in the initial state. It is not inductive, though, because a state that satisfies (9) could follow the second differential equation if it satisfies $x \geq 5$. In that case, if the velocity is positive, the invariant (9) would be violated immediately. Hence, at the height $x = 5$, the control has to make sure that the velocity is negative, so that the right differential equation in (8) has to stop immediately. Could (9) be augmented with a conjunction $v \leq 0$ to form an invariant?

$$5 \geq x \geq 0 \wedge v \leq 0$$

No, that would not work either, because the bounce on the ground immediately violates that invariant, because the whole point of bouncing is that the velocity will become positive again. In fact, the controller literally only ensures $v \leq 0$ at the event, which is detected at $x = 5$ at the latest. Gathering these thoughts, it turns out that the dL formula (8) can be proved in the dL calculus using the invariant:

$$5 \geq x \geq 0 \wedge (x = 5 \rightarrow v \leq 0) \tag{10}$$

This invariant retains that the possible range of x is safe but is just strong enough to also remember the correct control choice at the event boundary $x = 5$. It expresses that the ball is either in lower physics space or at the boundary of both physics spaces. But if the ball is at the boundary of the physics spaces, then it is moving downward.

That is the reason why (10) is easily seen to be an invariant of (8). The invariant (8) is initially true, because the ball is initially in range and moving down. The invariant trivially implies the postcondition, because it consists of the postcondition plus an extra conjunction. The inductive step is most easily seen by considering cases. If the position before the loop body ran was $x < 5$, then the only physics possible to evolve is lower physics, which, by construction, implies the conjunct $5 \geq x \geq 0$ from its evolution domain constraint. The extra conjunct $x = 5 \rightarrow v \leq 0$ is true after the loop body ran, because, should the height actually be 5, which is the only case for which this extra conjunct is not already vacuously true, then the controller made sure to turn the velocity downwards by checking $4 \leq x \leq 5 \wedge v \geq 0$ and negating the velocity. If the position before the loop body ran was $x \geq 5$ then the invariant (10) implies that the only position it could have had is $x = 5$ in which case either differential equation could be chosen. Except, if the first differential equation is chosen, the reasoning for inductiveness is as for the case $x < 5$. If the second differential equation is chosen, then the invariant (10) implies that the initial velocity is $v \leq 0$, which implies that the only possible duration that keeps the evolution domain constraint $x \geq 5$ of the upper physics true is duration 0, after which nothing changed so the invariant still holds.

Observe how the scrutiny of a proof, which necessitated the transition from the broken invariant (9) to the provable invariant (10), would have pointed us to subtleties with events and how ping pong balls would become unsafe if they fired repeatedly. We

found these issues out by careful formal modeling with our “safety first” approach and a good dose of Cartesian Doubt. But had we not noticed it, the proof would have not let us get away with such oversights, because the (unreflected) invariant candidate (9) would not have worked, nor would the broken controller (7) have been provable.

Finally, recall that (global) invariants need to be augmented with the usual mundane assumptions about the unchanged variables, like $c \geq 0 \wedge g > 0 \wedge f \geq 0$.

See [«Event-driven ping pong ball KeYmaera model»](#)

The model that (8) and the other controllers in this section adhere to is called event-driven control or sometimes also called event-driven architecture.

Note 6 (Event-driven control). *One common paradigm for designing controllers is event-driven control, in which the controller runs in response to certain events that happen in the system. The controller could possibly run under other circumstances as well—when in doubt, the controller simply skips over without any effect if it does not want to change anything about the behavior of the system. But event-driven controllers assume they will run for sure whenever certain events in the system happen.*

These events cannot be all too narrow, or else the system will not be implementable, though. For example, it is nearly impossible to build a controller that responds exactly at the point in time when the height of the bouncing ball is $x = 4.12345$. Chances are high that any particular execution of the system will have missed this particular height. Care must be taken in event-driven design models also that the events do not inadvertently restrict the evolution of the system to the behavioral cases outside or after the events have happened. Those executions must still be verified.

Are we sure in model (8) that events are taken into account faithfully? That depends on what exactly we mean by an event like $4 \leq x \leq 5$. Do we mean that this event happens for the first time? Or do we mean every time this event happens? If multiple successive runs of the ping pong ball’s controller see this condition satisfied, do these count as the same or separate instances of that event happening? Comparing the validity of (7) with the non-validity of (7) illustrates that these subtleties can have considerable impact on the system. Hence, a precise understanding of events and careful modeling is required.

The controller in (8) only takes an action for event $4 \leq x \leq 5$ when the ball is on the way up. Hence, the evolution domain constraint in the right continuous evolution is $x \geq 5$. Had we wanted to model the occurrence of event $4 \leq x \leq 5$ also when the ball is on its way down, then we would have to have a differential equation with evolution domain $x \geq 4$ to make sure the system does not miss $4 \leq x \leq 5$ when the ball is on its way down either, without imposing that it would have to notice $x = 5$ already. This could be achieved by splitting the evolution domain regions appropriately, but was not necessary for (8) since it never responds to balls falling down, only those climbing up.

Note 7 (Subtleties with events). *Events are a slippery slope and great care needs to be exercised to use them without introducing an inadequate executional bias into the model.*

There is a highly disciplined way of defining, detecting, and responding to general events in differential dynamic logic based on the there and back again axiom of differential dynamic logic [Pla12a]. That is, however, much more complicated than the simpler account shown here.

Finally, notice that the proof for (8) was entirely independent of the differential equation and just a consequence of the careful choices of the evolution domain constraint to reflect the events of interest as well as getting the controller responses to these events right. That is, ultimately, the reason why the invariant (10) could be so simple. This also often contributes to making event-driven controllers are easier to get right.

Note 8 (Correct event-driven control). *As long as the controller responds in the right ways to the right events, event-driven controllers can be built rather systematically and are easier to prove correct. But beware! You have to get the handling of events right, otherwise you only end up with a proof about counterfactual physics, which is not at all helpful since your actual CPS then follows an entirely different kind of physics.*

Note 9 (Physics versus control). *Observe that some parts of hybrid program models represent facts and constraints from physics, other parts represent controller decisions and choices. It is a good idea to keep the facts straight and remember which part of a hybrid program model comes from which. Especially, whenever a constraint is added, because of a controller decision, it is good practice to carefully think through what happens if this is not the case. That is how we ended up splitting physics into different evolution domain constraints, for example.*

Partitioning the hybrid program in (8) into the parts that come from physics (typographically marked like **physics**) and the parts that come from control (typographically marked like **control**) leads to:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad [(((x' = v, v' = -g \ \& \ x \geq 0 \wedge x \leq 5) \cup (x' = v, v' = -g \ \& \ x \geq 5)); \quad (8) \\
 &\quad \text{if}(x = 0) \ v := -cv \text{ else if}(4 \leq x \leq 5 \wedge v \geq 0) \ v := -fv)^*](0 \leq x \leq 5)
 \end{aligned}$$

Note that there could have been a second evolution domain constraint $x \geq 0$ for the physics in the second differential equation, but that evolution domain constraint was elided, because it is redundant in the presence of the evolution domain constraint $x \geq 5$ coming from the controller. Observe that only controller constraints have been added compared to the initial physical model of the bouncing ball (1) that was entirely physics.

4 Summary

This lecture studied event-driven control, which is one important principle for designing feedback mechanisms in CPS and embedded systems. The lecture illustrated the most important aspects for a running example of a ping pong ball. Even if the ping pong ball may not be the most exciting application of control in the world, the effects and pitfalls of events in control were sufficiently subtle to merit focusing on a simple intuitive case.

Event-driven control assumes that all events are detected perfectly and right away. The event-driven controller in (8) took some precautions by defining the event of interest for using the ping pong paddle to be $4 \leq x \leq 5$. This may look like a big event in space to be noticed in practice, except when the ball moves too quickly, in which case the event $4 \leq x \leq 5$ is over rather quickly. However, the model still has $x \leq 5$ as a hard limit in the evolution domain constraint to ensure that the event would never be missed in its entirety as the ball is rushing upwards.

Event-driven control assumes permanent continuous sensing of the event of interest, because the hard limit of the event is ultimately reflected in the evolution domain constraint of the differential equation. This evolution domain constraint is checked permanently according to its semantics ([Lecture 3 on Choice & Control](#)).

Exercises

Exercise 1. Can the ping pong paddle in (8) ever respond to the event $4 \leq x \leq 5$ twice in a row? What would happen if it did?

Exercise 2. Is the following formula an invariant for proving (8)?

$$0 \leq x \leq 5 \wedge (x = 5 \rightarrow v \leq 0) \wedge (x = 0 \rightarrow v \geq 0)$$

Exercise 3. Would the invariant (10) succeed in proving a variation of (8) in which the controller conjunction $\wedge v \geq 0$ is removed? If so explain why. If not, explain which part of the proof will fail.

Exercise 4. Would a generalization of formula (8) be valid in which the assumption $v \leq 0$ on the initial state is dropped? If yes, give a proof. If not, show a counterexample and explain how to fix this problem in a way that leads to a generalization of (8) that is still a valid formula.

Exercise 5. Could we replace the two differential equations in (8) with a single differential equation and a disjunction of their evolution domain constraints instead to retain a valid formula?

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad [((x' = v, v' = -g \ \& \ (x \geq 0 \wedge x \leq 5) \vee x \geq 5); \\ &\quad \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5 \wedge v \geq 0) v := -fv)^*](0 \leq x \leq 5) \end{aligned}$$

Exercise 6. Conduct a sequent proof proving the validity of $\text{d}\mathcal{L}$ formula (8). Track which assumptions are used for which case.

Exercise 7 ()*. Design a variation of the event-driven controller for the ping pong ball that is allowed to use the ping pong paddle within height $4 \leq x \leq 5$ but has a relaxed safety condition that accepts $0 \leq x \leq 2 \cdot 5$. Make sure to only force the use of the ping pong paddle when necessary. Find an invariant and conduct a proof.

References

- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. [doi:10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012. [doi:10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. [arXiv:1205.4788](https://arxiv.org/abs/1205.4788).

Lecture Notes on Reactions & Delays

André Platzer

Carnegie Mellon University
Lecture 9

1 Introduction

[Lecture 7 on Control Loops & Invariants](#) explained the central proof principle for loops based on induction using invariants. [Lecture 8 on Events & Responses](#) studied the important feedback mechanism of event-driven control and made crucial use of invariants for rigorously reasoning about event-driven control loops. Those invariants uncovered important subtleties with events that could be easily missed. In [Lecture 8 on Events & Responses](#), we, in fact, already noticed these subtleties thanks to our “safety first” approach to CPS design, which guided us to exercise the scrutiny of Cartesian Doubt on the CPS model before even beginning a proof.

But, even if the final answer for the event-driven controller for ping pong balls was rather clear and systematic, event-driven control had an unpleasantly large number of modeling subtleties in store for us. Furthermore, event-driven control has a rather high level of abstraction, because it assumes that all events would be detected perfectly and right away in continuous sensing. However, the event-driven model had $x \leq 5$ as a hard limit in the evolution domain constraint to ensure that the event $4 \leq x \leq 5$ would never be missed as the ball is rushing upwards.

As soon as we want to implement such an event detection, it becomes clear that real controller implementations can only perform discrete sensing, i.e. checking sensor data every once in a while at certain discrete points in time, whenever the measurement comes from the sensor and the controller has a chance to run. Most controller implementations would, thus, only end up checking for an event every once in a while, whenever the controller happens to run, rather than permanently as event-driven controllers pretend.

Today’s lecture focuses on the second important paradigm for making cyber interface with physics to form cyber-physical systems. The paradigm of *time-triggered control*,

which uses periodic actions to affect the behavior of the system at certain frequencies. This is to be contrasted with the paradigm from [Lecture 8 on Events & Responses](#) of *event-driven control*, where responses to events dominate the behavior of the system and an action is taken whenever one of the events is observed. Both paradigms play an equally important role in classical embedded systems and both paradigms fall out naturally from an understanding of the hybrid program principle for CPS.

These lecture notes are loosely based on [Pla12, Pla10].

Based on the understanding of loops from [Lecture 7 on Loops & Invariants](#), the most important learning goals of this lecture are:

Modeling and Control: Today's lecture provides a number of crucial lessons for modeling CPS and designing their controls. We develop an understanding of time-triggered control, which is an important design paradigm for control loops in CPS. This lecture studies ways of developing models and controls corresponding to this feedback mechanism, which will turn out to be surprisingly subtle to control. Knowing and contrasting both event-driven and time-triggered feedback mechanisms helps with identifying relevant dynamical aspects in CPS coming from events and reaction delays. Today's lecture focuses on CPS models assuming discrete sensing, i.e. sensing at (nondeterministic) discrete points in time.

Computational Thinking: This lecture uses the rigorous reasoning approach from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) and [Lecture 7 on Loops & Control](#) to study CPS models with time-triggered control. As a running example, the lecture continues to develop the extension from bouncing balls to ping pong balls, now using time-triggered control. We again add control decisions to the bouncing ball, turning it into a ping pong ball, which retains the intuitive simplicity of the bouncing ball, while enabling us to develop generalizable lessons about how to design time-triggered control systems correctly. The lecture will also crucially study invariants and show a development of the powerful technique of design-by-invariant in a concrete example. While the lecture could hardly claim showing how to verify CPS models of appropriate scale, the basics laid in this lecture definitely carry significance for numerous practical applications.

CPS Skills: This lecture develops an understanding for the semantics of time-triggered control. This understanding of the semantics will also guide our intuition of the operational effects of time-triggered control and especially the impact it has on finding correct control constraints. Finally, the lecture studies some aspects of higher-level model-predictive control, which will be followed up on later in the course.

2 Delays in Control

Event-driven control is a useful and intuitive model matching our expectation of having controllers react in response to certain critical conditions or events that necessitate

intervention by the controller. Yet, one of its difficulties is that event-driven control with its continuous sensing assumption can be hard or impossible to implement in reality. On a higher level of abstraction, it is very intuitive to design controllers that react to certain events and change the control actuation in response to what events have happened. Closer to the implementation, this turns out to be difficult, because actual computer control algorithms do not actually run all the time, only sporadically every once in a while, albeit sometimes very often. Implementing event-driven control faithfully would, in principle, require permanent continuous monitoring of the state to check whether an event has happened. That is not particularly realistic, because fresh sensor data will only be available every once in a while, and controller implementations will only run at certain discrete points in time causing delays in processing, and because actuators may sometimes take quite some time to get going. Think of the reaction time it takes you to turn the insight “I want to hit this ping pong ball there” into action so that your ping pong paddle will actually hit the ping pong ball.

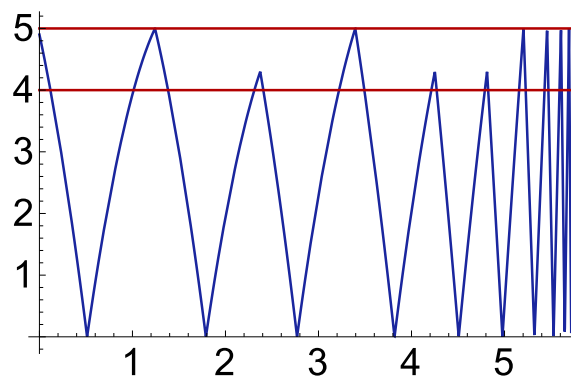


Figure 1: Sample trajectory of a ping pong ball (plotted as position over time) with the indicated ping pong paddle actuation range, sometimes actuating early, sometimes late

Back to the drawing desk. Let us reconsider the original $d\mathcal{L}$ formula (1) for the ping pong ball (Fig. 1) that we started out from for designing the event-driven version in [Lecture 8 on Events & Responses](#).

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[(x' = v, v' = -g \ \& \ x \geq 0; \right. \\
 &\quad \left. \text{if}(x = 0) \ v := -cv \ \text{else if}(4 \leq x \leq 5) \ v := -fv)^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{1}$$

This simplistic formula (1) turned out not to be valid, because its differential equation was not guaranteed to be interrupted when the event $4 \leq x \leq 5$ happens. Consequently, (1) needs some other evolution domain constraint to make sure all continuous evolutions are stopped at some point for the control to have a chance to react to situation changes. Yet, it should not be something like $\dots \ \& \ x \leq 5$ as in [Lecture 8 on Events &](#)

Responses, because continuously monitoring for $x \leq 5$ requires permanent continuous sensing of the height, which is difficult to implement.

Note 1 (Physical versus controller events). *Observe that the event $x = 0$ in the (physics) controller as well as the (physics) evolution domain constraint $x \geq 0$ for detecting the event $x = 0$ are perfectly justified in bouncing ball and ping pong ball models, because both represent physics. And physics is very well capable of keeping a ball above the ground, no matter how much checking for $x = 0$ it takes to make that happen. It is just in our controller code that we need to exercise care when modeling events and their reactions, because the controller implementations will not have the privilege that physics possesses of running all the time. Cyber happens every once in a while (even if may quickly), while physics happens all the time.*

How else could the continuous evolution of physics be interrupted to make sure the controller actually runs? By bounding the amount of time that physics is allowed to evolve before running the controller again. Before we can talk about time, the model needs to be changed to include a variable, say t , that reflects the progress of time with a differential equation $t' = 1$.

$$\begin{aligned} 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \left[(x' = v, v' = -g, t' = 1 \wedge x \geq 0 \wedge t \leq 1; \right. \\ \left. \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^* \right] (0 \leq x \leq 5) \end{aligned} \quad (2)$$

In order to bound time by 1, the evolution domain now includes $\dots \wedge t \leq 1$ and declares that the clock variable t evolves with time as $t' = 1$. Oops, that does not actually quite do it, because the HP in (2) restricts the evolution of the system so that it will never ever evolve beyond time 1, no matter how often the loop repeats. That is not what we meant to say. Rather we wanted the duration of each individual continuous evolution limited to at most one second. The trick is to reset the clock t to zero before the continuous evolution starts:

$$\begin{aligned} 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \left[(t := 0; (x' = v, v' = -g, t' = 1 \wedge x \geq 0 \wedge t \leq 1); \right. \\ \left. \text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv)^* \right] (0 \leq x \leq 5) \end{aligned} \quad (3)$$

In order to bound time by 1, the evolution domain now includes $\dots \wedge t \leq 1$ and the variable t is reset to 0 by $t := 0$ right before the differential equation. Hence, t represents a local clock measuring how long the evolution of the differential equation was. Its bound of 1 ensures that physics gives the controller a chance to react at least once per second. The system could very well stop the continuous evolution more often and earlier, because there is no lower bound on t in (3). Also see Exercise 1.

Before going any further, let's take a step back to notice an annoyance in the way the control in (3) was written. It is written in the style that the original bouncing ball and the event-driven ping pong ball were phrased: continuous dynamics followed by

control. That has the unfortunate effect that (3) lets physics happen before control does anything, which is not a very safe start. In other words, the initial condition would have to be modified to assume the initial control was fine. That is a nuisance duplicating part of the control into the assumptions on the initial state. Instead, let's switch the statements around to make sure control always happens before physics does anything.

$$\begin{aligned}
 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 \big[(\text{if}(x = 0) v := -cv \text{ else if}(4 \leq x \leq 5) v := -fv; \\
 t := 0; (x' = v, v' = -g, t' = 1 \ \& \ x \geq 0 \wedge t \leq 1))^* \big] (0 \leq x \leq 5)
 \end{aligned} \tag{4}$$

Now that $\text{d}\mathcal{L}$ formula (4) has an upper bound on the time it takes between two subsequent control actions, is it valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

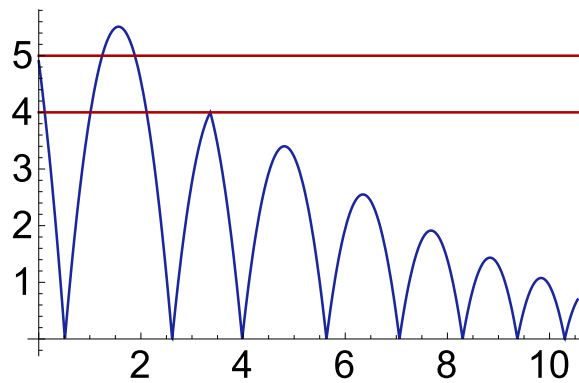


Figure 2: Sample trajectory of a time-triggered ping pong ball (as position over time), missing the first event

Even though (4) ensures a bound on how long it may take at most until the controller inspects the state and reacts, there is still a fundamental issue with (4). We can try to prove (4) and inspect the non-provable cases in the proof to find out what the issue is. The controller of (4) runs at least after one second (hence at least once per second) and then checks whether $4 \leq x \leq 5$. But if $4 \leq x \leq 5$ was not true when the controller ran last, there is no guarantee that it will be true when the controller runs next. In fact, the ball might very well have been at $x = 3$ at the last controller run, then evolved continuously to $x = 6$ within a second and so missed the event $4 \leq x \leq 5$ that it was supposed to detect (Exercise 2); see Fig. 2. Worse than that, the ping pong ball has then already become unsafe.

For illustration, driving a car would be similarly unsafe if you would only open your eyes once a second and monitor whether there is a car right in front of you. Too many things could have happened in between that should have prompted you to brake.

Note 2 (Delays may miss events). *Delays in controller reactions may cause events to be missed that they were supposed to monitor. When that happens, there is a discrepancy between an event-driven understanding of a CPS and the real time-triggered implementation. That happens especially for slow controllers monitoring small regions of a fast moving system. This relationship deserves special attention to make sure the impact of delays on a system controller cannot make it unsafe.*

It is often a good idea to first understand and verify an event-driven design of a CPS controller and then refine it to a time-triggered controller to analyze and verify that CPS in light of its reaction time. Discrepancies in this analysis hint at problems that event-driven designs will likely experience at runtime and they indicate a poor event abstraction.

How can this problem of (4) be solved? How can the CPS model make sure the controller does not miss its time to take action? Waiting until $4 \leq x \leq 5$ holds true is not guaranteed to be the right course of action for the controller.

Before you read on, see if you can find the answer for yourself.

The problem with (4) is that its controller is unaware of its own delay. It does not take into account how the ping pong ball could have moved further before it gets a chance to react next. If the ball is already close to the ping pong paddle's intended range of actuation, then the controller had better take action already if it is not sure whether next time will still be fine.

The controller would be in trouble if $x > 5$ might already hold in its next control cycle after the continuous evolution, which will be outside the operating range of the ping pong paddle (and already unsafe). Due to the evolution domain constraint, the continuous evolution can take at most 1 time unit, after which the ball will be at position $x + v - \frac{g}{2}$ as [Lecture 4](#) already showed by solving the differential equation. Choosing gravity $g = 1$ to simplify the math, the controller would be in trouble in the next control cycle after 1 second which would take the ball to position $x + v - \frac{1}{2} > 5$ if $x > 5\frac{1}{2} - v$ holds now.

The idea is to make the controller now act based on how it estimates the state might have evolved until the next control cycle (this is a very simple example of model-predictive control). [Lecture 8 on Events & Responses](#) already discovered for the event-driven case that the controller only wants to trigger action if the ball is flying up, not if it is already flying down. Thus, making (4) aware of the future in this way leads to:

$$\begin{aligned}
 &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\
 &\quad \left[\left(\text{if}(x = 0) v := -cv \text{ else if} \left(\left(x > 5\frac{1}{2} - v \right) \wedge v \geq 0 \right) v := -fv; \right. \right. \\
 &\quad \left. \left. t := 0; (x' = v, v' = -g, t' = 1 \ \& \ x \geq 0 \wedge t \leq 1) \right)^* \right] (0 \leq x \leq 5)
 \end{aligned} \tag{5}$$

Is conjecture (5) about its future-aware controller valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

The controller in formula (5) has been designed based on the prediction that the future may evolve for 1 time unit. If an action will no longer be possible in 1 time unit, because the event $x \leq 5$ has passed in that future time instant, then the controller in (5) takes action right now already. The issue with that, however, is that there is no guarantee at all that the ping pong ball will fly for exactly 1 time unit before the controller is asked to act again (and the postcondition is checked). The controller in (5) checks whether the ping pong ball could be too far up after one time unit and does not intervene unless that is the case. Yet, what if the ball only flies for $\frac{1}{2}$ time units? Clearly, if the ball will be safe after 1 time unit, which is what the controller in (5) checks, it will also be safe after just $\frac{1}{2}$ time unit, right?

Before you read on, see if you can find the answer for yourself.

Wrong! The ball may well be below 5 after 1 time unit but still could have been above 5 in between the current point of time and the time that is 1 time unit from now. Then the safety of the controller will be a mere rope of sand, because controller will have a false sense of safety after having checked what happens 1 time unit from now, ignoring whether it was safe until then. Such trajectories are shown in Fig. 3 from the same initial state and the same controller with different sampling periods. Such a bouncing ball would not be safe if it has been above 5 in between two sampling points.

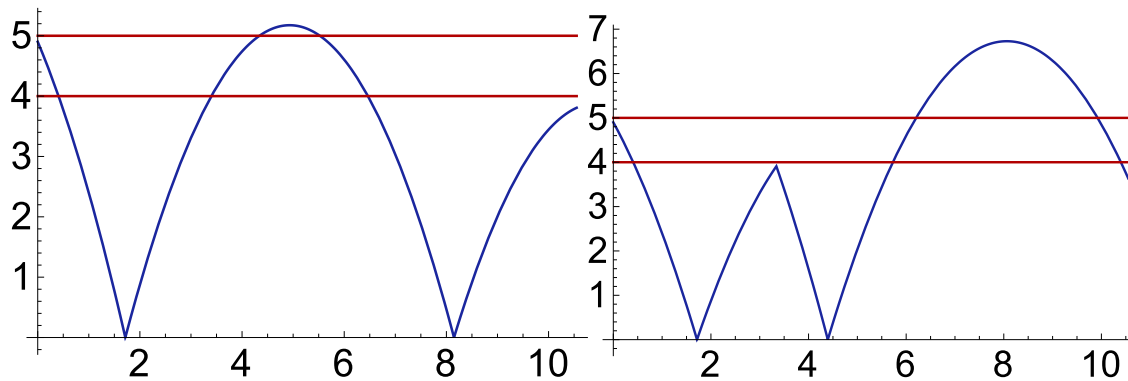


Figure 3: Sample trajectory of a time-triggered ping pong ball (as position over time), missing different events with different sampling periods

In order to get to the bottom of this, recall the invariant for the bouncing ball identified in [Lecture 4 on Safety & Contracts](#) and then used in [Lecture 7 on Loops & Invariants](#) to prove safety of the bouncing ball:

$$2gx = 2gH - v^2 \wedge x \geq 0 \wedge c = 1 \wedge g > 0 \quad (6)$$

This formula was proved to be an invariant of the bouncing ball, which means it holds true always while the bouncing ball is bouncing around. Invariants are the most crucial information about the behavior of a system that we can rely on all the time. Since (6) is only an invariant of the bouncing dynamics not the ping pong ball, it, of course, only holds until the ping pong paddle hits, which changes the control. But until the ping pong paddle is used, (6) summarizes concisely what we know about the state of the bouncing ball at all times. Of course, (6) is an invariant of the bouncing ball, but it still needs to be true initially. The easiest way to make that happen is to assume (6) in the beginning of the ping pong ball's life.¹ Because [Lecture 7](#) only conducted the proof of the bouncing ball invariant (6) for the case $c = 1$ to simplify the arithmetic, the ping pong ball now adopts this assumption as well. To simplify the arithmetic and arguments, also adopt the assumption $f = 1$ in addition to $c = 1 \wedge g = 1$ for the proofs.

¹Note that H is a variable that does not need to coincide with the upper height limit 5 like it did in the case of the bouncing ball, because the ping pong ball has more control at its fingertips. In fact, the most interesting case is if $H > 5$ in which case the ping pong ball will only stay safe because of its control. One way to think of H is as an indicator for the energy of the ball showing how high it might jump up if it would not be for all its interaction with the ground and the ping pong paddle.

Substituting the safety-critical height 5 for H in the invariant (6) for this instance of parameter choices leads to the following condition

$$2x > 2 \cdot 5 - v^2 \quad (7)$$

as an indicator for the fact that the ball might end up climbing too high, because its energy would allow it to. Adding this condition (7) to the controller (5) leads to:

$$\begin{aligned} \mathbf{2}x = \mathbf{2}H - v^2 \wedge 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 > 0 \wedge \mathbf{1} = c \geq 0 \wedge \mathbf{1} = f \geq 0 \rightarrow \\ \left[\left(\text{if}(x=0) v := -cv \text{ else if } (x > 5\frac{1}{2} - v \vee \mathbf{2}x > \mathbf{2} \cdot \mathbf{5} - v^2) \wedge v \geq 0 \right) v := -fv; \right. \quad (8) \\ \left. t := 0; (x' = v, v' = -g, t' = 1 \ \& \ x \geq 0 \wedge t \leq 1) \right)^* (0 \leq x \leq 5) \end{aligned}$$

Recall that the bouncing ball invariant (6) is now assumed to hold in the initial state.

Is dL formula (8) about its time-triggered controller valid? As usual, use an invariant or a counterexample for justification.

Before you read on, see if you can find the answer for yourself.

Formula (8) is “almost valid”. But it is still not valid for a very subtle reason. It is great to have proof to catch those subtle issues. The controller in (8) takes action for two different conditions on the height x . However, the ping pong paddle controller actually only runs in (8) if the ball is not at height $x = 0$, for otherwise ground control takes action of reversing the direction of the ball. Now, if the ball is flat on the floor already ($x = 0$) yet its velocity so incredibly high that it will rush past height 5 in less than 1 time unit, then the ping pong paddle controller will not even have had a chance to react before it is too late, because it does not execute on the ground according to (8); see Fig. 4.

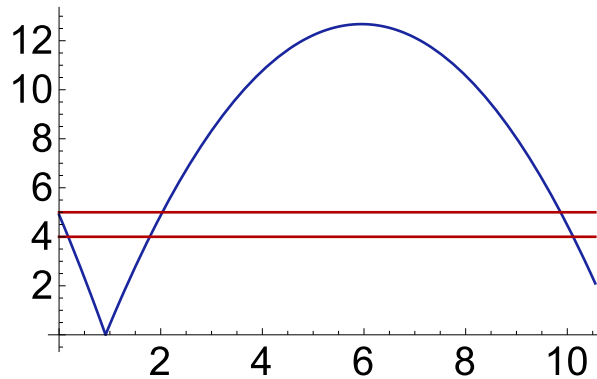


Figure 4: Sample trajectory of a time-triggered ping pong ball (as position over time), failing to control on the ground

Fortunately, these thoughts already indicate how that problem can be fixed. By turning the nested if-then-else cascade into a sequential composition of two separate if-then that will ensure the ping pong paddle controller to run for sure even if the bouncing ball is still on the ground (Exercise 3).

$$\begin{aligned}
 2x = 2H - v^2 \wedge 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 > 0 \wedge 1 = c \geq 0 \wedge 1 = f \geq 0 \rightarrow \\
 [(\text{if}(x = 0) v := -cv ; \text{if}((x > 5\frac{1}{2} - v \vee 2x > 2 \cdot 5 - v^2) \wedge v \geq 0) v := -fv; \quad (9) \\
 t := 0; (x' = v, v' = -g, t' = 1 \wedge x \geq 0 \wedge t \leq 1))^*](0 \leq x \leq 5)
 \end{aligned}$$

Now, is formula (9) finally valid, please? If so, using which invariant? Otherwise, show a counterexample.

Before you read on, see if you can find the answer for yourself.

Yes, formula (9) is valid. What invariant can be used to prove formula (9)?

Formula (9) is valid, which, for the case $g = c = f = 1$, can be proved with the following invariant:

$$2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5 \quad (10)$$

This invariant instantiates (6) for the present case of parameter choices and augments it with the desired safety constraint $x \leq 5$.

Yet, is the controller in (9) useful? That is where the problem lies now. The condition (7) that is the second disjunct in the controller of (9) checks whether the ping pong ball could possibly ever fly up to height 5. If this is ever true, it might very well be true long before the bouncing ball even approaches the critical control cycle where a ping pong paddle action needs to be taken. In fact, if (7) is ever true, it will also be true in the very beginning. After all, the formula (6), from which condition (7) derived, is an invariant, so always true for the bouncing ball. What would that mean?

That would cause the controller in (9) to take action right away at the mere prospects of the ball ever being able to climb way up high, even if the ping pong ball is still close to the ground and pretty far away from the last triggering height 5. That would make the ping pong ball safe, after all (9) is a valid formula. But it would also make it rather conservative and would not allow the ping pong ball to bounce around nearly as much as it would have loved to. It would basically make the bouncing ball lie flat on the ground, because of an overly anxious ping pong paddle. That would be a horrendously acrophobic bouncing ball if it never even started bouncing around in the first place. And the model would even require the (model) world to end, because there can be no progress beyond the point in time where the ball gets stuck on the ground. How can the controller in (9) be modified to resolve this problem?

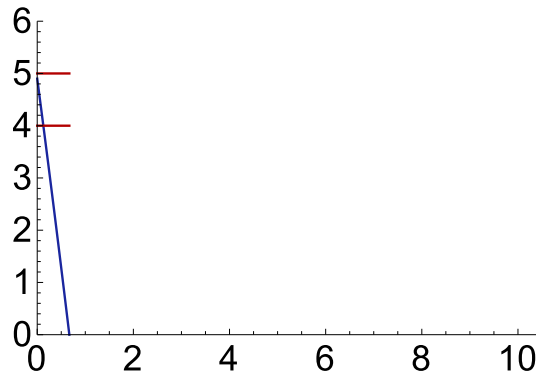
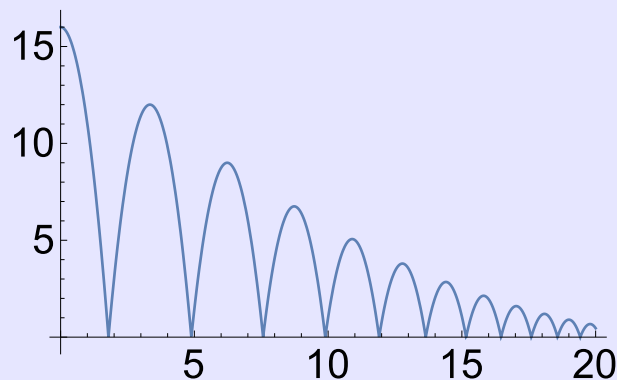


Figure 5: Sample trajectory of a time-triggered ping pong ball (as position over time), stuck on the ground

Before you read on, see if you can find the answer for yourself.

Note 3 (Zeno paradox). *There is something quite surprising about how (9) may cause the time to freeze. But, come to think of it, time did already freeze in mere bouncing balls.*



The duration between two hops on the ground in a bouncing ball keeps on decreasing rapidly. If, for simplicity, the respective durations are $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$, then these durations sum to

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{1 - \frac{1}{2}} = 2$$

which shows that the bouncing ball model will make the (model) world stop to never reach time 2 nor any time after. Hence, the bouncing ball model disobeys what is called divergence of time, i.e. that the real time keeps diverges to ∞ . The reason this happens is that the bouncing ball keeps on switching directions on the ground more and more frequently. This is very natural for bouncing balls, but can cause subtleties and issues in other systems if they switch infinitely often in finite time.

The name Zeno paradox comes from the Greek philosopher Zeno (ca. 490–430 BC) who found a paradox when fast runner Achilles gives the slow Tortoise a head start of 100 meters in a race: In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead. – recounted by Aristotle, Physics VI:9, 239b15

Pragmatic solutions for the Zeno paradox in bouncing balls add a statement that make the ball stop when the remaining velocity on the ground is too small. For example:

$$\text{if}(x = 0 \wedge -0.1 < v < 0.1) v := 0; x' = 0$$

The idea is to restrict the use of the second if-then disjunct (7) in (9) to slow velocities in order to make sure it only applies to the occasions that the first controller disjunct $x > 5\frac{1}{2} - v$ misses, because the ball will have been above height 5 in between. Only with slow velocities will the ball ever move so slowly that it is near its turning point to begin its descent and start falling down again before 1 time unit. And only then could the first condition miss out on the ball being able to evolve above 5 before 1 time unit. When is a velocity slow in this respect?

For the ball to turn around and descend, it first needs to reach velocity $v = 0$ by continuity (during the flying phase) on account of the mean-value theorem. In gravity $g = 1$ the ball can reach velocity 0 within 1 time unit exactly when its velocity was $v < 1$ before the differential equation, because the velocity changes according to $v(t) = v - gt$. Consequently, adding a conjunct $v < 1$ to the second disjunct in the controller makes sure that the controller only checks for turnaround when it might actually happen during the next control cycle.

$$\begin{aligned}
 2x = 2H - v^2 \wedge 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 &> 0 \wedge 1 = c \geq 0 \wedge 1 = f \geq 0 \rightarrow \\
 &[(\text{if}(x = 0) v := -cv; \text{if}((x > 5\frac{1}{2} - v \vee 2x > 2 \cdot 5 - v^2 \wedge v < 1) \wedge v \geq 0) v := -fv; \\
 &t := 0; (x' = v, v' = -g, t' = 1 \& x \geq 0 \wedge t \leq 1))^*](0 \leq x \leq 5)
 \end{aligned} \tag{11}$$

This $\text{d}\mathcal{L}$ formula is valid and provable with the same invariant (10) that was already used to prove (9). It has a much more aggressive controller than (9), though, so it is more fun for the ping pong ball to bounce around with it.

The easiest way of proving that $\text{d}\mathcal{L}$ formula (11) is valid using invariant (10) is to show that the invariant (10) holds after every line of code. Formally, this reasoning by lines corresponds to a number of uses of the generalization proof rule $\llbracket \text{gen}' \rrbracket$ to show that the invariant (11) remains true after each line if it was true before. The first statement $\text{if}(x = 0) v := -cv$ does not change the truth-value of (10), i.e.

$$2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5 \rightarrow [\text{if}(x = 0) v := -cv](2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5)$$

is valid, because, when $c = 1$, the statement can only change the sign of v and (10) is independent of signs, because the only occurrence of v satisfies $(-v)^2 = v^2$. Likewise, the second statement $\text{if}((x > 5\frac{1}{2} - v \vee 2x > 2 \cdot 5 - v^2 \wedge v < 1) \wedge v \geq 0) v := -fv$ does not change the truth-value of (10), i.e.

$$\begin{aligned}
 2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5 \rightarrow \\
 [\text{if}((x > 5\frac{1}{2} - v \vee 2x > 2 \cdot 5 - v^2 \wedge v < 1) \wedge v \geq 0) v := -fv](2x = 2H - v^2 \wedge x \geq 0 \wedge x \leq 5)
 \end{aligned}$$

is valid, because, for $f = 1$, the second statement can also only change the sign of v , which is irrelevant for the truth-value of (10). Finally, the relevant parts of (10) are a special case of (6), which has already been shown to be an invariant for the bouncing ball differential equation in [Lecture 7 on Loops & Invariants](#) and, thus, continues to be an invariant when adding a clock $t' = 1 \& t \leq 1$, which does not occur in (10). The additional invariant $x \leq 5$ that (10) has compared to (6) is easily taken care off using the corresponding knowledge about H .

Recall that (global) invariants need to be augmented with the usual trivial assumptions about the unchanged variables: $g = 1 \wedge 1 = c \wedge 1 = f$.

See [«Time-triggered ping pong ball KeYmaera model»](#)

Note 4 (Time-triggered control). *One common paradigm for designing controllers is time-triggered control, in which controllers run periodically or pseudo-periodically with certain frequencies to inspect the state of the system. Time-triggered systems are closer to implementation than event-driven control. They can be harder to build, however, because they invariably require the designer to understand the impact of delay on control decisions. That impact is important in reality, however, and, thus, effort invested in understanding the impact of time delays usually pays off in designing a safer system that is robust to bounded time delays.*

Partitioning the hybrid program in (11) into the parts that come from physics (typographically marked like **physics**) and the parts that come from control (typographically marked like **control**) leads to:

$$\begin{aligned}
 2x = 2H - v^2 \wedge 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g = 1 > 0 \wedge 1 = c \geq 0 \wedge 1 = f \geq 0 \rightarrow \\
 [(\text{if}(x = 0) v := -cv; \text{if}((x > 5\frac{1}{2} - v \vee 2x > 2 \cdot 5 - v^2 \wedge v < 1) \wedge v \geq 0) v := -fv; \\
 t := 0; (x' = v, v' = -g, t' = 1 \wedge x \geq 0 \wedge t \leq 1))^*](0 \leq x \leq 5)
 \end{aligned} \tag{11}$$

Note how part of the differential equation, namely $t' = 1$, comes from the controller, because it corresponds to putting a clock on the controller and running it with at least the sampling frequency 1 (coming from the evolution domain constraint $t \leq 1$).

3 Summary

This lecture studied time-triggered control, which, together with event-driven control from [Lecture 8 on Events & Responses](#), is an important principle for designing feedback mechanisms in CPS and embedded systems. The lecture illustrated the most important aspects for a running example of a ping pong ball. Despite or maybe even because of its simplicity, the ping pong ball was an instructive source for the most important subtleties involved with time-triggered control decisions. Getting time-triggered controllers correct requires predictions about how the system state might evolve over short periods of time (one control cycle). The effects and subtleties of time-triggered actions in control were sufficiently subtle to merit focusing on a simple intuitive case.

Unlike event-driven control, which assumes continuous sensing, time-triggered control is more realistic by only assuming the availability and processing of sensor data at discrete instants of time (discrete sensing). Time-triggered system models avoid the modeling subtleties that events tend to cause for the detection of events. It is, thus, often much easier to get the models right for time-triggered systems than it is for event-driven control. The price is that the burden of event-detection is then brought to the attention of the CPS programmer, whose time-triggered controller will now have to ensure it predicts and detects events early enough before it is too late to react to them. That is what makes time-triggered controllers more difficult to get correct, but is also

crucial because important aspects of reliable event detection may otherwise be brushed under the rug, which does not exactly help the final CPS become any safer either.

CPS design often begin by pretending the idealized world of event-driven control (if the controller is not even safe when events are checked and responded to continuously, it is broken already) and then subsequently morphing the event-driven controller into a time-triggered controller. This second step then often indicates additional subtleties that were missed in the event-driven designs. The additional insights gained in time-triggered controllers are crucial whenever the system reacts slowly or whenever it reacts fast but needs a high precision to remain safe. For example, the reaction time for ground control decisions to reach a rover on Mars are so prohibitively large that they could hardly be ignored. Reaction times in a surgical robotics system that is running at, say, 55Hz, are still crucial even if the system is moving slow and reacting fast, because the required precision of the system is in the sub-millimeter range [KRPK13].

Overall, the biggest issues with event-driven control, besides sometimes being hard to implement, is the subtleties involved in properly modeling event detection without accidentally defying the laws of physics in pursuit of an event. But controlling event-driven systems is reasonably straight-forward as long as the events are chosen well. Finding a model is comparably canonical in time-triggered control, but identifying the controller constraints takes a lot more thought, leading, however, to important insights about the system at hand.

Exercises

Exercise 1. The HP in (4) imposes an upper bound on the duration of a continuous evolution. How can you impose an upper bound 1 and a lower bound 0.5?

Exercise 2. Give an initial state for which the controller in (4) would skip over the event without noticing it.

Exercise 3. What would happen if the controller in (9) uses the ping pong paddle while the ball is still on the ground? To what physical phenomenon does that correspond?

Exercise 4. The formula (11) with the time-triggered controller of reaction time at most 1 time unit is valid. Yet, if a ball is let loose a wee bit above ground with a very fast negative velocity, couldn't it possibly bounce back and exceed the safe height 5 faster than the reaction time of 1 time unit? Does that mean the formula ought to have been falsifiable? No! Identify why and give a physical interpretation.

Exercise 5. The controller in (11) ran at least once a second. How can you change the model and controller so that it runs at least twice a second? What changes can you do in the controller to reflect that increased frequency? How do you need to change (11) if the controller only runs at least once every two seconds?

Exercise 6. The event-driven controller we designed in [Lecture 8 on Events & Responses](#) monitored the event $4 \leq x \leq 5$. The time-triggered controller in Sect. 2, however, ultimately only took the upper bound 5 into account. How and under which circumstances

can you modify the controller so that it really only reacts for the event $4 \leq x \leq 5$ rather than under all circumstances where the ball is in danger of exceeding 5?

Exercise 7. Devise a controller that reacts if the height changes by 1 when comparing the height before the continuous evolution to the height after. Can you make it safe? Can you implement it? Is it an event-driven or a time-triggered controller? How does it compare to the controllers developed in this lecture?

Exercise 8. The ping pong ball proof relied on the parameter assumptions $g = c = f = 1$ for mere convenience of the resulting arithmetic. Develop a time-triggered model, controller, and proof for the general ping pong ball.

Exercise 9 ().* Design a variation of the time-triggered controller for the ping pong ball that is allowed to use the ping pong paddle within height $4 \leq x \leq 5$ but has a relaxed safety condition that accepts $0 \leq x \leq 2 \cdot 5$. Make sure to only force the use of the ping pong paddle when necessary. Find an invariant and conduct a proof.

References

- [KRPK13] Yanni Kouskoulas, David W. Renshaw, André Platzer, and Peter Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In Calin Belta and Franjo Ivancic, editors, *HSCC*, pages 263–272. ACM, 2013. doi:[10.1145/2461328.2461369](https://doi.org/10.1145/2461328.2461369).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. [arXiv:1205.4788](https://arxiv.org/abs/1205.4788).

Lecture Notes on Differential Equations & Differential Invariants

André Platzer

Carnegie Mellon University
Lecture 10

1 Introduction

So far, this course explored only one way to deal with differential equations: the [\['\]](#) axiom from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#). However, in order to use the [\['\]](#) axiom or its sequent calculus counterpart the [\['\]r](#) rule from [Lecture 6 on Truth & Proof](#) for a differential equation $x' = \theta$, we must be able to find a symbolic solution to the symbolic initial value problem (i.e. a function $y(t)$ such that $y'(t) = \theta$ and $y(0) = x$). But what if the differential equation does not have such a solution $y(t)$? Or if $y(t)$ cannot be written down in first-order real arithmetic? [Lecture 2 on Differential Equations & Domains](#) allows many more differential equations to be part of CPS models than just the ones that happen to have simple solutions. These are the differential equations we will look at in this lecture.

You may have seen a whole range of methods for solving differential equations in prior courses. But, in a certain sense, “most” differential equations are impossible to solve, because they have no explicit closed-form solution with elementary functions, for instance [\[Zei03\]](#):

$$x''(t) = e^{t^2}$$

And even if they do have solutions, the solution may no longer be in first-order real arithmetic. A solution of

$$d' = e, e' = -d$$

for example is $d(t) = \sin t, e(t) = \cos t$, which is not expressible in real arithmetic (recall that both are infinite power series) and leads to undecidable arithmetic [\[Pla08a\]](#).

Today's lecture reinvestigates differential equations from a more fundamental perspective, which will lead to a way of proving properties of differential equations without using their solutions.

The lecture seeks unexpected analogies among the seemingly significantly different dynamical aspects of discrete dynamics and of continuous dynamics. The first and influential observation is that differential equations and loops have more in common than one might suspect.¹ Discrete systems may be complicated, but have a powerful ally: induction as a way of establishing truth for discrete dynamical systems by generically analyzing the one step that it performs (repeatedly like the body of a loop). What if we could use induction for differential equations? What if we could prove properties of differential equations directly by analyzing how these properties change along the differential equation rather than having to find a global solution first and inspecting whether it satisfies that property? What if we could tame the analytic complexity of differential equations by analyzing the generic local "step" that a continuous dynamical system performs (repeatedly). The biggest conceptual challenge will, of course, be in understanding what exactly the counterpart of a step even is for continuous dynamical systems, because there is no such thing as a next step for a differential equation.

More details can be found in [Pla10b, Chapter 3.5] and [Pla10a, Pla12d, Pla12a, Pla12b]. Differential invariants were originally conceived in 2008 [Pla10a, Pla08b] and later used for an automatic proof procedure for hybrid systems [PC08, PC09].

This lecture is of central significance for the Foundations of Cyber-Physical Systems. The analytic principles begun in this lecture will be a crucial basis for analyzing all complex CPS. The most important learning goals of this lecture are:

Modeling and Control: This lecture will advance the core principles behind CPS by developing a deeper understanding of their continuous dynamical behavior. This lecture will also illuminate another facet of how discrete and continuous systems relate to one another, which will ultimately lead to a fascinating view on understanding hybridness [Pla12a].

Computational Thinking: This lecture exploits computational thinking in its purest form by seeking and exploiting surprising analogies among discrete dynamics and continuous dynamics, however different both may appear at first sight. This lecture is devoted to rigorous reasoning about the differential equations in CPS models. Such rigorous reasoning is crucial for understanding the continuous behavior that CPS exhibit over time. Without sufficient rigor in their analysis it can be impossible to understand their intricate behavior and spot subtle flaws in their control or say for sure whether and why a design is no longer faulty. This lecture systematically develops one reasoning principle for equational properties of differential equations that is based on *induction for differential equations*. Subsequent lectures expand the same core principles developed in this lecture to the study of general properties of differential equations. This lecture continues the *axiomatiza-*

¹ In fact, discrete and continuous dynamics turn out to be proof-theoretically quite intimately related [Pla12a].

tion of differential dynamic logic $d\mathcal{L}$ [Pla12c, Pla12a] pursued since [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) and lifts $d\mathcal{L}$'s proof techniques to systems with more complex differential equations. The concepts developed in this lecture form the differential facet illustrating the more general relation of *syntax* (which is notation), *semantics* (what carries meaning), and *axiomatics* (which internalizes semantic relations into universal syntactic transformations). These concepts and their relations jointly form the significant *logical trinity* of syntax, semantics, and axiomatics. Finally, the verification techniques developed in this lecture are critical for verifying CPS models of appropriate scale and technical complexity.

CPS Skills: We will develop a deeper understanding of the semantics of the continuous dynamical aspects of CPS models and develop and exploit a significantly better intuition for the operational effects involved in CPS.

2 Global Descriptive Power of Local Differential Equations

Differential equations let the physics evolve continuously for longer periods of time. They describe such global behavior locally, however, just by the right-hand side of the differential equation.

Note 1 (Local descriptions of global behavior by differential equations). *The key principle behind the descriptive power of differential equations is that they describe the evolution of a continuous system over time using only a local description of the direction into which the system evolves at any point in space. The solution of a differential equation is a global description of how the system evolves, while the differential equation itself is a local characterization. While the global behavior of a continuous system can be subtle and challenging, its local description as a differential equation is much simpler.*

This difference between local description and global behavior can be exploited for proofs.

The semantics of a differential equation was described in [Lecture 2 on Differential Equations & Domains](#) as:

$$\rho(x' = \theta \ \& \ H) = \{(\varphi(0), \varphi(r)) \ : \ \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \\ \text{for a solution } \varphi : [0, r] \rightarrow \mathcal{S} \text{ of any duration } r\}$$

The solution φ describes the global behavior of the system, which is specified locally by the right-hand side θ of the differential equation.

[Lecture 2](#) has shown a number of examples illustrating the descriptive power of differential equations. That is, examples in which the solution was very complicated even though the differential equation was rather simple. This is a strong property of differential equations: they can describe even complicated processes in simple ways. Yet, that representational advantage of differential equations does not carry over into the verification when verification is stuck with proving properties of differential equations

only by way of their solutions, which, by the very nature of differential equations, are more complicated again.

This lecture, thus, investigates ways of proving properties of differential equations using the differential equations themselves, not their solutions. This leads to *differential invariants* [Pla10a, Pla12d], which can perform induction for differential equations.

3 Differential Equations vs. Loops

A programmatic way of developing an intuition for differential invariants leads through a comparison of differential equations with loops. This perhaps surprising relation can be made completely rigorous and is at the heart of a deep connection equating discrete and continuous dynamics proof-theoretically [Pla12a]. This lecture will stay at the surface of this surprising connection but still leverage the relation of differential equations to loops for our intuition.

To get started with relating differential equations to loops, compare

$$x' = \theta \quad \text{vs.} \quad (x' = \theta)^*$$

How does the differential equation $x' = \theta$ compare to the same differential equation in a loop $(x' = \theta)^*$ instead? Unlike the differential equation $x' = \theta$, the repeated differential equation $(x' = \theta)^*$ can run the differential equation $x' = \theta$ repeatedly. Albeit, on second thought, does that get the repetitive differential equation $(x' = \theta)^*$ to any more states than where the differential equation $x' = \theta$ could evolve to?

Not really, because chaining lots of solutions of differential equations from a repetitive differential equation $(x' = \theta)^*$ together will still result in a single solution for the same differential equation $x' = \theta$ that we could have followed all the way.²

Note 2 (Looping differential equations). $(x' = \theta)^*$ is equivalent to $x' = \theta$, written $(x' = \theta)^* \equiv (x' = \theta)$, i.e. both have the same transition semantics:

$$\rho((x' = \theta)^*) = \rho(x' = \theta)$$

Differential equations “are their own loop”.³

In light of Note 2, differential equations already have some aspects in common with loops. Like nondeterministic repetitions, differential equations might stop right away. Like nondeterministic repetitions, differential equations could evolve for longer or shorter durations and the choice of duration is nondeterministic. Like in nondeterministic repetitions, the outcome of the evolution of the system up to an intermediate state influences what happens in the future. And, in fact, in a deeper sense, differential equations

²This is related to classical results about the continuation of solutions, e.g., [Pla10b, Proposition B.1].

³Beware not to confuse this with the case for differential equations with evolution domain constraints, which is subtly different (Exercise 1).

actually really do correspond to loops executing their discrete Euler approximations [Pla12a].

With this rough relation in mind, let's advance the dictionary translating differential equation phenomena into loop phenomena and back. The local description of a differential equation as a relation $x' = \theta$ of the state to its derivative corresponds to the local description of a loop by a repetition operator $*$ applied to the loop body α . The global behavior of a global solution of a differential equation $x' = \theta$ corresponds to the full global execution trace of a repetition α^* , but are similarly unwieldy objects to handle. Because the local descriptions are so much more concise than the respective global behaviors, but still carry all information about how the system will evolve over time, we also say that the local relation $x' = \theta$ is the *generator* of the global system solution and that the loop body α is the *generator* of the global behavior of repetition of the loop. Proving a property of a differential equation in terms of its solution corresponds to proving a property of a loop by unwinding it (infinitely long) by axiom $[*n]$ from Lecture 5 on Dynamical Systems & Dynamic Axioms.

Note 3 (Correspondence map between loops and differential equations).

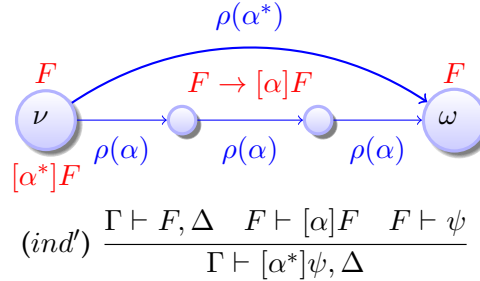
<i>loop</i> α^*	<i>differential equation</i> $x' = \theta$
could repeat 0 times	could evolve for duration 0
repeat any number $n \in \mathbb{N}$ of times	evolve for any duration $0 \leq r \in \mathbb{R}$
effect depends on previous loop iteration	effect depends on the past solution
local generator α	local generator $x' = \theta$
full global execution trace	global solution $\varphi : [0, r] \rightarrow \mathcal{S}$
proof by unwinding iterations with $[*n]$	proof by global solution with axiom $[']$
proof by induction with loop invariant rule <i>ind'</i>	proofs by differential invariants

Now, Lecture 7 on Control Loops & Invariants made the case that unwinding the iterations of a loop can be a rather tedious way of proving properties about the loop, because there is no good way of ever stopping to unwind, unless a counterexample can be found after a finite number of unwindings. This is where working with a global solution of a differential equation with axiom $[']$ is actually already more useful, because the solution can actually be handled completely because of the quantifier $\forall t \geq 0$ over all durations. But Lecture 7 introduced induction with invariants as the preferred way of proving properties of loops, by, essentially, cutting the loop open and arguing that the generic state after any run of the loop body has the same characterization as the generic state before. After all these analogous correspondences between loops and differential equations, the obvious question is what the differential equation analogue of a proof concept would be that corresponds to proofs by induction for loops, which is the premier technique for proving loops.

Induction can be defined for differential equations using what is called *differential invariants* [Pla10a, Pla12d]. They have a similar principle as the proof rules for induction for loops. Differential invariants prove properties of the solution of the differential equation using only its local generator: the right-hand side of the differential equation.

Recall the loop induction proof rule from [Lecture 7 on Loops & Invariants](#):

$$\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \quad \text{with} \quad \alpha^{n+1} \equiv \alpha^n; \alpha \text{ and } \alpha^0 \equiv ?true$$



4 Intuition of Differential Invariants

Just as inductive invariants are the premier technique for proving properties of loops, differential invariants [\[Pla10a, Pla12d, Pla08b, Pla10b\]](#) provide the primary inductive technique we use for proving properties of differential equations (without having to solve them).

The core principle behind loop induction is that the induction step investigates the local generator α and shows that it never changes the truth-value of the invariant F (see the middle premise $F \vdash [\alpha]F$ of proof rule *ind'* or the only premise of the core induction proof rule *ind* from [Lecture 7](#)). Let us try to establish the same inductive principle, just for differential equations. The first and third premise of rule *ind'* transfer easily to differential equations. The challenge is to figure out what the counterpart of $F \vdash [\alpha]F$ would be since differential equations do not have a notion of “one step”.

What does the local generator of a differential equation $x' = \theta$ tell us about the evolution of a system? And how does it relate to the truth of a formula F all along the solution of that differential equation? That is, to the truth of the dL formula $[x' = \theta]F$ expressing that all runs of $x' = \theta$ lead to states satisfying F . Fig. 1 depicts an example of a vector field for a differential equation (plotting the right-hand side of the differential equation as a vector at every point in the state space), a global solution (in red), and an unsafe region $\neg F$ (shown in blue). The safe region F is the complement of the blue unsafe region $\neg F$.

One way of proving that $[x' = \theta]F$ is true in a state ν would be to compute a solution from that state ν , check every point in time along the solution to see if it is in the safe region F or the unsafe region $\neg F$. Unfortunately, these are uncountably infinitely many points in time to check. Furthermore, that only considers a single initial state ν , so proving validity of a formula would require considering every of the uncountably infinitely many possible initial states and computing and following a solution in each of them. That is why this naïve approach would not compute.

A similar idea can still be made to work when the symbolic initial-value problem can be solved with a symbolic initial value x and a quantifier for time can be used, which

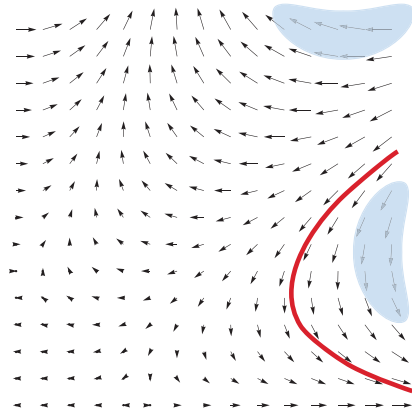


Figure 1: Vector field and one solution of a differential equation that does not enter the blue unsafe regions

is what the solution axiom [7] does. Yet, even that only works when a solution to the symbolic initial-value problem can be computed and the arithmetic resulting from the quantifier for time can be decided. For polynomial solutions, this works, for example. But polynomial come from very simple systems only (called nilpotent linear differential equation systems).

Reexamining the illustration in Fig. 1, we suggest an entirely different way of checking whether the system could ever lead to an unsafe state in $\neg F$ when following the differential equation $x' = \theta$. The intuition is the following. If there were a vector in Fig. 1 that points from a safe state in F to an unsafe state $\neg F$ (in the blue region), then following that vector could get the system into an unsafe $\neg F$. If, instead, all vectors point from safe states to safe states in F , then, intuitively, following such a chain of vectors will only lead from safe states to safe states. So if the system also started in a safe state, it would stay safe forever.

Let us make this intuition rigorous to obtain a sound proof principle that is perfectly reliable in order to be usable in CPS verification. What we need to do is to find a way of characterizing how the truth of F changes when moving along the differential equation.

5 Deriving Differential Invariants

How can the intuition about directions of evolution of a logical formula F with respect to a differential equation $x' = \theta$ be made rigorous? We develop this step by step.

As a guiding example, consider a conjecture about the rotational dynamics where d and e represent the direction of a vector rotating clockwise in a circle of radius r (Fig. 2):

$$d^2 + e^2 = r^2 \rightarrow [d' = e, e' = -d] d^2 + e^2 = r^2 \quad (1)$$

The conjectured $d\mathcal{L}$ formula (1) is valid, because, indeed, if the vector (d, e) is initially at distance r from the origin $(0,0)$, then it will always be when rotating around the ori-

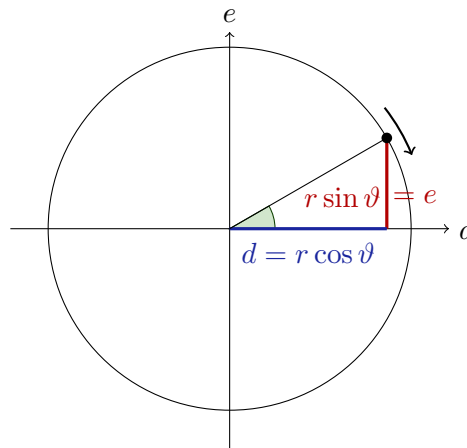


Figure 2: One scenario for the rotational dynamics and relationship of vector (d, e) to radius r and angle ϑ

gin, which is what the dynamics does. That is, the point (d, e) will always remain on the circle of radius r . But how can we prove that? In this particular case, we could possibly investigate solutions, which are trigonometric functions (although the ones shown in Fig. 2 are not at all the only solutions). With those solutions, we could perhaps find an argument why they stay at distance r from the origin. But the resulting arithmetic will be unnecessarily difficult and, after all, the argument for why the simple $d\mathcal{L}$ formula (1) is valid should be easy. And it is, after we have discovered the right proof principle as this lecture will do.

First, what is the direction into which a continuous dynamical system evolves? The direction is exactly described by the differential equation, because the whole point of a differential equation is to describe in which direction the state evolves at every point in space. So the direction into which a continuous system obeying $x' = \theta$ follows from state ν is exactly described by the time-derivative, which is exactly the value $\llbracket \theta \rrbracket_\nu$ of term θ in state ν . Recall that the term θ can mention x and other variables so its value $\llbracket \theta \rrbracket_\nu$ depends on the state ν .

Note 4 (Differential invariants are “formulas that remain true in the direction of the dynamics”). *Proving $d\mathcal{L}$ formula $[x' = \theta]F$ does not really require us to answer where exactly the system evolves to but just how the evolution of the system relates to the formula F and the set of states ν in which F evaluates to true. It is enough to show that the system only evolves into directions in which formula F will stay true.*

A logical formula F is ultimately built from atomic formulas that are comparisons of (polynomial or rational) terms such as, say, $\eta = 5$. Let η denote such a (polynomial) term in the variable (vector) x that occurs in the formula F . The semantics of a polynomial term η in a state ν is the real number $\llbracket \eta \rrbracket_\nu$ that it evaluates to. In which direction does the value of η evolve when following the differential equation $x' = \theta$ for some

time? That depends both on the term η that is being evaluated and on the differential equation $x' = \theta$ that describes how the respective variables x evolve over time.

Note 5. *Directions of evolutions are described by derivatives, after all the differential equation $x' = \theta$ describes that the time-derivative of x is θ .*

Let's derive the term η of interest and see what that tells us about how η evolves over time. How can we derive η ? The term η could be built from any of the operators discussed in [Lecture 2 on Differential Equations & Domains](#), to which we now add division for rational terms to make it more interesting. Let Σ denote the set of all variables. Terms θ are defined by the grammar (where θ, η are terms, x a variable, and r a rational number constant):

$$\theta, \eta ::= x \mid r \mid \theta + \eta \mid \theta - \eta \mid \theta \cdot \eta \mid \theta / \eta$$

It is, of course, important to take care that division θ/η only makes sense in a context where the divisor η is guaranteed not to be zero in order to avoid undefinedness. We only allow division to be used in a context where the divisor is ensured not to be zero.

If η is a sum $a + b$, its derivative is the derivative of a plus the derivative of b . If η is a product $a \cdot b$, its derivative is the derivative of a times b plus a times the derivative of b by Leibniz' rule. The derivative of a rational number constant $r \in \mathbb{Q}$ is zero.⁴ The other operators are similar, leaving only the case of a single variable x . What is its derivative?

Before you read on, see if you can find the answer for yourself.

⁴Of course, the derivative of real number constants $r \in \mathbb{R}$ is also zero, but only rational number constants are allowed to occur in the formulas of first-order logic of real arithmetic, more precisely, of real-closed fields.

The exact value of the derivative of a variable x certainly depends on the current state and on the overall continuous evolution of the system. So for now, we define the derivative of a variable x in a seemingly innocuous way to be the symbol x' and consider what to do with it later. This gives rise to the following definition for computing the derivative of a term syntactically.

Definition 1 (Derivation). The operator $(\cdot)'$ that is defined as follows on terms is called *syntactic (total) derivation*:

$$(r)' = 0 \quad \text{for numbers } r \in \mathbb{Q} \quad (2a)$$

$$(x)' = x' \quad \text{for variable } x \in \Sigma \quad (2b)$$

$$(a + b)' = (a)' + (b)' \quad (2c)$$

$$(a - b)' = (a)' - (b)' \quad (2d)$$

$$(a \cdot b)' = (a)' \cdot b + a \cdot (b)' \quad (2e)$$

$$(a/b)' = ((a)' \cdot b - a \cdot (b)')/b^2 \quad (2f)$$

Note that the intuition (and precise semantics) of derivatives of terms will ultimately be connected with more complicated aspects of how values change over time, the computation of derivatives of terms is a straightforward recursive definition on terms.

Expedition 1 (Differential algebra). Even though the following names are not needed for his course, let's take a brief expedition to align Def. 1 with the algebraic structures from differential algebra [Kol72] in order to illustrate the systematic principles behind Def. 1. Case (2a) defines (rational) number symbols alias literals as *differential constants*, which do not change their value during continuous evolution. Their derivative is zero. The number symbol 5 will always have the value 5 and never change, no matter what differential equation is considered. Equation (2c) and the *Leibniz* or *product rule* (2e) are the defining conditions for *derivation operators on rings*. The derivative of a sum is the sum of the derivatives (additivity or a homomorphic property with respect to addition, i.e. the operator $(\cdot)'$ applied to a sum equals the sum of the operator applied to each summand) according to equation (2c). Furthermore, the derivative of a product is the derivative of one factor times the other factor plus the one factor times the derivative of the other factor as in (2e). Equation (2d) is a derived rule for subtraction according to the identity $a - b = a + (-1) \cdot b$ and again expresses a homomorphic property, now with respect to subtraction rather than addition.

The equation (2b) uniquely defines the operator $(\cdot)'$ on the *differential polynomial algebra* spanned by the *differential indeterminates* $x \in \Sigma$, i.e. the symbols x that have indeterminate derivatives x' . It says that we understand the differential symbol x' as the derivative of the symbol x for all state variables $x \in \Sigma$. Equation (2f) canonically extends the derivation operator $(\cdot)'$ to the *differential field of quotients* by

the usual *quotient rule*. As the base field \mathbb{R} has no zero divisors^a, the right-hand side of (2f) is defined whenever the original division a/b can be carried out, which, as we assumed for well-definedness, is guarded by $b \neq 0$.

^aIn this setting, \mathbb{R} has no zero divisors, because the formula $ab = 0 \rightarrow a = 0 \vee b = 0$ is valid, i.e. a product is zero only if a factor is zero.

The derivative of a division a/b uses a division, which is where we need to make sure not to accidentally divide by zero. Yet, in the definition of $(a/b)'$, the division is by b^2 which has the same roots that b already has, because $b = 0 \leftrightarrow b^2 = 0$ is valid for any term b . Hence, in any context in which a/b was defined, its derivative $(a/b)'$ will also be defined.

Now that we have a precise definition of derivation at hand, the question still is which of the terms should be derived when trying to prove (1)? Since that is not necessarily clear so far, let's turn the formula (1) around and consider the following equivalent (Exercise 2) $d\mathcal{L}$ formula instead, which only has a single nontrivial term to worry about:

$$d^2 + e^2 - r^2 = 0 \rightarrow [d' = e, e' = -d]d^2 + e^2 - r^2 = 0 \quad (3)$$

Derivation of the only relevant term $d^2 + e^2 - r^2$ in the postcondition of (3) gives

$$(d^2 + e^2 - r^2)' = 2dd' + 2ee' - 2rr' \quad (4)$$

Def. 1 makes it possible to derive any polynomial or rational term. Deriving them with the total derivative operator $(\cdot)'$ does *not* result in a term over the signature of the original variables in Σ , but, instead, a *differential term*, i.e. a term over the extended signature $\Sigma \cup \Sigma'$, where $\Sigma' \stackrel{\text{def}}{=} \{x' : x \in \Sigma\}$ is the set of all differential symbols x' for variables $x \in \Sigma$. The total derivative $(\eta)'$ of a polynomial term η is not a polynomial term, but may mention differential symbols such as x' in addition to the symbols that where in η to begin with. All syntactic elements of those differential terms are easy to interpret based on the semantics of terms defined in Lecture 2, except for the differential symbols. What is the meaning of a differential symbol x' ?

Before you read on, see if you can find the answer for yourself.

6 The Meaning of Prime

The meaning $\llbracket x \rrbracket_\nu$ of a variable symbol x is defined by the state ν as $\llbracket x \rrbracket_\nu = \nu(x)$. It is crucial to notice that the meaning of a differential symbol x' cannot be defined in a state ν , because derivatives do not even exist in isolated points. It is meaningless to ask for the change of the value of x over time in a single isolated state ν .

Along a (differentiable) continuous evolution $\varphi : [0, r] \rightarrow \mathcal{S}$ of a system, however, we can make sense of what x' means. At any point in time $\zeta \in [0, r]$ along such a continuous evolution φ , the differential symbol x' can be taken to mean the time-derivative of the value $\llbracket x \rrbracket_{\varphi(\zeta)}$ of x at ζ [Pla10a]. That is, at any point in time ζ along φ , it makes sense to give x' the meaning of the rate of change of the value of x over time along φ .

Definition 2 (Semantics of differential symbols). The value of x' at time $\zeta \in [0, r]$ of a differentiable function $\varphi : [0, r] \rightarrow \mathcal{S}$ of some duration $r \in \mathbb{R}$ is defined as the analytic time-derivative at ζ :

$$\llbracket x' \rrbracket_{\varphi(\zeta)} = \frac{d\varphi(t)(x)}{dt}(\zeta)$$

Intuitively, $\llbracket x' \rrbracket_{\varphi(\zeta)}$ is determined by considering how the value $\llbracket x \rrbracket_{\varphi(\zeta)} = \varphi(\zeta)(x)$ of x changes along the function φ when we change time ζ “only a little bit”. Visually, it corresponds to the slope of the tangent of the value of x at time ζ ; see Fig. 3.

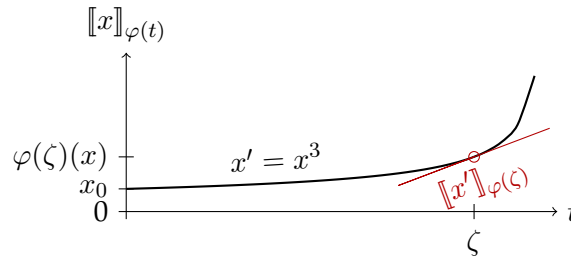


Figure 3: Semantics of differential symbols

Yet, what exactly do we know about the right-hand side in Def. 2, i.e. the time-derivative of the value of x along φ at time ζ ? For differentiable φ , that analytic time-derivative is always defined, but that does not mean it would be computable for any arbitrary φ . If, however, the continuous evolution φ follows a differential equation $x' = \theta$, i.e. φ solves $x' = \theta$, then $\llbracket x' \rrbracket_{\varphi(\zeta)}$ can be described easily in terms of that differential equation, because at any time $\zeta \in [0, r]$ the time-derivative of the value of x is $\llbracket \theta \rrbracket_{\varphi(\zeta)}$ simply by definition of what it means for φ to be a solution of $x' = \theta$ (cf. [Lecture 2 on Differential Equations & Domains](#)).

Now Def. 1 defines how to derive a term η syntactically to form $(\eta)'$ and Def. 2 defines how to interpret the differential symbols x' that occur in the total derivative $(\eta)'$. When interpreting all differential symbols as defined in Def. 2 for an evolution φ that follows

the differential equation $x' = \theta$, this defines a value for the derivative $(\eta)'$ of any term η along that function φ . What does this value mean? How does it relate to how the value of η changes over time?

Before you read on, see if you can find the answer for yourself.

When interpreting differential symbols by derivatives along a function φ , the value of $(\eta)'$ at any time ζ coincides with the analytic time-derivative of the value of η at ζ . The key insight behind this is the derivation lemma, a differential analogue of the substitution lemma.

Expedition 2 (Substitutions in logic). The substitution lemma shows that syntactic substitution has the same effect as changing the value of the variables it replaces.

Lemma 3 (Substitution Lemma [Pla10b, Lemma 2.2]). *Let the substitution of θ for x in ϕ to form ϕ_x^θ be admissible; then*

$$\text{for each } \nu : \llbracket \phi_x^\theta \rrbracket_\nu = \llbracket \phi \rrbracket_{\nu_x^\theta} \text{ where } e = \llbracket \theta \rrbracket_\nu$$

That is, semantically evaluating ϕ after modifying the interpretation of the symbol x replaced by its new value $\llbracket \theta \rrbracket_\nu$ is the same as semantically evaluating the result of syntactically substituting x by θ in ϕ in the original state.

The substitution lemma is a very powerful tool, because, among other things, it can be used to replace equals for equals without changing the valuation (substitution property). If we know that x and θ have the same value in ν , then we can substitute θ for x in a formula ϕ (if admissible) without changing the truth-value of ϕ , that is:

Lemma 4 (Substitution property [Pla10b, Lemma 2.3]). *If $\nu \models x = \theta$, then $\nu \models \phi \leftrightarrow \phi_x^\theta$ for any formula ϕ for which the substitution replacing x with θ is admissible.*

The substitution property implies that equals can be substituted for equals, i.e. left-hand sides of equations can be substituted by right-hand sides of equations within formulas in which the equations hold. Lemma 4 is the reason why the equality substitution proof rules $=l, =r$ from [Lecture 6 on Truth & Proof](#) are sound.

The following central lemma, which is the differential counterpart of the substitution lemma, establishes the connection between syntactic derivation of terms and semantic differentiation as an analytic operation to obtain analytic derivatives of valuations along differential state flows. It will allow us to draw analytic conclusions about the behaviour of a system along differential equations from the truth of purely algebraic formulas obtained by syntactic derivation. In a nutshell, the following lemma shows that, along a flow, analytic derivatives of valuations coincide with valuations of syntactic derivations.

Lemma 5 (Derivation lemma). *Let $\varphi : [0, r] \rightarrow \mathcal{S}$ be a differentiable function of duration $r > 0$. Then for all terms η that are defined all along φ and all times $\zeta \in [0, r]$:*

$$\frac{d \llbracket \eta \rrbracket_{\varphi(t)}}{dt}(\zeta) = \llbracket (\eta)' \rrbracket_{\varphi(\zeta)}$$

where differential symbols are interpreted according to Def. 2. In particular, $\llbracket \eta \rrbracket_{\varphi(\zeta)}$ is continuously differentiable.

Proof. The proof is an inductive consequence of the correspondence of the semantics of differential symbols and analytic derivatives along a flow (Def. 2). It uses the assumption that φ remains within the domain of definition of η and is continuously differentiable in all variables of η . In particular, all denominators are nonzero during φ .

- If η is a variable x , the conjecture holds immediately by Def. 2:

$$\frac{d \llbracket x \rrbracket_{\varphi(t)}}{dt}(\zeta) = \frac{d \varphi(t)(x)}{dt}(\zeta) = \llbracket (x)' \rrbracket_{\varphi(\zeta)}.$$

The derivative exists, because φ is assumed to be differentiable.

- If η is of the form $a + b$, the desired result can be obtained by using the properties of analytic derivatives, syntactic derivations (Def. 1), and valuation of terms (Lecture 2):

$$\begin{aligned} & \frac{d}{dt}(\llbracket a + b \rrbracket_{\varphi(t)})(\zeta) \\ &= \frac{d}{dt}(\llbracket a \rrbracket_{\varphi(t)} + \llbracket b \rrbracket_{\varphi(t)})(\zeta) && \llbracket \cdot \rrbracket_{\nu} \text{ homomorphic for } + \\ &= \frac{d}{dt}(\llbracket a \rrbracket_{\varphi(t)})(\zeta) + \frac{d}{dt}(\llbracket b \rrbracket_{\varphi(t)})(\zeta) && \frac{d}{dt} \text{ is a (linear) derivation} \\ &= \llbracket (a)' \rrbracket_{\varphi(\zeta)} + \llbracket (b)' \rrbracket_{\varphi(\zeta)} && \text{by induction hypothesis} \\ &= \llbracket (a)' + (b)' \rrbracket_{\varphi(\zeta)} && \llbracket \cdot \rrbracket_{\nu} \text{ homomorphic for } + \\ &= \llbracket (a + b)' \rrbracket_{\varphi(\zeta)} && (\cdot)' \text{ is a syntactic derivation} \end{aligned}$$

- The case where η is of the form $a - b$ is similar, using subtractivity (2d) of Def. 1.
- The case where η is of the form $a \cdot b$ is similar, using Leibniz product rule (2e) of

Def. 1:

$$\begin{aligned}
 & \frac{d}{dt}(\llbracket a \cdot b \rrbracket_{\varphi(t)})(\zeta) \\
 &= \frac{d}{dt}(\llbracket a \rrbracket_{\varphi(t)} \cdot \llbracket b \rrbracket_{\varphi(t)})(\zeta) && \llbracket \cdot \rrbracket_{\nu} \text{ homomorphic for } \cdot \\
 &= \frac{d}{dt}(\llbracket a \rrbracket_{\varphi(t)})(\zeta) \cdot \llbracket b \rrbracket_{\varphi(\zeta)} + \llbracket a \rrbracket_{\varphi(\zeta)} \cdot \frac{d}{dt}(\llbracket b \rrbracket_{\varphi(t)})(\zeta) && \frac{d}{dt} \text{ is a (Leibniz) derivation} \\
 &= \llbracket (a)' \rrbracket_{\varphi(\zeta)} \cdot \llbracket b \rrbracket_{\varphi(\zeta)} + \llbracket a \rrbracket_{\varphi(\zeta)} \cdot \llbracket (b)' \rrbracket_{\varphi(\zeta)} && \text{by induction hypothesis} \\
 &= \llbracket (a)' \cdot b \rrbracket_{\varphi(\zeta)} + \llbracket a \cdot (b)' \rrbracket_{\varphi(\zeta)} && \llbracket \cdot \rrbracket_{\nu} \text{ homomorphic for } \cdot \\
 &= \llbracket (a)' \cdot b + a \cdot (b)' \rrbracket_{\varphi(\zeta)} && \llbracket \cdot \rrbracket_{\nu} \text{ homomorphic for } + \\
 &= \llbracket (a \cdot b)' \rrbracket_{\varphi(\zeta)} && (\cdot)' \text{ is a syntactic derivation}
 \end{aligned}$$

- The case where η is of the form a/b uses (2f) of Def. 1 and further depends on the assumption that $b \neq 0$ along φ . This holds as the value of η is assumed to be defined all along state flow φ .
- The values of numbers $r \in \mathbb{Q}$ do not change during a state flow (in fact, they are not affected by the state at all); hence their derivative is $(r)' = 0$. \square

Note 11 (The derivation lemma clou). Lemma 5 shows that analytic derivatives coincide with syntactic derivations. The clou with Lemma 5 is that it equates precise but sophisticated analytic derivatives with tame and computable syntactic derivations. The analytic derivatives on the left-hand side of Lemma 5 are mathematically precise and pinpoint exactly what we are interested in: the rate of change of the value of η along φ . But they are unwieldy for computers, because analytic derivatives are ultimately defined in terms of limit processes. The syntactic derivations on the right-hand side of Lemma 5 are computationally tame, because they can be computed easily by the simple recursive construction in Def. 1. But the syntactic derivations need to be aligned with the intended analytic derivatives, which is what Lemma 5 is good for. And, of course, deriving polynomials and rational functions is much easier syntactically than by unpacking the meaning of analytic derivatives in terms of limit processes.

Lemma 5 shows that the value of the total derivative of a term coincides with the analytic derivative of the term, provided that differential symbols are interpreted according to Def. 2. Now, along a differential equation $x' = \theta$, the differential symbols themselves actually have a simple interpretation, the interpretation determined by the differential equation. Putting these thoughts together leads to replacing differential symbols with the corresponding right-hand sides of their respective differential equations. That is, replacing left-hand sides of differential equations with their right-hand sides.

Note 12. The direction into which the value of a term η evolves as the system follows a differential equation $x' = \theta$ depends on the derivation η' of the term η and on the differential equation $x' = \theta$ that locally describes the evolution of x over time.

The substitution property can replace equals for equals (Lemma 4). It can be lifted to differential equations such that differential equations can be used for equivalent substitutions along continuous flows respecting these differential equations. The following lemma can be used to substitute right-hand sides of differential equations for the left-hand side derivatives in flows along which these differential equations hold.

Lemma 6 (Differential substitution property for terms). If $\varphi : [0, r] \rightarrow \mathcal{S}$ solves the differential equation $x' = \theta$, i.e. $\varphi \models x' = \theta$, then $\varphi \models (\eta)' = (\eta)'_{x'}$ for all terms η , i.e.:

$$\llbracket (\eta)' \rrbracket_{\varphi(\zeta)} = \llbracket (\eta)'_{x'} \rrbracket_{\varphi(\zeta)} \quad \text{for all } \zeta \in [0, r]$$

Proof. The proof is a simple inductive consequence of Lemma 5 using that $\llbracket x' \rrbracket_{\varphi(\zeta)} = \llbracket \theta \rrbracket_{\varphi(\zeta)}$ at each time ζ in the domain of φ . \square

The operation mapping term η to $(\eta)'_{x'}$ is called *Lie-derivative* of η with respect to $x' = \theta$.

Differential substitution of the differential equation $d' = e, e' = -d$ from (3) into (4) results in

$$(d^2 + e^2 - r^2)'_{d' e'}^{-d} = (2dd' + 2ee' - 2rr')_{d' e'}^{-d} = 2de + 2e(-d) + 2rr'$$

Oops, that did not make all differential symbols disappear, because r' is still around, since r did not have a differential equation in (3). Stepping back, what we mean by a differential equation like $d' = e, e' = -d$ that does not mention r' is that r is not supposed to change. If r is supposed to change during a continuous evolution, there has to be a differential equation for r .

Note 14 (Explicit change). Hybrid programs are explicit change: nothing changes unless an assignment or differential equation specifies how (compare the semantics from Lecture 3). In particular, if a differential equation (system) $x' = \theta$ does not mention z' , then the variable z does not change during $x' = \theta$, so the differential equation systems $x' = \theta$ and $x' = \theta, z' = 0$ are equivalent.

We will often assume $z' = 0$ without further notice for variables z that do not change during a differential equation.

Since (3) does not have a differential equation for r , Note 14 implies that its differential equation $d' = e, e' = -d$ is equivalent to $d' = e, e' = -d, r' = 0$. Hence, when adding zero derivatives for all unchanged variables, differential substitution of the differential equation $d' = e, e' = -d$ along with the explicit-change assumption $r' = 0$ into (4) gives

$$(d^2 + e^2 - r^2)'_{d' e' r'}^{-d 0} = (2dd' + 2ee' - 2rr')_{d' e' r'}^{-d 0} = 2de + 2e(-d) \quad (5)$$

This is good news, because the last part of (5) is a standard term of first-order logic of real arithmetic, because it no longer has any differential symbols. So we can make sense of $2de + 2e(-d)$ and, by Lemma 6, its value along a solution of $d' = e, e' = -d$ is the same as that of the derivative $(d^2 + e^2 - r^2)'$, which, by Lemma 5 is the same as the value of the time-derivative of the original term $d^2 + e^2 - r^2$ along such a solution. Simple arithmetic shows that the term $2de + 2e(-d)$ in (5) is 0. Consequently, by Lemma 5 and Lemma 6, the time-derivative of the term $d^2 + e^2 - r^2$ in the postcondition of (3) is 0 along any solution φ of its differential equation:

$$\begin{aligned} \frac{d[d^2 + e^2 - r^2]_{\varphi(t)}(\zeta)}{dt} &\stackrel{\text{Lem5}}{=} [(d^2 + e^2 - r^2)']_{\varphi(\zeta)} \\ &\stackrel{\text{Lem6}}{=} [(d^2 + e^2 - r^2)']_{d' e' -d 0}^e_{r'}_{\varphi(\zeta)} \\ &\stackrel{(5)}{=} [2de + 2e(-d)]_{\varphi(\zeta)} = 0 \end{aligned}$$

for all times ζ . That means that the value of $d^2 + e^2 - r^2$ never changes during the rotation, and, hence (3) is valid, because $d^2 + e^2 - r^2$ stays 0 if it was 0 in the beginning, which is what (3) assumes.

This is amazing, because we found out that the value of $d^2 + e^2 - r^2$ does not change over time (its time-derivative is zero) along the differential equation $d' = e, e' = -d$. And we found that out without ever solving the differential equation, just by a few lines of simple symbolic computations. We only need to make sure to systematize this reasoning and make it accessible in the dC proof calculus by reflecting it in a proof rule, preferably one that is much more general than the special argument we needed to convince ourselves that (3) was valid.

7 Differential Invariant Terms

In order to be able to use the above reasoning as part of a sequent proof, we need to capture such arguments in a proof rule, preferably one that is more general than this particular argument. The argument is not specific to the term $d^2 + e^2 - r^2$ but works for any other term η and for any differential equation $x' = \theta$.

What we set out to find is a general proof rule for concluding properties of differential equations from properties of derivatives. As a first shot, we stay with equations of the form $\eta = 0$, which gives us soundness for the following proof rule.

Lemma 7 (Differential invariant terms). *The following special case of the differential invariants proof rule is sound, i.e. if its premise is valid then so is its conclusion:*

$$(DI=0) \frac{\vdash \eta'_{x'}^\theta = 0}{\eta = 0 \vdash [x' = \theta]\eta = 0}$$

Proof. Assume the premise $\eta'_{x'}^\theta = 0$ to be valid, i.e. true in all states. In order to prove that the conclusion $\eta = 0 \vdash [x' = \theta]\eta = 0$ is valid, consider any state ν . Assume that

$$\frac{d\llbracket \eta \rrbracket_{\varphi(t)}}{dt}(\zeta) \stackrel{\text{Lem 5}}{=} \llbracket (\eta)' \rrbracket_{\varphi(\zeta)} \stackrel{\text{Lem 6}}{=} \llbracket (\eta)'_{x'} \rrbracket_{\varphi(\zeta)} \stackrel{\text{premise}}{=} 0 \quad (6)$$

This proof rule enables us to prove $\text{d}\mathcal{L}$ formula (3) easily in $\text{d}\mathcal{L}$'s sequent calculus:

	*	
\mathbb{R}	$\vdash 2de + 2e(-d) - 0 = 0$	
	$\vdash (2dd' + 2ee' - 2rr')_{d' e' r'}^e - d \ 0 = 0$	
$\text{DI}=0$	$d^2 + e^2 - r^2 = 0 \vdash [d' = e, e' = -d] d^2 + e^2 - r^2 = 0$	
$\rightarrow r$	$\vdash d^2 + e^2 - r^2 = 0 \rightarrow [d' = e, e' = -d] d^2 + e^2 - r^2 = 0$	

Taking a step back, this is an exciting development, because, thanks to differential invariants, the property (3) of a differential equation with a nontrivial solution has a very simple proof that we can easily check. The proof did not need to solve the differential equation, which has infinitely many solutions with combinations of trigonometric functions.⁵ The proof only required deriving the postcondition and substituting the differential equation in.

$$d(t) = a \cos t + b \sin t, \quad e(t) = b \cos t - a \sin t$$

ANDRÉ PLATZER

This is a possible way of proving the original (1), but also unnecessarily complicated. Differential invariants can prove (1) directly once we generalize proof rule $\text{DI}_{=0}$ appropriately. For other purposes, however, it is still important to have the principle of generalization Note 16 in our repertoire of proof techniques.

9 Example Proofs

Of course, differential invariants are just as helpful for proving properties of other differential equations.

Example 8 (Self-crossing). Another example is the following invariant property illustrated in Fig. 4:

$$x^2 + x^3 - y^2 - c = 0 \rightarrow [x' = -2y, y' = -2x - 3x^2] x^2 + x^3 - y^2 - c = 0$$

This $\text{d}\mathcal{L}$ formula proves easily using $\text{DI}_{=0}$:

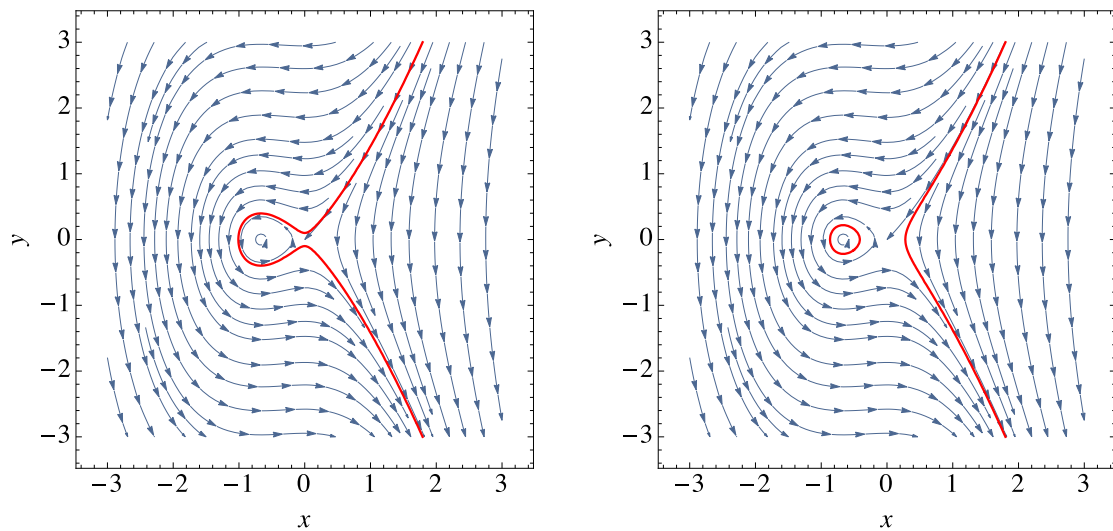


Figure 4: Two differential invariants of the indicated dynamics (illustrated in thick red) for different values of c

	*
\mathbb{R}	$\vdash 2x(-2y) + 3x^2(-2y) - 2y(-2x - 3x^2) = 0$
	$\vdash (2xx' + 3x^2x' - 2yy' - c') \frac{-2y}{x'} \frac{-2x-3x^2}{y'} \frac{0}{c'} = 0$
$\text{DI}_{=0}$	$x^2 + x^3 - y^2 - c = 0 \vdash [x' = -2y, y' = -2x - 3x^2] x^2 + x^3 - y^2 - c = 0$
$\rightarrow r$	$\vdash x^2 + x^3 - y^2 - c = 0 \rightarrow [x' = -2y, y' = -2x - 3x^2] x^2 + x^3 - y^2 - c = 0$

See [«Self-crossing polynomial invariant»](#)

Example 9 (Moztkin). Another nice example is the Moztkin polynomial, which is an invariant of the following dynamics (see Fig. 5):

$$x^4y^2 + x^2y^4 - 3x^2y^2 + 1 = c \rightarrow$$

$$[x' = 2x^4y + 4x^2y^3 - 6x^2y, y' = -4x^3y^2 - 2xy^4 + 6xy^2] x^4y^2 + x^2y^4 - 3x^2y^2 + 1 = c$$

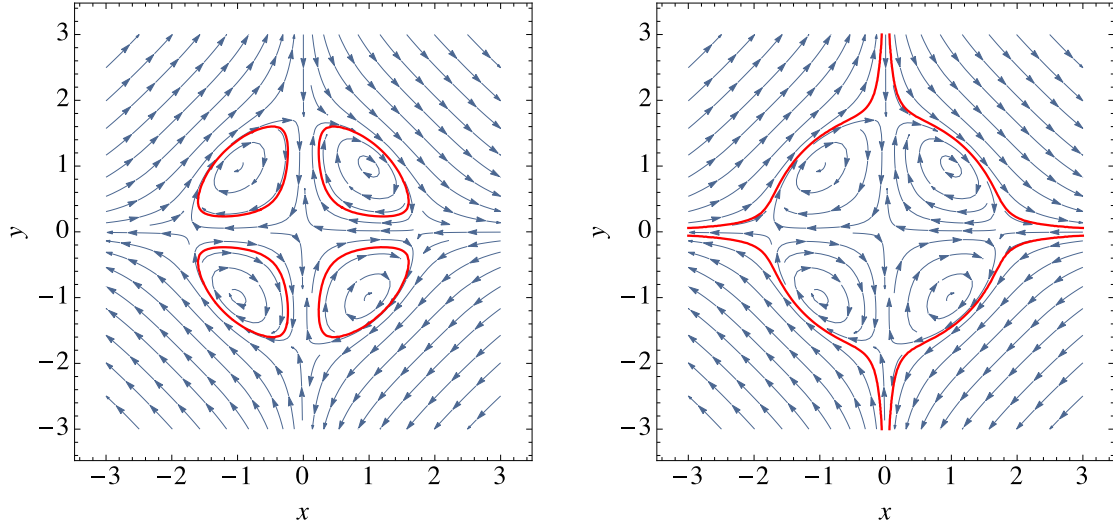


Figure 5: Two differential invariants of the indicated dynamics is the Moztkin polynomial (illustrated in thick red) for different values of c

This $d\mathcal{L}$ formula proves easily using $\text{DI}_{=0}$, again after normalizing the equation to have right-hand side 0:

$$\begin{array}{l}
 * \\
 \hline
 \mathbb{R} \quad \vdash 0 = 0 \\
 \hline
 \vdash ((x^4y^2 + x^2y^4 - 3x^2y^2 + 1 - c)')_{\substack{x' \\ y'}}^{2x^4y+4x^2y^3-6x^2y \quad -4x^3y^2-2xy^4+6xy^2} = 0 \\
 \hline
 \text{DI}_{=0} \quad \dots \vdash [x' = 2x^4y + 4x^2y^3 - 6x^2y, y' = -4x^3y^2 - 2xy^4 + 6xy^2] x^4y^2 + x^2y^4 - 3x^2y^2 + 1 - c = 0 \\
 \hline
 \rightarrow r \quad \vdash \dots \rightarrow [x' = 2x^4y + 4x^2y^3 - 6x^2y, y' = -4x^3y^2 - 2xy^4 + 6xy^2] x^4y^2 + x^2y^4 - 3x^2y^2 + 1 - c = 0
 \end{array}$$

This time, the proof step that comes without a label is simple, but requires some space:

$$(x^4y^2 + x^2y^4 - 3x^2y^2 + 1 - c)' = (4x^3y^2 + 2xy^4 - 6xy^2)x' + (2x^4y + 4x^2y^3 - 6x^2y)y'$$

After substituting in the differential equation, this gives

$$(4x^3y^2 + 2xy^4 - 6xy^2)(2x^4y + 4x^2y^3 - 6x^2y) + (2x^4y + 4x^2y^3 - 6x^2y)(-4x^3y^2 - 2xy^4 + 6xy^2)$$

which simplifies to 0 after expanding the polynomials, and, thus, leads to the equation $0 = 0$, which is easy to prove.

See [«Motzkin polynomial invariant»](#). Note that the arithmetic complexity reduces when hiding unnecessary contexts as shown in [Lecture 6 on Truth & Proof](#).

Thanks to Andrew Sogokon for the nice Example 9.

10 Differential Invariant Terms and Invariant Functions

It is not a coincidence that these examples were provable by differential invariant proof rule $\text{DI}_{=0}$, because that proof rule can handle arbitrary invariant functions.

Expedition 3 (Lie characterization of invariant functions). The proof rule $\text{DI}_{=0}$ works by deriving the postcondition and substituting the differential equation in:

$$(\text{DI}_{=0}) \frac{\vdash \eta'_{x'}^\theta = 0}{\eta = 0 \vdash [x' = \theta]\eta = 0}$$

There is something quite peculiar about $\text{DI}_{=0}$. Its premise is independent of the constant term in η . If, for any constant symbol c , the formula $\eta = 0$ is replaced by $\eta - c = 0$ in the conclusion, then the premise of $\text{DI}_{=0}$ stays the same, because $c' = 0$. Consequently, if $\text{DI}_{=0}$ proves

$$\eta = 0 \vdash [x' = \theta]\eta = 0$$

then it also proves

$$\eta - c = 0 \vdash [x' = \theta]\eta - c = 0 \quad (7)$$

for any constant c . This observation is the basis for a more general result, which simultaneously proves all formulas (7) for all c from the premise of $\text{DI}_{=0}$.

On open domains, equational differential invariants are even a necessary and sufficient characterization of such *invariant functions*, i.e. functions that are invariant along the dynamics of a system, because, whatever value c that function had in the initial state, the value will stay the same forever. The equational case of differential invariants are intimately related to the seminal work by Sophus Lie on what are now called Lie groups [[Lie93](#), [Lie97](#)].

Theorem 10 (Lie [[Pla12b](#)]). Let $x' = \theta$ be a differential equation system and H a domain, i.e., a first-order formula of real arithmetic characterizing an open set. The following proof rule is a sound global equivalence rule, i.e. the conclusion is valid if and only if the premise is:

$$(\text{DI}_c) \frac{H \vdash \eta'_{x'}^\theta = 0}{\vdash \forall c (\eta = c \rightarrow [x' = \theta \ \& \ H]\eta = c)}$$

Despite the power that differential invariant terms offer, challenges lie ahead in proving properties. Theorem 10 gives an indication where challenges remain.

Example 11 (Generalizing differential invariants). The following $d\mathcal{L}$ formula is valid

$$x^2 + y^2 = 0 \rightarrow [x' = 4y^3, y' = -4x^3] x^2 + y^2 = 0 \quad (8)$$

but cannot be proved directly using $DI_{=0}$, because $x^2 + y^2$ is no invariant function of the dynamics. In combination with generalization (\boxed{gen}' to change the postcondition to the equivalent $x^4 + y^4 = 0$) and a *cut* (to change the antecedent to the equivalent $x^4 + y^4 = 0$), however, there is a proof using differential invariants $DI_{=0}$:

$$\begin{array}{c}
 \mathbb{R} \quad \frac{\frac{*}{\vdash 4x^3(4y^3) + 4y^3(-4x^3) = 0}}{\vdash (4x^3x' + 4y^3y') \frac{4y^3 - 4x^3}{x' y'} = 0} \\
 \frac{DI_{=0} \quad x^4 + y^4 = 0 \vdash [x' = 4y^3, y' = -4x^3] x^4 + y^4 = 0}{\text{cut}, \boxed{gen}' \quad x^2 + y^2 = 0 \vdash [x' = 4y^3, y' = -4x^3] x^2 + y^2 = 0} \\
 \rightarrow r \quad \vdash x^2 + y^2 = 0 \rightarrow [x' = 4y^3, y' = -4x^3] x^2 + y^2 = 0
 \end{array}$$

The use of \boxed{gen}' leads to another branch $x^4 + y^4 = 0 \vdash x^2 + y^2 = 0$ that is elided above. Similarly, *cut* leads to another branch $x^2 + y^2 = 0 \vdash x^4 + y^4 = 0$ that is also elided. Both prove easily by real arithmetic (\mathbb{R}).

See \ll [Differential invariant after generalization](#) \gg

How could this happen? How could the original formula (8) be provable only after generalizing its postcondition to $x^4 + y^4 = 0$ and not before?

Note 18 (Strengthening induction hypotheses). *An important phenomenon we already encountered in [Lecture 7 on Loops & Invariants](#) and other uses of induction is that, sometimes, the only way to prove a property is to strengthen the induction hypothesis. Differential invariants are no exception. It is worth noting, however, that the inductive structure in differential invariants includes their differential structure. And, indeed, the derivatives of $x^4 + y^4 = 0$ are different and more conducive for an inductive proof than those of $x^2 + y^2 = 0$ even if both have the same set of solutions.*

Theorem 10 explains why $x^2 + y^2 = 0$ was doomed to fail as a differential invariant while $x^4 + y^4 = 0$ succeeded. All formulas of the form $x^4 + y^4 = c$ for all c are invariants of the dynamics in (8), because the proof succeeded. But $x^2 + y^2 = c$ only is an invariant for the lucky choice $c = 0$ and only equivalent to $x^4 + y^4 = 0$ for this case.

There also is a way of deciding equational invariants of algebraic differential equations using a higher-order generalization of differential invariants called differential radical invariants [GP14].

11 Summary

This lecture showed one form of differential invariants: the form where the differential invariants are terms whose value always stays 0 along all solutions of a differential equation. The next lecture will investigate more general forms of differential invariants and more advanced proof principles for differential equations. They all share the important discovery in today's lecture: that properties of differential equations can be proved using the differential equation rather than its solution.

The most important technical insight of today's lecture was that even very complicated behavior that is defined by mathematical properties of the semantics can be captured by purely syntactical proof principles using syntactic derivations. The derivation lemma proved that the values of the (easily computable) syntactic derivations coincides with the analytic derivatives of the values. The differential substitution lemma allowed us the intuitive operation of substituting differential equations into terms. Proving properties of differential equations using these simple proof principles is much more civilized and effective than working with solutions of differential equations. The proofs are also computationally easier, because the proof arguments are local.

The principles begun in this lecture have more potential, though, and are not limited to proving only properties of the rather limited form $\eta = 0$. Subsequent lectures will make use of the results obtained and build on the derivation lemma and differential substitution lemma to develop more general proof principles for differential equations.

Exercises

Exercise 1. Note 2 explained that $(x' = \theta)^*$ is equivalent to $x' = \theta$. Does the same hold for differential equations with evolution domain constraints? Are $(x' = \theta \ \& \ H)^*$ and $x' = \theta \ \& \ H$ equivalent or not? Justify or modify the statement and justify the variation.

Exercise 2. We argued that $\text{d}\mathcal{L}$ formulas (1) and (3) are equivalent and have then gone on to find a proof of (3). Continue this proof of (3) to a proof of (1) using the generalization rule $\llbracket \text{gen}' \rrbracket$ and the *cut* rule.

Exercise 3. Prove the cases of Lemma 5 where η is of the form $a - b$ and a/b .

Exercise 4. What happens in the proof of Lemma 7 if there is no solution φ ? Show that this is not a counterexample to proof rule $\text{DI}_{=0}$, but that the rule is sound in that case.

Exercise 5. Carry out the polynomial computations needed to prove Example 9 using proof rule $\text{DI}_{=0}$.

Exercise 6. Prove the following $\text{d}\mathcal{L}$ formula using differential invariants:

$$xy = c \rightarrow [x' = -x, y' = y, z' = -z]xy = c$$

Exercise 7. Prove the following $\text{d}\mathcal{L}$ formula using differential invariants:

$$x^2 + 4xy - 2y^3 - y = 1 \rightarrow [x' = -1 + 4x - 6y^2, y' = -2x - 4y]x^2 + 4xy - 2y^3 - y = 1$$

Exercise 8. Prove the following $d\mathcal{L}$ formula using differential invariants:

$$x^2 + \frac{x^3}{3} = c \rightarrow [x' = y^2, y' = -2x]x^2 + \frac{x^3}{3} = c$$

Exercise 9 (Hénon-Heiles). Prove a differential invariant of a Hénon-Heiles system:

$$\frac{1}{2}(u^2 + v^2 + Ax^2 + By^2) + x^2y - \frac{1}{3}\varepsilon y^3 = 0 \rightarrow$$

$$[x' = u, y' = v, u' = -Ax - 2xy, v' = -By + \varepsilon y^2 - x^2]\frac{1}{2}(u^2 + v^2 + Ax^2 + By^2) + x^2y - \frac{1}{3}\varepsilon y^3 = 0$$

References

- [GP14] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. doi:10.1007/978-3-642-54862-8_19.
- [Kol72] Ellis Robert Kolchin. *Differential Algebra and Algebraic Groups*. Academic Press, New York, 1972.
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Lie93] Sophus Lie. *Vorlesungen über kontinuierliche Gruppen mit geometrischen und anderen Anwendungen*. Teubner, Leipzig, 1893.
- [Lie97] Sophus Lie. Über Integralinvarianten und ihre Verwertung für die Theorie der Differentialgleichungen. *Leipz. Berichte*, 49:369–410, 1897.
- [PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. doi:10.1007/978-3-540-70545-1_17.
- [PC09] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. Special issue for selected papers from CAV’08. doi:10.1007/s10703-009-0079-8.
- [Pla08a] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.
- [Pla08b] André Platzer. *Differential Dynamic Logics: Automated Theorem Proving for Hybrid Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Dec 2008. Appeared with Springer.
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. doi:10.1093/logcom/exn070.
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.

- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:10.1109/LICS.2012.64.
- [Pla12b] André Platzer. A differential operator approach to equational differential invariants. In Lennart Beringer and Amy Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 28–48. Springer, 2012. doi:10.1007/978-3-642-32347-8_3.
- [Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:10.1109/LICS.2012.13.
- [Pla12d] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. doi:10.2168/LMCS-8(4:16)2012.
- [Zei03] Eberhard Zeidler, editor. *Teubner-Taschenbuch der Mathematik*. Teubner, 2003.

Lecture Notes on Differential Equations & Proofs

André Platzer

Carnegie Mellon University
Lecture 11

1. Introduction

[Lecture 10 on Differential Equations & Differential Invariants](#) introduced equational differential invariants of the form $\eta = 0$ for differential equations that are much more general than the ones supported by axiom ['] from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#). Axiom ['] replaces properties of differential equations with universally quantified properties of solutions, but is limited to differential equations that have explicit closed-form solutions whose arithmetic can be handled (mostly polynomials or rational functions). But axiom ['] works for any arbitrary postcondition. The equational differential invariant proof rule $\text{DI}_{=0}$ supports general differential equations, but was limited to equational postconditions of the form $\eta = 0$.

The goal of this lecture is to generalize the differential invariant proof rules to work for more general postconditions but retaining the flexibility with the differential equations that differential invariants provide. Indeed, the principles developed in [Lecture 10](#) generalize beautifully to logical formulas other than the limited form $\eta = 0$. This lecture will establish generalizations that make the differential invariant proof rule work for formulas F of more general forms. The most important part will be soundly defining the total derivative F' , because the basic shape of the differential invariants proof rule stays the same:

$$\frac{\vdash (F')_{x'}^\theta}{F \vdash [x' = \theta] F}$$

More details can be found in [[Pla10b](#), Chapter 3.5] and [[Pla10a](#), [Pla12d](#), [Pla12a](#), [Pla12b](#)]. Differential invariants were originally conceived in 2008 [[Pla10a](#), [Pla08](#)] and later used for an automatic proof procedure for hybrid systems [[PC08](#)].

This lecture advances the capabilities of differential invariants begun in [Lecture 10 on Differential Equations & Differential Invariants](#) and continues to be of central significance for the Foundations of Cyber-Physical Systems. The most important learning goals of this lecture are:

Modeling and Control: This lecture continues the study of the core principles behind CPS by developing a deeper understanding of how continuous dynamical behavior affects the truth of logical formulas. The differential invariants developed in this lecture also have a significance for developing models and controls using the design-by-invariant principle.

Computational Thinking: This lecture exploits computational thinking continuing the surprising analogies among discrete dynamics and continuous dynamics discovered in [Lecture 10](#). This lecture is devoted to rigorous reasoning about the differential equations in CPS models, which is crucial for understanding the continuous behavior that CPS exhibit over time. This lecture systematically expands on the differential invariant terms for equational properties of differential equations developed in [Lecture 10](#) and generalizes the same core principles to the study of general properties of differential equations. Computational thinking is exploited in a second way by generalizing Gentzen’s cut principle, which is of seminal significance in discrete logic, to differential equations. This lecture continues the *axiomatization* of differential dynamic logic $d\mathcal{L}$ [[Pla12c](#), [Pla12a](#)] pursued since [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) and lifts $d\mathcal{L}$ ’s proof techniques to systems with more complex properties of more complex differential equations. The concepts developed in this lecture continue the differential facet illustrating the more general relation of *syntax* (which is notation), *semantics* (what carries meaning), and *axiomatics* (which internalizes semantic relations into universal syntactic transformations). These concepts and their relations jointly form the significant *logical trinity* of syntax, semantics, and axiomatics. Finally, the verification techniques developed in this lecture are critical for verifying CPS models of appropriate scale and technical complexity.

CPS Skills: The focus in this lecture is on reasoning about differential equations. As a beneficial side effect, we will develop a better intuition for the operational effects involved in CPS by getting better tools for understanding how exactly state changes while the system follows a differential equation and what properties of it will not change.

2. Recall

Recall the following results from [Lecture 10 on Differential Equations & Differential Invariants](#):

Definition 1 (Derivation). The operator $(\cdot)'$ that is defined as follows on terms is called *syntactic (total) derivation*:

$$(r)' = 0 \quad \text{for numbers } r \in \mathbb{Q} \quad (1a)$$

$$(x)' = x' \quad \text{for variable } x \in \Sigma \quad (1b)$$

$$(a + b)' = (a)' + (b)' \quad (1c)$$

$$(a - b)' = (a)' - (b)' \quad (1d)$$

$$(a \cdot b)' = (a)' \cdot b + a \cdot (b)' \quad (1e)$$

$$(a/b)' = ((a)' \cdot b - a \cdot (b)')/b^2 \quad (1f)$$

Definition 2 (Semantics of differential symbols). The value of x' at time $\zeta \in [0, r]$ of a differentiable function $\varphi : [0, r] \rightarrow \mathcal{S}$ of some duration $r \in \mathbb{R}$ is defined as the analytic time-derivative at ζ :

$$\llbracket x' \rrbracket_{\varphi(\zeta)} = \frac{d\varphi(t)(x)}{dt}(\zeta)$$

Lemma 3 (Derivation lemma). Let $\varphi : [0, r] \rightarrow \mathcal{S}$ be a differentiable function of duration $r > 0$. Then for all terms η that are defined all along φ and all times $\zeta \in [0, r]$:

$$\frac{d \llbracket \eta \rrbracket_{\varphi(t)}}{dt}(\zeta) = \llbracket (\eta)' \rrbracket_{\varphi(\zeta)}$$

where differential symbols are interpreted according to Def. 2. In particular, $\llbracket \eta \rrbracket_{\varphi(\zeta)}$ is continuously differentiable.

Lemma 4 (Differential substitution property for terms). If $\varphi : [0, r] \rightarrow \mathcal{S}$ solves the differential equation $x' = \theta$, i.e. $\varphi \models x' = \theta$, then $\varphi \models (\eta)' = (\eta)'_{x'}^{\theta}$ for all terms η , i.e.:

$$\llbracket (\eta)' \rrbracket_{\varphi(\zeta)} = \llbracket (\eta)'_{x'}^{\theta} \rrbracket_{\varphi(\zeta)} \quad \text{for all } \zeta \in [0, r]$$

3. Differential Invariant Terms

Lecture 10 on [Differential Equations & Differential Invariants](#) proved soundness for a proof rule for differential invariant terms, which can be used to prove normalized invariant equations of the form $\eta = 0$.

Lemma 5 (Differential invariant terms). *The following special case of the differential invariants proof rule is sound, i.e. if its premise is valid then so is its conclusion:*

$$(DI_{=0}) \frac{\vdash \eta'_{x'} = 0}{\eta = 0 \vdash [x' = \theta]\eta = 0}$$

Differential invariant terms led to an indirect proof of

$$d^2 + e^2 = r^2 \rightarrow [d' = e, e' = -d]d^2 + e^2 = r^2 \quad (2)$$

by generalizing the formula using [\[gen'\]](#) and [cut](#) to

$$d^2 + e^2 - r^2 = 0 \rightarrow [d' = e, e' = -d]d^2 + e^2 - r^2 = 0 \quad (3)$$

after normalizing the equation to have 0 on the right-hand side as required by the differential invariant term proof rule [DI₌₀](#).

4. Equational Differential Invariants

There are more general logical formulas that we would like to prove to be invariants of differential equations, not just the polynomial equations normalized such that they are single terms equaling 0. For example, we should generalize differential invariants to enable a direct proof of (2) with an invariant of the form $\kappa = \eta$, rather than insisting on normalizing equations to the form $\eta = 0$ by generalization [\[gen'\]](#) first.

Thinking back of the soundness proof for [DI₌₀](#) in [Lecture 10](#), the argument was based on the value of the left-hand side term $h(t) = \llbracket \eta \rrbracket_{\varphi(t)}$ as a function of time t . The same argument can be made by considering the difference $h(t) = \llbracket \kappa - \eta \rrbracket_{\varphi(t)}$ instead to prove postconditions of the form $\kappa = \eta$. How does the inductive step for formula $\kappa = \eta$ need to be define to make a corresponding differential invariant proof rule sound? That is, for what premise is the following a sound proof rule?

$$\frac{\vdash ???}{\kappa = \eta \vdash [x' = \theta]\kappa = \eta}$$

Before you read on, see if you can find the answer for yourself.

Defining the total derivative of an equation $\kappa = \eta$ as

$$(\kappa = \eta)' \equiv ((\kappa)' = (\eta)')$$

results in a sound proof rule by a simple variation of the soundness proof for $\text{DI}_{=0}$ as sketched above. The resulting proof rule

$$(\text{DI}_{=}) \frac{\vdash (\kappa' = \eta')_{x'}^{\theta}}{\kappa = \eta \vdash [x' = \theta] \kappa = \eta}$$

for equational differential invariants captures the basic intuition that κ always stays equal to η if it has been initially (antecedent of conclusion) and the derivative of κ is the same as the derivative of η with respect to the differential equation $x' = \theta$. This intuition is made precise by Lemma 3 and Lemma 4. Instead of going through a proper soundness proof for $\text{DI}_{=}$, however, let's directly generalize the proof principles further and see if differential invariants can prove even more formulas for us. We will later prove soundness for the general differential invariant rule, from which $\text{DI}_{=}$ derives as a special case.

Example 6 (Rotational dynamics). The rotational dynamics $d' = e, e' = -d$ is complicated in that the solution involves trigonometric functions, which are generally outside decidable classes of arithmetic. Yet, we can easily prove interesting properties about it using DI and decidable polynomial arithmetic. For instance, $\text{DI}_{=}$ can directly prove formula (2), i.e. that $d^2 + e^2 = r^2$ is a differential invariant of the dynamics, using the following proof:

$$\begin{array}{c} * \\ \hline \mathbb{R} \vdash 2de + 2e(-d) = 0 \\ \hline \vdash (2dd' + 2ee' = 0)_{d' e'}^{e -d} \\ \hline \text{DI} \frac{d^2 + e^2 = r^2 \vdash [d' = e, e' = -d] d^2 + e^2 = r^2}{\vdash d^2 + e^2 = r^2 \rightarrow [d' = e, e' = -d] d^2 + e^2 = r^2} \\ \rightarrow r \end{array}$$

This proof is certainly much easier and more direct than the previous proof based on \square_{gen}' .

5. Differential Invariant Inequalities

The differential invariant proof rules considered so far give a good (initial) understanding of how to prove equational invariants. What about inequalities? How can they be proved?

Before you read on, see if you can find the answer for yourself.

The primary question to generalize the differential invariant proof rule is again how to define the total derivative

$$(\kappa \leq \eta)' \equiv ((\kappa)' \leq (\eta)')$$

which gives the differential invariant proof rule:

$$\frac{\vdash (\kappa' \leq \eta')_{x'}^\theta}{\kappa \leq \eta \vdash [x' = \theta] \kappa \leq \eta}$$

Example 7 (Cubic dynamics). Similarly, differential induction can easily prove that $\frac{1}{3} \leq 5x^2$ is an invariant of the cubic dynamics $x' = x^3$; see the proof in Fig. 7 for the dynamics in Fig. 1. To apply the differential induction rule **DI**, we again form the total deriva-

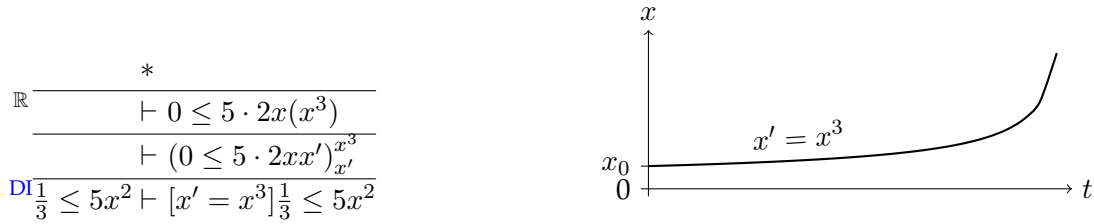


Figure 1: a Cubic dynamics proof

1b: Cubic dynamics

tive of the differential invariant $F \equiv \frac{1}{3} \leq 5x^2$, which gives the differential expression $F' \equiv (\frac{1}{3} \leq 5x^2)' \equiv 0 \leq 5 \cdot 2xx'$. Now, the differential induction rule **DI** takes into account that the derivative of state variable x along the dynamics is known. Substituting the differential equation $x' = x^3$ into the inequality yields $F'_{x'} \equiv 0 \leq 5 \cdot 2xx^3$, which is a valid formula and closes by quantifier elimination with \mathbb{R} .

Differential invariants that are inequalities are not just a minor variation of equational differential invariants, because they can prove more. That is, it can be shown [Pla12d] that there are valid formulas that can be proved using differential invariant inequalities but cannot be proved just using equations as differential invariants (**DI**₌). So sometimes, you need to be prepared to look for inequalities that you can use as differential invariants. The converse is not true. Everything that is provable using **DI**₌ is also provable using differential invariant inequalities [Pla12d], but you should still look for equational differential invariants if they give easier proofs.

Strict inequalities can also be used as differential invariants when defining their total derivatives as:

$$(\kappa < \eta)' \equiv ((\kappa)' < (\eta)')$$

It is easy to see (Exercise 1) that the following slightly relaxed definition would also be sound:

$$(\kappa < \eta)' \equiv ((\kappa)' \leq (\eta)')$$

Understanding that differential substitution is sound for formulas, i.e. replacing the left-hand side of the differential equation by its right-hand side, requires a few more

thoughts now, because the equational differential substitution principle Lemma 4 does not apply directly. The differential substitution principle not only works for terms, however, but also for *differential first-order formulas*, i.e. first-order formulas in which differential symbols occur:

Lemma 8 (Differential substitution property for differential formulas). *If $\varphi : [0, r] \rightarrow S$ solves the differential equation $x' = \theta$, i.e. $\varphi \models x' = \theta$, then $\varphi \models \mathcal{D} \leftrightarrow \mathcal{D}_{x'}^\theta$ for all differential first-order formulas \mathcal{D} , i.e. first-order formulas over $\Sigma \cup \Sigma'$.*

Proof. The proof is by using the Substitution Lemma [Pla10b, Lemma 2.2] for first-order logic on the basis of $\llbracket x' \rrbracket_{\varphi(\zeta)} = \llbracket \theta \rrbracket_{\varphi(\zeta)}$ at each time ζ in the domain of φ by Def. 2. \square

By Lemma 8, differential equations can always be substituted in along their solutions. Hence, the focus on developing differential invariant proof rules is in defining appropriate total derivatives, since Lemma 8 shows how to handle differential symbols by substitution.

Where do differential first-order formulas come from? They come from the analogue of the total derivation operator on formulas. On formulas, the total derivation operator applies the total derivation operator from Def. 1 to all terms in a first-order formula, yet it also flips disjunctions into conjunctions and existential quantifiers into universal quantifiers.

Example 9 (Rotational dynamics). An inequality property can be proved easily for the rotational dynamics $d' = e, e' = -d$ using the following proof:

$$\begin{array}{c}
 \text{R} \frac{*}{\frac{\vdash 2de + 2e(-d) \leq 0}{\vdash (2dd' + 2ee' \leq 0)_{d' e'}^e -d}} \\
 \text{DI} \frac{d^2 + e^2 \leq r^2 \vdash [d' = e, e' = -d]d^2 + e^2 \leq r^2}{\vdash d^2 + e^2 \leq r^2 \rightarrow [d' = e, e' = -d]d^2 + e^2 \leq r^2} \\
 \rightarrow \text{r}
 \end{array}$$

Example 10 (Damped oscillator). This proof shows the invariant illustrated in Fig. 2:

$$\begin{array}{c}
 \text{R} \frac{*}{\frac{\omega \geq 0 \wedge d \geq 0 \vdash 2\omega^2 xy + 2y(-\omega^2 x - 2d\omega y) \leq 0}{\omega \geq 0 \wedge d \geq 0 \vdash (2\omega^2 xx' + 2yy' \leq 0)_{x' y'}^y -\omega^2 x - 2d\omega y}} \\
 \text{DI} \frac{\omega^2 x^2 + y^2 \leq c^2 \vdash [x' = y, y' = -\omega^2 x - 2d\omega y \ \& \ (\omega \geq 0 \wedge d \geq 0)] \omega^2 x^2 + y^2 \leq c^2}{\vdash \omega^2 x^2 + y^2 \leq c^2 \rightarrow [x' = y, y' = -\omega^2 x - 2d\omega y \ \& \ (\omega \geq 0 \wedge d \geq 0)] \omega^2 x^2 + y^2 \leq c^2}
 \end{array}$$

6. Disequational Differential Invariants

The case that is missing in differential invariant proof rules are for postconditions that are disequalities $\kappa \neq \eta$? How can they be proved?

Before you read on, see if you can find the answer for yourself.

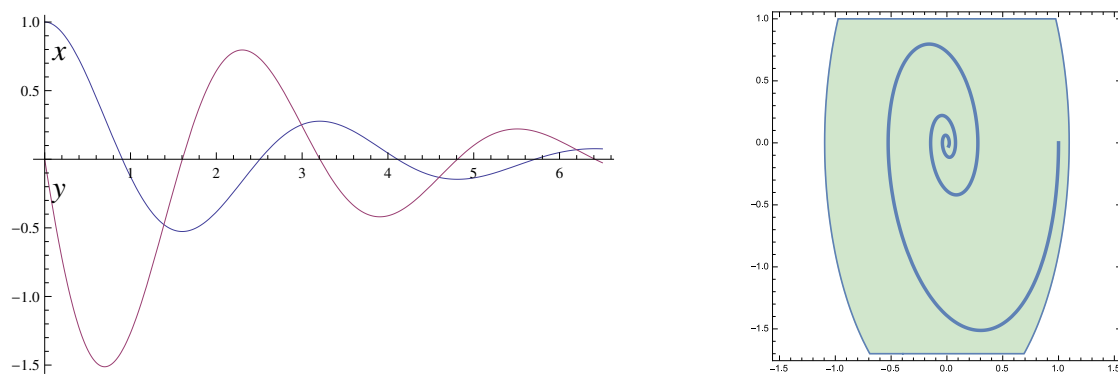


Figure 2: Damped oscillator time trajectory (**left**) and invariant in phase space (**right**)

By analogy to the previous cases, one might expect the following definition:

$$(\kappa \neq \eta)' \stackrel{?}{=} ((\kappa)' \neq (\eta)') \quad ???$$

It is crucial for soundness of differential invariants that $(\kappa \neq \eta)'$ is *not* defined that way! In the following counterexample, variable x can reach $x = 0$ without its derivative ever being 0; again, see Fig. 3 for the dynamics. Of course, just because κ and η start out

$$\frac{\frac{* \text{ (unsound)}}{\vdash 1 \neq 0}}{\textcolor{red}{\vdash} x \neq 5 \vdash [x' = 1]x \neq 5}$$

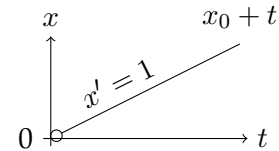


Figure 3: **a** Unsound attempt of using disequalities

3b: Linear dynamics

different, does not mean they would always stay different if they evolve with different derivatives. Au contraire, it is because both evolve with different derivatives that they might catch each other.

Instead, if κ and η start out differently and evolve with the same derivatives, they will always stay different. So the sound definition is slightly unexpected:

$$(\kappa \neq \eta)' \equiv ((\kappa)' = (\eta)')$$

7. Conjunctive Differential Invariants

The next case to consider is where the invariant that we want to prove is a conjunction $F \wedge G$. Lemma 8 takes care of how to handle differential substitution for the differential equations, if only we define the correct total derivative of $(F \wedge G)'$.

Before you read on, see if you can find the answer for yourself.

To show that a conjunction $F \wedge G$ is invariant it is perfectly sufficient to prove that both are invariant. This can be justified separately, but is more obvious when recalling the following equivalence from [Lecture 7](#):

$$([\wedge] \ [\alpha])(\phi \wedge \psi) \leftrightarrow [\alpha]\phi \wedge [\alpha]\psi$$

which is valid for all hybrid programs α , also when α is just a differential equation. Consequently, the total derivative of a conjunction is the conjunction of the total derivatives (i.e. $(\cdot)'$ is a homomorphism for \wedge):

$$(F \wedge G)' \equiv (F)' \wedge (G)'$$

Again, we will not develop a proper soundness argument, because it will follow from the general differential invariant proof rule.

With a corresponding proof rule that enables us to do the following proof:

$$\begin{array}{c} \text{R} \quad \frac{\begin{array}{c} * \\ \hline \vdash 2de + 2e(-d) \leq 0 \wedge 2de + 2e(-d) \geq 0 \\ \hline \vdash (2dd' + 2ee' \leq 0 \wedge 2dd' + 2ee' \geq 0)_{d' \ e'}^e \ -d \\ \hline \text{DI} \ d^2 + e^2 \leq r^2 \wedge d^2 + e^2 \geq r^2 \vdash [d' = e, e' = -d](d^2 + e^2 \leq r^2 \wedge d^2 + e^2 \geq r^2) \end{array}}{} \end{array}$$

Since the invariant $d^2 + e^2 \leq r^2 \wedge d^2 + e^2 \geq r^2$ is easily proved to be equivalent to $d^2 + e^2 = r^2$, the above proof gives yet another proof of (2) when combined with a corresponding use of the generalization rule [\[gen'\]](#).

8. Disjunctive Differential Invariants

The next case to consider is where the invariant that we want to prove is a disjunction $F \vee G$. Lemma 8 takes care of how to handle differential substitution for the differential equations, if only we define the correct total derivative of $(F \vee G)'$. How?

Before you read on, see if you can find the answer for yourself.

The total derivative of a conjunction is the conjunction of the total derivatives. So, by analogy, it might stand to reason to define the total derivative of a disjunction as the disjunction of the total derivatives.

$$(F \vee G)' \stackrel{?}{=} (F)' \vee (G)' \quad ???$$

Let's try it:

$$\begin{array}{c} \text{unsound} \\ \hline \mathbb{R} \quad \vdash 2de + 2e(-d) = 0 \vee 5d + re \geq 0 \\ \hline \vdash (2dd' + 2ee' = 0 \vee r'd + rd' \geq 0)_{d'}^e \quad \neg^d \\ \hline \textcolor{red}{\downarrow} d^2 + e^2 = r^2 \vee rd \geq 0 \vdash [d' = e, e' = -d, r' = 5](d^2 + e^2 = r^2 \vee rd \geq 0) \end{array}$$

That would be spectacularly wrong, however, because the formula at the bottom is not actually valid, so it does not deserve a proof. We have no business of proving formulas that are not valid and if we ever could, we would have found a serious unsoundness in the proof rules.

For soundness of differential induction, it is crucial that Def. 1 defines the total derivative $(F \vee G)'$ of a disjunction conjunctively as $(F)' \wedge (G)'$ instead of as $(F)' \vee (G)'$. From an initial state ν which satisfies $\nu \models F$, and hence $\nu \models F \vee G$, the formula $F \vee G$ only is sustained differentially if F itself is a differential invariant, not if G is. For instance, $d^2 + e^2 = r^2 \vee rd \geq 0$ is no invariant of the above differential equation, because $rd \geq 0$ will be invalidated if we just follow the circle dynamics long enough. So if the disjunction was true because $rd \geq 0$ was true in the beginning, it does not stay invariant.

In practice, splitting differential induction proofs over disjunctions by [V1](#) can be useful if a direct proof with a single common differential invariant does not succeed:

$$\begin{array}{c} \text{DI} \frac{\vdash A'_{x'}^\theta}{A \vdash [x' = \theta]A} \quad \textcolor{blue}{\text{vr}} \frac{\textcolor{blue}{ax} \frac{*}{A \vdash A, B}}{A \vdash A \vee B} \quad \textcolor{blue}{\text{gen}'} \frac{}{A \vdash [x' = \theta](A \vee B)} \\ \hline \textcolor{blue}{\text{V1}} \frac{}{A \vee B \vdash [x' = \theta](A \vee B)} \\ \hline \textcolor{blue}{\rightarrow r} \frac{}{\vdash A \vee B \rightarrow [x' = \theta](A \vee B)} \end{array}$$

9. Differential Invariants

Differential invariants are a general proof principles for proving invariants of formulas. Summarizing what this lecture has discovered so far leads to a single proof rule for differential invariants. That is why all previous proofs just indicated [DI](#) when using the various special cases of the differential invariant proof rule to be developed next.

All previous arguments remain valid when the differential equation has an evolution domain constraint H that it cannot leave by definition. In that case, the inductive proof step can even assume the evolution domain constraint to hold, because the system, by definition, is not allowed to leave it.

Definition 11 (Derivation). The operator $(\cdot)'$ that is defined as follows on first-order real-arithmetic formulas is called *syntactic (total) derivation*:

$$(F \wedge G)' \equiv (F)' \wedge (G)' \quad (4a)$$

$$(F \vee G)' \equiv (F)' \vee (G)' \quad (4b)$$

$$(\forall x F)' \equiv \forall x (F)' \quad (4c)$$

$$(\exists x F)' \equiv \exists x (F)' \quad (4d)$$

$$(a \geq b)' \equiv (a)' \geq (b)' \quad \text{accordingly for } <, >, \leq, =, \text{ but not } \neq \quad (4e)$$

Furthermore, $F'_{x'}^\theta$ is defined to be the result of substituting θ for x' in F' . The operation mapping F to $(F)'_{x'}^\theta$ is called *Lie-derivative* of F with respect to $x' = \theta$.

By (4e), the derivation $(F)'$ on formulas F uses the derivation $(a)'$ on terms a that occur within F .

That is, to replace the left-hand side of a differential equation by the right-hand side.

Lemma 12 (Differential invariants). *The differential invariant rule is sound:*

$$(DI) \frac{H \vdash F'_{x'}^\theta}{F \vdash [x' = \theta \ \& \ H] F} \quad (DI') \frac{\Gamma \vdash F, \Delta \quad H \vdash F'_{x'}^\theta \quad F \vdash \psi}{\Gamma \vdash [x' = \theta \ \& \ H] \psi, \Delta}$$

The version *DI'* can be derived easily from the more fundamental, essential form *DI* similar to how the most useful loop induction rule *ind'* derives from the essential form *ind*.^a

^aThe proof rule *DI'* is not implemented in KeYmaera, because a differential cut *DC* with F and a subsequent *DI* will lead to the same proof (Sect. 12).

The basic idea behind rule *DI* is that the premise of *DI* shows that the total derivative F' holds within evolution domain H when substituting the differential equations $x' = \theta$ into F' . If F holds initially (antecedent of conclusion), then F itself always stays true (succedent of conclusion). Intuitively, the premise gives a condition showing that, within H , the total derivative F' along the differential constraints is pointing inwards or transversally to F but never outwards to $\neg F$; see Fig. 4 for an illustration. Hence,

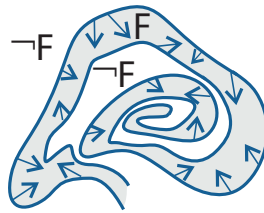


Figure 4: Differential invariant F for safety

if we start in F and, as indicated by F' , the local dynamics never points outside F ,

then the system always stays in F when following the dynamics. Observe that, unlike F' , the premise of **DI** is a well-formed formula, because all differential expressions are replaced by non-differential terms when forming $F'^\theta_{x'}$.

Proof. Assume the premise $F'^\theta_{x'} = 0$ to be valid, i.e. true in all states. In order to prove that the conclusion $F \vdash [x' = \theta]F$ is valid, consider any state ν . Assume that $\nu \models F$, as there is otherwise nothing to show (sequent is trivially *true* since antecedent evaluates to *false*). If $\zeta \in [0, r]$ is any time during any solution $\varphi : [0, r] \rightarrow \mathcal{S}$ of any duration $r \in \mathbb{R}$ of $x' = \theta$ beginning in initial state $\varphi(0) = \nu$, then it remains to be shown that $\varphi(r) \models F$. By antecedent, $\nu \models F$, in the initial state $\nu = \varphi(0)$.

If the duration of φ is $r = 0$, we have $\varphi(0) \models F$ immediately, because $\nu \models F$. For duration $r > 0$, we show that F holds all along φ , i.e., $\varphi(\zeta) \models F$ for all $\zeta \in [0, r]$.

We have to show that $\nu \models F \rightarrow [x' = \theta \ \& \ H]F$ for all states ν . Let ν satisfy $\nu \models F$ as, otherwise, there is nothing to show. We can assume F to be in disjunctive normal form and consider any disjunct G of F that is true at ν . In order to show that F remains true during the continuous evolution, it is sufficient to show that each conjunct of G is. We can assume these conjuncts to be of the form $\eta \geq 0$ (or $\eta > 0$ where the proof is accordingly). Finally, using vectorial notation, we write $x' = \theta$ for the differential equation system. Now let $\varphi : [0, r] \rightarrow (V \rightarrow \mathbb{R})$ be any solution of $x' = \theta \ \& \ H$ beginning in $\varphi(0) = \nu$. If the duration of φ is $r = 0$, we have $\varphi(0) \models \eta \geq 0$ immediately, because $\nu \models \eta \geq 0$. For duration $r > 0$, we show that $\eta \geq 0$ holds all along the solution φ , i.e., $\varphi(\zeta) \models \eta \geq 0$ for all $\zeta \in [0, r]$.

Suppose there was a $\zeta \in [0, r]$ with $\varphi(\zeta) \models \eta < 0$, which will lead to a contradiction. The function $h : [0, r] \rightarrow \mathbb{R}$ defined as $h(t) = \llbracket \eta \rrbracket_{\varphi(\zeta)}$ satisfies the relation $h(0) \geq 0 > h(\zeta)$, because $h(0) = \llbracket \eta \rrbracket_{\varphi(0)} = \llbracket \eta \rrbracket_\nu$ and $\nu \models \eta \geq 0$ by antecedent of the conclusion. By Lemma 3, h is continuous on $[0, r]$ and differentiable at every $\xi \in (0, r)$. By mean value theorem, there is a $\xi \in (0, \zeta)$ such that $\frac{dh(t)}{dt}(\xi) \cdot (\zeta - 0) = h(\zeta) - h(0) < 0$. In particular, since $\zeta \geq 0$, we can conclude that $\frac{dh(t)}{dt}(\xi) < 0$. Now Lemma 3 implies that $\frac{dh(t)}{dt}(\xi) = \llbracket (\eta)' \rrbracket_{\varphi(\xi)} < 0$. This, however, is a contradiction, because the premise implies that the formula $H \rightarrow (\eta \geq 0)'$ is true in all states along φ , including $\varphi(\xi) \models H \rightarrow (\eta \geq 0)'$. In particular, as φ is a solution for $x' = \theta \ \& \ H$, we know that $\varphi(\xi) \models H$ holds, and we have $\varphi(\xi) \models (\eta \geq 0)'$, which contradicts $\llbracket (\eta)' \rrbracket < 0$. \square

This proof rule enables us to easily prove (3) and all previous proofs as well:

$$\begin{array}{c}
 * \\
 \hline
 \mathbb{R} \quad \vdash 2de + 2e(-d) \leq 0 \\
 \hline
 \vdash (2dd' + 2ee' \leq 2rr')_{d' \ e' \ r'}^e \ -d \ -0 \\
 \hline
 \text{DI} \quad d^2 + e^2 \leq r^2 \vdash [d' = e, e' = -d]d^2 + e^2 \leq r^2 \\
 \hline
 \rightarrow r \quad \vdash d^2 + e^2 \leq r^2 \rightarrow [d' = e, e' = -d]d^2 + e^2 \leq r^2
 \end{array}$$

10. Example Proofs

Example 13 (Quartic dynamics). The following simple $\text{d}\mathcal{L}$ proof uses **DI** to prove an invariant of a quartic dynamics.

$$\begin{array}{c}
 * \\
 \hline
 \mathbb{R} \quad a \geq 0 \vdash 3x^2((x-3)^4 + a) \geq 0 \\
 \hline
 a \geq 0 \vdash (3x^2x' \geq 0)_{x'}^{(x-3)^4+a} \\
 \hline
 \text{DI} \quad x^3 \geq -1 \vdash [x' = (x-3)^4 + a \ \& \ a \geq 0] x^3 \geq -1
 \end{array}$$

Observe that rule **DI** directly makes the evolution domain constraint $a \geq 0$ available as an assumption in the premise, because the continuous evolution is never allowed to leave it.

Example 14. Consider the dynamics $x' = y, y' = -\omega^2 x - 2d\omega y$ of the damped oscillator with the undamped angular frequency ω and the damping ratio d . See Fig. 5 for one example of an evolution along this continuous dynamics. Figure 5 shows a trajectory

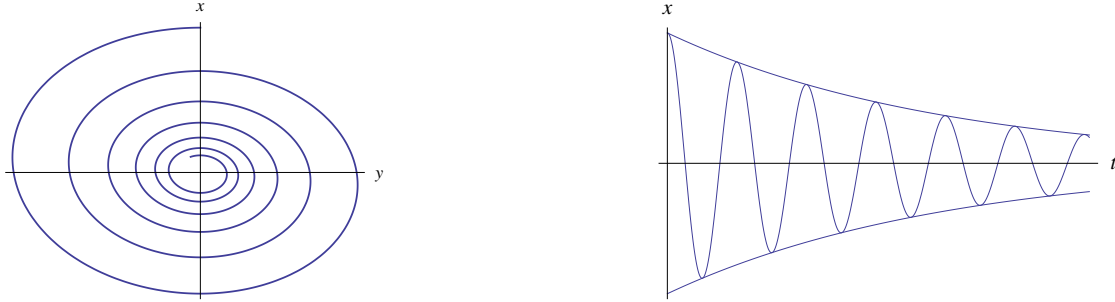


Figure 5: Trajectory and evolution of a damped oscillator

in the x, y space on the left, and an evolution of x over time t on the right. General symbolic solutions of symbolic initial-value problems for this differential equation can become surprisingly difficult. Mathematica, for instance, produces a long equation of exponentials that spans 6 lines of terms just for one solution. A differential invariant proof, instead, is very simple:

$$\begin{array}{c}
 * \\
 \hline
 \mathbb{R} \quad \omega \geq 0 \wedge d \geq 0 \vdash 2\omega^2xy - 2\omega^2xy - 4d\omega y^2 \leq 0 \\
 \hline
 \omega \geq 0 \wedge d \geq 0 \vdash (2\omega^2xx' + 2yy' \leq 0)_{x' y'}^{y \quad -\omega^2x - 2d\omega y} \\
 \hline
 \text{DI} \quad \omega^2x^2 + y^2 \leq c^2 \vdash [x' = y, y' = -\omega^2x - 2d\omega y \ \& \ (\omega \geq 0 \wedge d \geq 0)] \omega^2x^2 + y^2 \leq c^2
 \end{array}$$

Observe that rule **DI** directly makes the evolution domain constraint $\omega \geq 0 \wedge d \geq 0$ available as an assumption in the premise, because the continuous evolution is never allowed to leave it.

11. Assuming Invariants

Let's make the dynamics more interesting and see what happens. Suppose there is a robot at a point with coordinates (x, y) that is facing in direction (d, e) . Suppose the robot moves with constant (linear) velocity into direction (d, e) , which is rotating as before. Then the corresponding dynamics is:

$$x' = d, y' = e, d' = \omega e, e' = -\omega d$$

because the derivative of the x coordinate is the component d of the direction and the derivative of the y coordinate is the component e of the direction. If the rotation of the direction (d, e) is faster or slower, the differential equation would be formed correspondingly. Consider the following conjecture:

$$(x-1)^2 + (y-2)^2 \geq p^2 \rightarrow [x' = d, y' = e, d' = \omega e, e' = -\omega d](x-1)^2 + (y-2)^2 \geq p^2 \quad (5)$$

This conjecture expresses that the robot at position (x, y) will always stay at distance p from the point $(1, 2)$ if it started there. Let's try to prove conjecture (5):

$$\frac{\begin{array}{c} \vdash 2(x-1)d + 2(y-2)e \geq 0 \\ \vdash (2(x-1)x' + 2(y-2)y' \geq 0)_{x' y'}^{d e} \end{array}}{\text{DI} \frac{(x-1)^2 + (y-2)^2 \geq p^2 \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d](x-1)^2 + (y-2)^2 \geq p^2}{}}$$

Unfortunately, this differential invariant proof does not work. As a matter of fact, *fortunately* it does not work out, because conjecture (5) is not valid, so we will, fortunately, not be able to prove it with a sound proof technique. Conjecture (5) is too optimistic. Starting from some directions far far away, the robot will most certainly get too close to the point $(1, 2)$. Other directions may be fine.

Inspecting the above failed proof attempt, we could prove (5) if we knew something about the directions (d, e) that would make the remaining premise prove. What could that be?

Before you read on, see if you can find the answer for yourself.

Certainly, if we knew $d = e = 0$, the resulting premise would prove. Yet, that case is pretty boring because it corresponds to the point (x, y) being stuck forever. A more interesting case in which the premise would easily prove is if we knew $x - 1 = -e$ and $y - 2 = d$. In what sense could we “know” $x - 1 = -e \wedge y - 2 = d$? Certainly, we would have to assume this compatibility condition for directions versus position is true in the initial state, otherwise we would not necessarily know the condition holds true where we need it. So let’s modify (5) to include this assumption:

$$x - 1 = -e \wedge y - 2 = d \wedge (x - 1)^2 + (y - 2)^2 \geq p^2 \rightarrow \\ [x' = d, y' = e, d' = \omega e, e' = -\omega d](x - 1)^2 + (y - 2)^2 \geq p^2 \quad (6)$$

Yet, the place in the proof where we need to know $x - 1 = -e \wedge y - 2 = d$ for the above sequent prove to continue is in the middle of the inductive step. How could we make that happen?

Before you read on, see if you can find the answer for yourself.

One step in the right direction is to check whether $x - 1 = -e \wedge y - 2 = d$ is a differential invariant of the dynamics, so it stays true forever if it was true initially:

$$\begin{array}{c}
 \text{not valid} \\
 \hline
 \vdash d = -(-\omega d) \wedge e = \omega e \\
 \hline
 \vdash (x' = -e' \wedge y' = d') \overset{d}{x'} \overset{e}{y'} \overset{\omega e}{d'} \overset{-\omega d}{e'} \\
 \hline
 \text{DI } x - 1 = -e \wedge y - 2 = d \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d](x - 1 = -e \wedge y - 2 = d)
 \end{array}$$

This prove does not quite work out, because both sides of the equations are off by a factor of ω and, indeed, $x - 1 = -e \wedge y - 2 = d$ is not an invariant unless $\omega = 1$. On second thought, that makes sense, because the angular velocity ω determines how quickly the robot turns, so if there is any relation between position and direction, it should somehow depend on the angular velocity ω .

Let's refine the conjecture to incorporate the angular velocity on the side of the equation where it was missing in the above proof and consider $\omega(x - 1) = -e \wedge \omega(y - 2) = d$ instead. That knowledge would still help the proof of (5), just with the same extra factor on both terms. So let's modify (6) to use this assumption on the initial state:

$$\begin{aligned}
 \omega(x - 1) = -e \wedge \omega(y - 2) = d \wedge (x - 1)^2 + (y - 2)^2 \geq p^2 &\rightarrow \\
 [x' = d, y' = e, d' = \omega e, e' = -\omega d](x - 1)^2 + (y - 2)^2 \geq p^2 &\quad (7)
 \end{aligned}$$

A simple proof shows that the new addition $\omega(x - 1) = -e \wedge \omega(y - 2) = d$ is a differential invariant of the dynamics, so it holds always if it holds in the beginning:

$$\begin{array}{c}
 * \\
 \mathbb{R} \vdash \omega d = -(-\omega d) \wedge \omega e = \omega e \\
 \hline
 \vdash (\omega x' = -e' \wedge \omega y' = d') \overset{d}{x'} \overset{e}{y'} \overset{\omega e}{d'} \overset{-\omega d}{e'} \\
 \hline
 \text{DI } \omega(x - 1) = -e \wedge \omega(y - 2) = d \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d](\omega(x - 1) = -e \wedge \omega(y - 2) = d)
 \end{array}$$

Now, how can this freshly proved invariant $\omega(x - 1) = -e \wedge \omega(y - 2) = d$ be made available in the previous proof? Perhaps we could prove (7) using the conjunction of the invariant we want with the additional invariant we need:

$$(x - 1)^2 + (y - 2)^2 \geq p^2 \wedge \omega(x - 1) = -e \wedge \omega(y - 2) = d$$

That does not work (eliding the antecedent in the conclusion just for space reasons)

$$\begin{array}{c}
 \vdash 2(x - 1)d + 2(y - 2)e \geq 0 \wedge \omega d = -(-\omega d) \wedge \omega e = \omega e \\
 \hline
 \vdash (2(x - 1)x' + 2(y - 2)y' \geq 0 \wedge \omega x' = -e' \wedge \omega y' = d') \overset{d}{x'} \overset{e}{y'} \overset{\omega e}{d'} \overset{-\omega d}{e'} \\
 \hline
 \text{DI } (x - 1)^2 \dots \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d]((x - 1)^2 + (y - 2)^2 \geq p^2 \wedge \omega(x - 1) = -e \wedge \omega(y - 2) = d)
 \end{array}$$

because the right conjunct in the premise still proves beautifully but the left conjunct in the premise needs to know the invariant, which the differential invariant proof rule DI does not make the invariant F available in the antecedent of the premise.

In the case of loops, the invariant F can be assumed to hold before the loop body in the induction step:

$$(ind) \frac{F \vdash [\alpha]F}{F \vdash [\alpha^*]F}$$

By analogy, we could augment the differential invariant proof rule [DI](#) similarly to include F in the assumptions. Is that a good idea?

Before you read on, see if you can find the answer for yourself.

It looks tempting to suspect that rule [DI](#) could be improved by assuming the differential invariant F in the antecedent of the premise:

$$(DI_{??}) \frac{H \wedge F \vdash F'_{x'}}{F \vdash [x' = \theta \ \& \ H]F} \quad \text{sound?}$$

After all, we really only care about staying safe when we are still safe. And that would indeed easily prove the formula (7), which might make us cheer. But implicit properties of differential equations are a subtle business. Assuming F like in rule [DI_{??}](#) would, in fact, be *unsound*, as the following simple counterexample shows, which “proves” an invalid property using the unsound proof rule [DI_{??}](#):

$$\begin{array}{c} \text{(unsound)} \\ \hline d^2 - 2d + 1 = 0 \vdash 2de - 2e = 0 \\ \hline d^2 - 2d + 1 = 0 \vdash (2dd' - 2d' = 0)_{d' \ e'}^e \ -d \\ \hline \textcolor{red}{\vdash} d^2 - 2d + 1 = 0 \vdash [d' = e, e' = -d] d^2 - 2d + 1 = 0 \end{array}$$

Of course, $d^2 - 2d + 1 = 0$ does not stay true for the rotational dynamics, because d changes. And there are many other invalid properties that the unsound proof rule [DI_{??}](#) would claim to “prove”, for example:

$$\begin{array}{c} \text{(unsound)} \\ \hline \vdash -(x - y)^2 \geq 0 \rightarrow -2(x - y)(1 - y) \geq 0 \\ \hline \vdash -(x - y)^2 \geq 0 \rightarrow (-2(x - y)(x' - y') \geq 0)_{x' \ y'}^1 \ y \\ \hline \textcolor{red}{\vdash} -(x - y)^2 \geq 0 \vdash [x' = 1, y' = y] (-(x - y)^2 \geq 0) \end{array}$$

Assuming an invariant of a differential equation during its own proof is, thus, terribly incorrect, even though it has been suggested numerous times in the literature. There are some cases for which rule [DI_{??}](#) or variations of it would be sound, but these are nontrivial [[Pla10a](#), [Pla12d](#), [Pla12b](#), [GP14](#), [GSP14](#)].

The reason why assuming invariants for their own proof is problematic for the case of differential equations is somewhat subtle [[Pla10a](#), [Pla12d](#)]. In a nutshell, the proof rule [DI_{??}](#) assumes more than it knows, so that the argument becomes cyclic. The antecedent only provides the invariant in a single point and [Lecture 10](#) already explained that derivatives are not particularly well-defined in a single point.

12. Differential Cuts

Instead of these ill-guided attempts of assuming invariants for their own proof, there is a complementary proof rule for *differential cuts* [[Pla10a](#), [Pla08](#), [Pla12d](#), [Pla12b](#)] that can be used to strengthen assumptions about differential equations in a sound way:

$$(DC) \frac{\Gamma \vdash [x' = \theta \ \& \ H]C, \Delta \quad \Gamma \vdash [x' = \theta \ \& \ (H \wedge C)]F, \Delta}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta}$$

The differential cut rule works like a cut, but for differential equations. Recall the *cut* rule from [Lecture 6](#) which can be used to prove a formula C on the left premise and then assume it on the right premise:

$$(cut) \frac{\Gamma \vdash C, \Delta \quad \Gamma, C \vdash \Delta}{\Gamma \vdash \Delta}$$

Similarly, differential cut rule **DC** proves a property C of a differential equation in the left premise and then assumes C to hold in the right premise, except that it assumes C to hold during a differential equation by restricting the behavior of the system. In the right premise, rule **DC** restricts the system evolution to the subdomain $H \wedge C$ of H , which changes the system dynamics but is a pseudo-restriction, because the left premise proves that C is an invariant anyhow (e.g. using rule **DI**). Note that rule **DC** is special in that it changes the dynamics of the system (it adds a constraint to the system evolution domain region), but it is still sound, because this change does not reduce the reachable set, thanks to the left premise; see [Fig. 6](#)

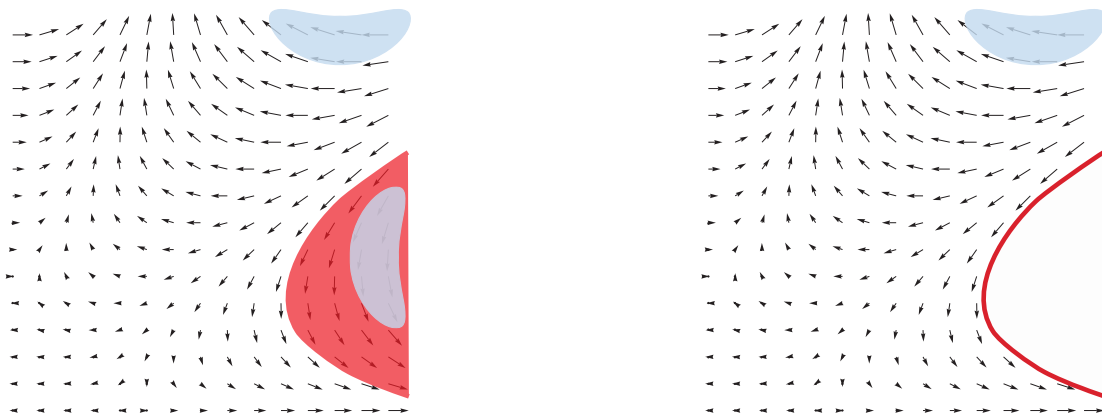


Figure 6: If the solution of the differential equation can never leave region C and enter the red region $\neg C$ (**left**), then this unreachable region $\neg C$ can be cut out of the state space without changing the dynamics of the system (**right**)

The benefit of rule **DC** is that C will (soundly) be available as an extra assumption for all subsequent **DI** uses on the right premise (see, e.g., the use of the evolution domain constraint in [Example 14](#)). In particular, the differential cut rule **DC** can be used to strengthen the right premise with more and more auxiliary differential invariants C that will be available as extra assumptions on the right premise, once they have been proven to be differential invariants in the left premise.

Proving the robot formula (7) in a sound way is now easy using a differential cut **DC** by $\omega(x - 1) = -e \wedge \omega(y - 2) = d$:

$$\begin{array}{c}
\text{R} \frac{*}{\vdash \omega d = -(-\omega d) \wedge \omega e = \omega e} \quad \text{R} \frac{*}{\omega(x-1)=-e \wedge \omega(y-2)=d \vdash 2(x-1)d + 2(y-2)e \geq 0} \\
\vdash (\omega x' = -e' \wedge \omega y' = d') \frac{d}{x'} \frac{e}{y'} \frac{\omega e}{d'} \frac{-\omega d}{e'} \\
\text{DI} \frac{\omega \dots \vdash [x'=d \dots](\omega(x-1)=-e \wedge \omega(y-2)=d)}{(x-1)^2 + (y-2)^2 \geq p^2 \vdash [x'=d, y'=e, d'=\omega e, e'=-\omega d \ \& \ \omega(x-1)=-e \wedge \omega(y-2)=d](x-1)^2 + (y-2)^2 \geq p^2} \\
\text{C} \frac{(x-1)^2 + (y-2)^2 \geq p^2, \omega(x-1)=-e \wedge \omega(y-2)=d \vdash [x'=d, y'=e, d'=\omega e, e'=-\omega d](x-1)^2 + (y-2)^2 \geq p^2}{(x-1)^2 + (y-2)^2 \geq p^2, \omega(x-1)=-e \wedge \omega(y-2)=d \vdash [x'=d, y'=e, d'=\omega e, e'=-\omega d](x-1)^2 + (y-2)^2 \geq p^2}
\end{array}$$

Amazing. Now we have a proper sound proof of the quite nontrivial robot motion property (7). And it even is a surprisingly short proof.

See [«Curved motion model»](#)

It is not always enough to just do a single differential cut. Sometimes, you may want to do a differential cut with a formula C , then use C on the right premise of [DC](#) to prove a second differential cut with a formula D and then on its right premise have $C \wedge D$ available to continue the proof; see Fig. 7. For example, we could also have gotten a proof of (7) by first doing a differential cut with $\omega(x-1) = -e$, then continue with a differential cut with $\omega(y-2) = d$, and then finally use both to prove the postcondition (Exercise 3). Using this differential cut process repeatedly has turned out to be extremely useful in practice and even simplifies the invariant search, because it leads to several simpler properties to find and prove instead of a single complex property [[PC08](#), [PC09](#), [Pla10b](#)].

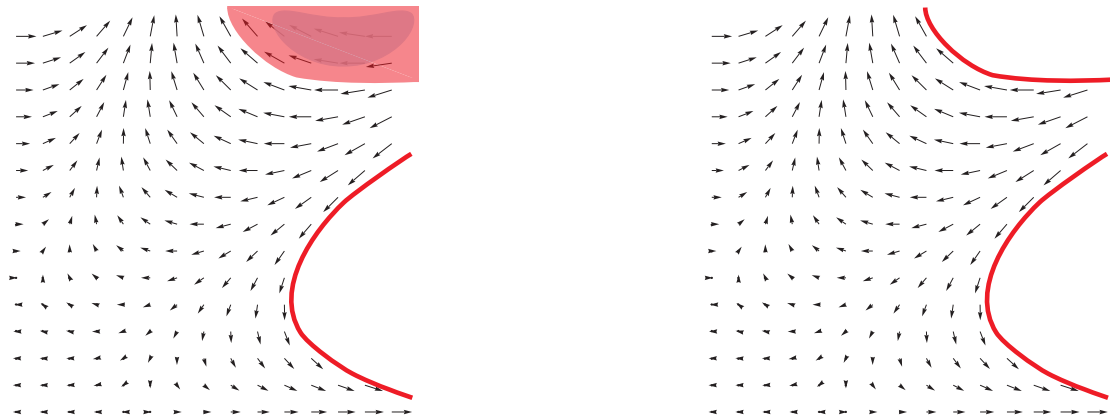


Figure 7: If the solution of the differential equation can never leave region D and enter the top red region $\neg D$ (**left**), then this unreachable region $\neg D$ can also be cut out of the state space without changing the dynamics of the system (**right**)

Proof of Soundness of [DC](#). For simplicity, consider only the case where $H \equiv \text{true}$ here. Rule [DC](#) is sound using the fact that the left premise implies that every solution φ that satisfies $x' = \theta$ also satisfies C *all along* the solution. Thus, if solution φ satisfies $x' = \theta$, it also satisfies $x' = \theta \ \& \ C$, so that the right premise entails the conclusion. The proof is accordingly for the case \square

See [«Tutorial Video on Differential Invariants, Differential Cuts»](#)

13. Differential Weakening

One simple but computable proof rule is *differential weakening*:

$$(DW) \frac{H \vdash F}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta}$$

This rule is obviously sound, because the system $x' = \theta \ \& \ H$, by definition, will stop before it leaves H , hence, if H implies F (i.e. the region H is contained in the region F), then F is an invariant, no matter what the actual differential equation $x' = \theta$ does. Unfortunately, this simple proof rule cannot prove very interesting properties, because it only works when H is very informative. It can, however, be useful in combination with stronger proof rules (e.g., differential cuts).

For example, after having performed the differential cut illustrated in Fig. 6 and, then, subsequently, performing the differential cut illustrated in Fig. 7, all unsafe blue regions have been cut out of the state space, so that the system in Fig. 7(right) is trivially safe by differential weakening, because there are no more unsafe blue regions. That is, the ultimate evolution domain constraint $H \wedge C \wedge D$ after the two differential cuts with C and with D trivially implies the safety condition F , i.e. $H \wedge C \wedge D \vdash F$ is valid. But notice that it took the two differential cuts to make differential weakening useful. The original evolution domain constraint H was not strong enough to imply safety, since there were still unsafe blue regions in the original system in Fig. 6(left) and even still in the intermediate system in Fig. 7(left) obtained after one differential cut with C .

14. Summary

This lecture introduced very powerful proof rules for differential invariants, with which you can prove even complicated properties of differential equations in easy ways. Just like in the case of loops, where the search for invariants is nontrivial, differential invariants also require some smarts (or good automatic procedures) to be found. Yet, once differential invariants have been identified, the proof follows easily.

Note 9 (Proof rules for differential equations). *The following are sound proof rules for differential equations:*

$$\begin{array}{ll} (DI) \frac{H \vdash F'_{x'}}{F \vdash [x' = \theta \ \& \ H]F} & (DW) \frac{H \vdash F}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta} \\ (DC) \frac{\Gamma \vdash [x' = \theta \ \& \ H]C, \Delta \quad \Gamma \vdash [x' = \theta \ \& \ (H \wedge C)]F, \Delta}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta} \end{array}$$

A. Proving Aerodynamic Bouncing Balls

This section studies a hybrid system with differential invariants. Remember the bouncing ball that was proved in [Lecture 7 on Loops & Invariants](#)?

The little acrophobic bouncing ball graduated from its study of loops and control and yearningly thinks back of its joyful time when it was studying continuous behavior. Caught up in nostalgia, the bouncing ball suddenly discovers that it unabashedly neglected the effect that air has on bouncing balls all the time. It sure is fun to fly through the air, so the little bouncing ball swiftly decides to make up for that oversight by including a proper aerodynamical model into its favorite differential equation. The effect that air has on the bouncing ball is air resistance and, it turns out, air resistance gets stronger the faster the ball is flying. After a couple of experiments, the little bouncing ball finds out that air resistance is quadratic in the velocity with an aerodynamic damping factor $r > 0$.

Now the strange thing with air is that air is always against the flying ball. Air always provides resistance, no matter which direction the ball is flying. If the ball is hurrying up, the air holds it back and slows it down by decreasing its positive speed $v > 0$. If the ball is rushing back down to the ground, the air still holds the ball back and slows it down, only then that actually means *increasing* the negative velocity $v < 0$, because that corresponds to decreasing the absolute value $|v|$. How could that be modeled properly?

One way of modeling this situation would be to use the (discontinuous) sign function $\text{sign } v$ that has value 1 for $v > 0$, value -1 for $v < 0$, and value 0 for $v = 0$:

$$x' = v, v' = -g - (\text{sign } v)rv^2 \ \& \ x \geq 0 \quad (8)$$

That, however, gives a differential equation with a difficult right-hand side. Instead, the little bouncing ball learned to appreciate the philosophy behind hybrid systems, which advocates for keeping the continuous dynamics simple and moving discontinuities and switching aspects to where they belong: the discrete dynamics. After all, switching and discontinuities is what the discrete dynamics is good at.

Consequently, the little bouncing ball decides to split modes and separate the upward flying part $v \geq 0$ from the downward flying part $v \leq 0$ and offer the system a nondeterministic choice between the two:¹

$$\begin{aligned} & x \leq H \wedge v = 0 \wedge x \geq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge r \geq 0 \rightarrow \\ & \left[\left(\text{if}(x = 0) \ v := -cv; \right. \right. \\ & \quad \left. \left(x' = v, v' = -g - rv^2 \ \& \ x \geq 0 \wedge v \geq 0 \cup x' = v, v' = -g + rv^2 \ \& \ x \geq 0 \wedge v \leq 0 \right) \right)^* \\ & \left. \right] \ (0 \leq x \leq H) \end{aligned} \quad (9)$$

In pleasant anticipation of the new behavior that this *aerodynamic bouncing ball* model provides, the little bouncing ball is eager to give it a try. Before daring to bounce around with this model, though, the acrophobic bouncing ball first wants to be convinced that it would be safe to use, i.e. the model actually satisfies the height limit property in (9). So the bouncing ball first sets out on a proof adventure. After writing down several ingenious proof steps, the bouncing ball finds out that its previous proof does not carry

¹Note that the reason for splitting modes and offering a nondeterministic choice in between are not controller events as they have been in [Lecture 8 on Events & Responses](#), but, rather, come from the physical model itself. The mechanism is the same, though, whatever the reason for splitting.

over. For one thing, the nonlinear differential equations can no longer be solved quite so easily. That makes the solution axiom [\[\]](#) rather useless. But, fortunately, the little bouncing ball brightens up again as it remembers that unsolvable differential equations was what differential invariants were good at. And the ball was rather keen on trying them in the wild, anyhow.

Yet, first things first. The first step of the proof after $\rightarrow r$ is the search for an invariant for the loop induction proof rule [ind'](#). Yet, since the proof of (9) cannot work by solving the differential equations, we will also need to identify differential invariants for the differential equations. If we are lucky, maybe the same invariant could even work for both? Whenever we are in such a situation, we can search from both ends and either identify an invariant for the loop first and then try to adapt it to the differential equation, or, instead, look for a differential invariant first.

Since we know the loop invariant for the ordinary bouncing ball from [Lecture 7](#), let's look at the loop first. The loop invariant for the ordinary bouncing ball was

$$2gx = 2gH - v^2 \wedge x \geq 0$$

We cannot really expect that invariant to work out for the aerodynamic ball (9) as well, because the whole point of the air resistance is that it slows the ball down. Since air resistance always works against the ball's motion, the height is expected to be less:

$$E_{x,v} \stackrel{\text{def}}{=} 2gx \leq 2gH - v^2 \wedge x \geq 0 \quad (10)$$

In order to check right away whether this invariant that we suspect to be a loop invariant works for the differential equations as well, the bouncing ball checks for differential invariance:

$$\begin{array}{c}
 * \\
 \hline
 \mathbb{R} \quad \frac{g > 0 \wedge r \geq 0, x \geq 0 \wedge v \geq 0 \vdash 2gv \leq 2gv + 2rv^3}{g > 0 \wedge r \geq 0, x \geq 0 \wedge v \geq 0 \vdash 2gv \leq -2v(-g - rv^2)} \\
 \hline
 \frac{g > 0 \wedge r \geq 0, x \geq 0 \wedge v \geq 0 \vdash (2gx' \leq -2vv')_{x' v'}^v -g - rv^2}{\text{DI} \quad g > 0 \wedge r \geq 0, 2gx \leq 2gH - v^2 \vdash [x' = v, v' = -g - rv^2 \ \& \ x \geq 0 \wedge v \geq 0] 2gx \leq 2gH - v^2}
 \end{array}$$

Note that for this proof to work, it is essential to keep the constants $g > 0 \wedge r \geq 0$ around, or at least $r \geq 0$. The easiest way of doing that is to perform a differential cut [DC](#) with $g > 0 \wedge r \geq 0$ and prove it to be a (trivial) differential invariant, because both parameters do not change, to make $g > 0 \wedge r \geq 0$ available in the evolution domain constraint for the rest of the proof.²

²Since this happens so frequently, KeYmaera implements a proof rule that, similar to the local version of loop invariants, keeps context assumptions around, which is fine as long as they are constant.

	*	
\mathbb{R}	$g > 0 \wedge r \geq 0, x \geq 0 \wedge v \leq 0 \vdash 2gv \leq 2gv - 2rv^3$	
	$g > 0 \wedge r \geq 0, x \geq 0 \wedge v \leq 0 \vdash 2gv \leq -2v(-g + rv^2)$	
	$g > 0 \wedge r \geq 0, x \geq 0 \wedge v \leq 0 \vdash (2gx' \leq -2vv')_{x', v'}^v \neg^{g+rv^2}$	
\mathbb{D}	$g > 0 \wedge r \geq 0, 2gx \leq 2gH - v^2 \vdash [x' = v, v' = -g + rv^2 \ \& \ x \geq 0 \wedge v \leq 0] 2gx \leq 2gH - v^2$	

Not exactly, because **DI** would lead to $(x' \geq 0)_{x'}^v \equiv v \geq 0$, which is obviously not always true for bouncing balls (except in the mode $x \geq 0 \wedge v \geq 0$). However, after proving the above differential invariants, a differential weakening argument by **DW** easily shows that the relevant part $x \geq 0$ of the evolution domain constraint always holds after the differential equation.

Now, what is left to do is a matter of proving (10) to be a loop invariant of (9).

$$A_{x,v} \stackrel{\text{def}}{=} x \leq H \wedge v = 0 \wedge x \geq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge r \geq 0$$

$$B_{x,v} \stackrel{\text{def}}{=} 0 \leq x \wedge x \leq H$$

$$(x'' \dots v \geq 0) \stackrel{\text{def}}{=} (x' = v, v' = -g - rv^2 \ \& \ x \geq 0 \wedge v \geq 0)$$

$$(x'' \dots v \leq 0) \stackrel{\text{def}}{=} (x' = v, v' = -g + rv^2 \ \& \ x \geq 0 \wedge v \leq 0)$$

$$E_{x,v} \stackrel{\text{def}}{=} 2gx \leq 2gH - v^2 \wedge x \geq 0$$

$$\frac{\frac{A_{x,v} \vdash E_{x,v} \quad \text{[} \vdash \text{]}}{A_{x,v} \vdash [\text{if}(x=0) v := -cv] E_{x,v}} \quad \frac{\frac{E_{x,v} \vdash [\text{if}(x=0) v := -cv] E_{x,v} \quad \frac{E_{x,v} \vdash [x'' \dots v \geq 0] E_{x,v} \quad E_{x,v} \vdash [x'' \dots v \leq 0] E_{x,v}}{E_{x,v} \vdash [x'' \dots v \geq 0] E_{x,v} \wedge [x'' \dots v \leq 0] E_{x,v}} \text{[} \wedge \text{]}}{E_{x,v} \vdash [x'' \dots v \geq 0 \cup x'' \dots v \leq 0] E_{x,v}} \text{[} \cup \text{]}} \quad \frac{E_{x,v} \vdash [\text{if}(x=0) v := -cv] E_{x,v} \quad \frac{E_{x,v} \vdash [\text{if}(x=0) v := -cv] [x'' \dots v \geq 0 \cup x'' \dots v \leq 0] E_{x,v}}{E_{x,v} \vdash [\text{if}(x=0) v := -cv; (x'' \dots v \geq 0 \cup x'' \dots v \leq 0)] E_{x,v}} \text{[} \text{gen}' \text{]}}{A_{x,v} \vdash [\text{if}(x=0) v := -cv; (x'' \dots v \geq 0 \cup x'' \dots v \leq 0)]^* B_{x,v}} \text{[} \vdash \text{]}} \quad E_{x,v} \vdash B_{x,v} \quad \text{[} \vdash \text{]}}{A_{x,v} \vdash [(\text{if}(x=0) v := -cv; (x'' \dots v \geq 0 \cup x'' \dots v \leq 0))^*] B_{x,v}} \text{[} \vdash \text{]}} \quad \text{ind}'$$

ANDRÉ PLATZER

worry about, which proves as follows:

$$\begin{array}{c}
 \frac{E_{x,v}, x = 0 \vdash E_{x,-cv}}{[:=]r \frac{E_{x,v}, x = 0 \vdash [v := -cv]E_{x,v}}{\rightarrow r \frac{E_{x,v} \vdash x = 0 \rightarrow [v := -cv]E_{x,v}}{[?]r \frac{E_{x,v} \vdash [?x = 0][v := -cv]E_{x,v}}{[i]r \frac{E_{x,v} \vdash [?x = 0; v := -cv]E_{x,v}}{\wedge r \frac{E_{x,v} \vdash [?x = 0; v := -cv]E_{x,v} \wedge [?x \neq 0]E_{x,v}}{[?]r \frac{E_{x,v} \vdash [?x = 0; v := -cv \cup ?x \neq 0]E_{x,v}}{E_{x,v} \vdash [\text{if}(x = 0) v := -cv]E_{x,v}}}}}} \\
 \frac{E_{x,v}, x \neq 0 \vdash E_{x,v}}{*} \quad \frac{E_{x,v} \vdash x \neq 0 \rightarrow E_{x,v}}{\rightarrow r} \quad \frac{E_{x,v} \vdash [?x \neq 0]E_{x,v}}{[?]r}
 \end{array}$$

This sequent proof first expands the $\text{if}()$, recalling that it is an abbreviation for a choice with tests. The right resulting premise proves trivially by axiom (there was no state change in the corresponding part of the execution), the left premise proves by arithmetic, because $2gH - v^2 \leq 2gH - (-cv)^2$ since $1 \geq c \geq 0$.

This completes the sequent proof for the safety of the aerodynamic bouncing ball expressed in dL formula (9). That is pretty neat!

See [«Aerodynamic bouncing ball model»](#)

It is about time for the newly upgraded aerodynamic acrophobic bouncing ball to notice a subtlety in its (provably safe) model. The bouncing ball had innocently split the differential equation (8) into two modes, one for $v \geq 0$ and one for $v \leq 0$ when developing the model (9). This seemingly innocuous step would have required more thought than the little bouncing ball had put in at the time. Of course, the single differential equation (8) could, in principle, switch between velocity $v \geq 0$ and $v \leq 0$ any arbitrary number of times during a single continuous evolution. The HP in (9) that split the mode, however, enforces that the ground controller $\text{if}(x = 0) v := -cv$ will run in between switching from the mode $v \geq 0$ to the mode $v \leq 0$ or back. On its way up when gravity is just about to win over and pull the ball back down again, that is of no consequence, because the trigger condition $x = 0$ will not be the case then anyhow, unless the ball really started the day without much energy ($x = v = 0$). On its way down, the condition will very well be true, namely when the ball is currently on the ground and just inverted its velocity. In that case, however, the evolution domain constraint $x \geq 0$ would have forced a ground controller action in the original system already anyhow.

So even if, in this particular model, the system could not in fact actually switch back and forth between the two modes too much in ways that would really matter, it is important to understand how to properly split modes in general, because that will be crucial for other systems. What the little bouncing ball should have done to become aerodynamical in a systematic way is to add an additional mini-loop around just the two differential equations, so that the system could switch modes without enforcing a discrete ground controller action to happen. This leads to the following dL formula

with a systematical mode split, which is provably safe just the same (Exercise 4):

$$\begin{aligned}
 & x \leq H \wedge v = 0 \wedge x \geq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge r \geq 0 \rightarrow \\
 & \left[\left(\text{if}(x = 0) v := -cv; \right. \right. \\
 & \quad \left. \left. (x' = v, v' = -g - rv^2 \ \& \ x \geq 0 \wedge v \geq 0 \cup x' = v, v' = -g + rv^2 \ \& \ x \geq 0 \wedge v \leq 0)^* \right)^* \right. \\
 & \left. \right] (0 \leq x \leq H)
 \end{aligned} \tag{11}$$

Exercises

Exercise 1. We have chosen to define

$$(\theta < \eta)' \equiv ((\theta)' < (\eta)')$$

Prove that the following slightly relaxed definition would also give a sound proof rule for differential invariants:

$$(\theta < \eta)' \equiv ((\theta)' \leq (\eta)')$$

Exercise 2. We have defined

$$(\theta \neq \eta)' \equiv ((\theta)' = (\eta)')$$

Suppose you remove this definition so that you can no longer use the differential invariant proof rule for formulas involving \neq . Can you derive a proof rule to prove such differential invariants regardless? If so, how? If not, why not?

Exercise 3. Prove dL formula(7) by first doing a differential cut with $\omega(x - 1) = -e$, then continue with a differential cut with $\omega(y - 2) = d$, and then finally use both to prove the original postcondition. Compare this proof to the proof in Sect. 12.

Exercise 4. The aerodynamic bouncing ball model silently imposed that no mode switching could happen without ground control being executed first. Even if that is not an issue for the bouncing ball, prove the more general formula (11) with its extra loop regardless. Compare the resulting proof to the sequent proof for (9).

Exercise 5. The least that the proof rules for differential equations get to assume is the evolution domain constraint H , because the system does not evolve outside it. Prove the following slightly stronger formulation of DI that assumes H to hold initially:

$$(\text{DI}) \frac{H \vdash F'_{x'}^\theta}{[?H]F \vdash [x' = \theta \ \& \ H]F}$$

Exercise 6. Prove the following definitions to be sound for the differential invariant proof rule:

$$\begin{aligned}
 \text{true}' &\equiv \text{true} \\
 \text{false}' &\equiv \text{true}
 \end{aligned}$$

Show how you can use those to prove the formula

$$A \rightarrow [x' = \theta \ \& \ H]B$$

in the case where $A \rightarrow \neg H$ is provable, i.e. where the system initially starts outside the evolution domain constraint H .

References

- [GP14] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. doi:[10.1007/978-3-642-54862-8_19](https://doi.org/10.1007/978-3-642-54862-8_19).
- [GSP14] Khalil Ghorbal, Andrew Sogokon, and André Platzer. Invariance of conjunctions of polynomial equalities for algebraic differential equations. In Markus Müller-Olm and Helmut Seidl, editors, *SAS*, volume 8723 of *LNCS*, pages 151–167. Springer, 2014. doi:[10.1007/978-3-319-10936-7_10](https://doi.org/10.1007/978-3-319-10936-7_10).
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. doi:[10.1007/978-3-540-70545-1_17](https://doi.org/10.1007/978-3-540-70545-1_17).
- [PC09] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. Special issue for selected papers from CAV’08. doi:[10.1007/s10703-009-0079-8](https://doi.org/10.1007/s10703-009-0079-8).
- [Pla08] André Platzer. *Differential Dynamic Logics: Automated Theorem Proving for Hybrid Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Dec 2008. Appeared with Springer.
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. doi:[10.1093/logcom/exn070](https://doi.org/10.1093/logcom/exn070).
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. A differential operator approach to equational differential invariants. In Lennart Beringer and Amy Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 28–48. Springer, 2012. doi:[10.1007/978-3-642-32347-8_3](https://doi.org/10.1007/978-3-642-32347-8_3).
- [Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:[10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).

- [Pla12d] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. [doi:10.2168/LMCS-8\(4:16\)2012](https://doi.org/10.2168/LMCS-8(4:16)2012).

Lecture Notes on Ghosts & Differential Ghosts

André Platzer

Carnegie Mellon University
Lecture 12

1. Introduction

Lecture 10 on [Differential Equations & Differential Invariants](#) and Lecture 11 on [Differential Equations & Proofs](#) equipped us with powerful tools for proving properties of differential equations without having to solve them. *Differential invariants* (DI) [Pla10a, Pla12] prove properties of differential equations by induction based on the right-hand side of the differential equation, rather than its much more complicated global solution. *Differential cuts* (DC) [Pla10a, Pla12] made it possible to simply prove another property C of a differential equation and then change the dynamics of the system around so that it is restricted to never leave region C . It can be shown that differential cuts are a fundamental proof principle for differential equations [Pla12], because some properties can only be proved with differential cuts. That is the *No Differential Cut Elimination* theorem, because, unlike cuts in first-order logic, differential cuts cannot generally be eliminated but are sometimes necessary [Pla12].

Yet, it can also be shown that there are properties for which even differential cuts are not enough, but differential ghosts become necessary [Pla12]. Differential ghosts [Pla12], spooky as they may sound, turn out to be a useful proof technique for differential equations. Differential ghosts or differential auxiliaries are extra variables that are introduced into the system solely for the purpose of the proof. Differential ghosts are the differential analogue of ghost variables or auxiliary variables, which sometimes have to be added into systems for the purpose of the proof. Both ghosts and differential ghosts serve a similar intuitive purpose: remember intermediate state values so that the relation of the values at intermediate states to values at final states can be analyzed. And that is also where the somewhat surprising name comes from. Auxiliary variables are often called ghosts, because they are not really present in the actual system, but just invented to make the story more interesting or, rather, the proof more

conclusive. Ghosts give the proof a way of referring to how the state used to be that is no more. There are many reasons for introducing ghost state into a system, which will be investigated subsequently.

This lecture is based on [Pla12, Pla10b].

The most important learning goals of this lecture are:

Modeling and Control: This lecture does not have much impact on modeling and control of CPS, because, after all, the whole point of ghosts and differential ghosts is that they are only added for the purposes of the proof. However, it can still sometimes be more efficient to add such ghost and differential ghost variables into the original model right away. It is good style to mark such additional variables in the model and controller as ghost variables in order to retain the information that they do not need to be included in the final system executable.

Computational Thinking: This lecture leverages computational thinking principles for the purposes of rigorous reasoning about CPS models by analyzing how extra dimensions can simplify or enable reasoning about lower-dimensional systems. From a state space perspective, extra dimensions are a bad idea, because, e.g., the number of points on a gridded space grows exponentially in the number of dimensions. From a reasoning perspective, the important insight of this lecture is that extra state variables sometimes help and may even make reasoning possible that is otherwise impossible. One intuition why extra ghost state may help reasoning is that it can be used to consume the energy that a given dissipative system is leaking (a similar purpose that dark matter had been speculated to exist) or produce the energy that a given system consumes. The addition of such extra ghost state then enables invariants of generalized energy constants involving both original and ghost state that was not possible using only the original state. That is, ghost state may now cause energy invariants. This lecture continues the trend of generalizing important logical phenomena from discrete systems to continuous systems. The verification techniques developed in this lecture are critical for verifying some CPS models of appropriate scale and technical complexity but are not necessary for all CPS models. A secondary goal of today's lecture is to develop more intuition and deeper understandings of differential invariants and differential cuts.

CPS Skills: The focus in this lecture is on reasoning about CPS models, but there is an indirect impact on developing better intuitions for operational effects in CPS by introducing the concept of relations of state to extra ghost state. A good grasp on such relations can help with the understanding of CPS dynamics quite substantially. The reason is that ghosts and differential ghosts enable extra invariants, which enable stronger statements about what we can rely on as a CPS evolves.

2. Recap

Recall the following proof rules of differential invariants (DI), differential weakening (DW) and differential cuts (DC) for differential equations from [Lecture 11 on Differential Equations & Proofs](#):

Note 1 (Proof rules for differential equations).

$$\begin{array}{ll}
 \text{(DI)} \frac{H \vdash F'_{x'}^\theta}{F \vdash [x' = \theta \ \& \ H]F} & \text{(DW)} \frac{H \vdash F}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta} \\
 \text{(DC)} \frac{\Gamma \vdash [x' = \theta \ \& \ H]C, \Delta \quad \Gamma \vdash [x' = \theta \ \& \ (H \wedge C)]F, \Delta}{\Gamma \vdash [x' = \theta \ \& \ H]F, \Delta}
 \end{array}$$

With cuts and generalizations, earlier lectures have also shown that the following can be proved:

$$\frac{A \vdash F \quad F \vdash [x' = \theta \ \& \ H]F \quad F \vdash B}{A \vdash [x' = \theta \ \& \ H]B} \quad (1)$$

This is useful for replacing a precondition A and postcondition B by another invariant F that implies postcondition B and is implied by precondition A .

3. Arithmetic Ghosts

The easiest way to see why it sometimes makes sense to add variables into a system model is to take a look at divisions. Divisions are not officially part of real arithmetic, because divisions can be defined. For example, when a division b/c is ever mentioned in a term such as $q = b/c$, then we can characterize q to remember the value of b/c by indirectly characterizing q in terms of b and c without $/$ and then subsequently use q wherever b/c first occurred:

$$q := \frac{b}{c} \rightsquigarrow q := *; ?qc = b \rightsquigarrow q := *; ?qc = b \wedge c \neq 0$$

where $q := *$ is the nondeterministic assignment that assigns an arbitrary real number to q . The first transformation (simply written \rightsquigarrow) characterizes $q = b/s$ indirectly by multiplying up as $qc = b$. The second transformation then conscientiously remembers that divisions only make all the sense in the world when we avoid dividing by zero. Because divisions by zero only cause a lot of trouble. This transformation can be used when b/c occurs in the middle of a term too:

$$x := 2 + \frac{b}{c} + e \rightsquigarrow q := *; ?qc = b; x := 2 + q + e \rightsquigarrow q := *; ?qc = b \wedge c \neq 0; x := 2 + q + e$$

Here q is called an *arithmetic ghost*, because q is an auxiliary variable that is only added to the hybrid program for the sake of defining the arithmetic quotient $\frac{b}{c}$.

4. Nondeterministic Assignments & Ghosts of Choice

The HP statement $x := *$ that has been used in Sect. 3 is a nondeterministic assignment that assigns an arbitrary real number to x . Comparing with the syntax of hybrid programs from [Lecture 3 on Choice & Control](#), however, it turns out that such a statement is not in the official language of hybrid programs.

$$\alpha, \beta ::= x := \theta \mid ?H \mid x' = \theta \& H \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \quad (2)$$

What now?

One possible solution, which is the one taken in the implementation of KeYmaera [PQ08], is to simply add the nondeterministic assignment $x := *$ as a statement to the syntax of hybrid programs.

$$\alpha, \beta ::= x := \theta \mid ?H \mid x' = \theta \& H \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid x := *$$

Consequently, nondeterministic assignments need a semantics to become meaningful:

$$7. \rho(x := *) = \{(\nu, \omega) : \omega = \nu \text{ except for the value of } x, \text{ which can be any real number}\}$$

And nondeterministic assignments need proof rules so that they can be handled in proofs:

$$\text{Note 2. } (\langle :* \rangle) \frac{\exists x \phi}{\langle x := * \rangle \phi} \quad ([*:]) \frac{\forall x \phi}{[x := *] \phi}$$

Proof rule $\langle :* \rangle$ says that there is one way of assigning an arbitrary value to x so that ϕ holds afterwards (i.e. $\langle x := * \rangle \phi$ holds) if (and only if) ϕ holds for some value of x (i.e. $\exists x \phi$ holds). And proof rule $[*:]$ says that ϕ holds for all ways of assigning an arbitrary value to x (i.e. $[x := *] \phi$ holds) if (and only if) ϕ holds for all values of x (i.e. $\forall x \phi$ holds) because x might have any such value after running $x := *$.

An alternative approach for adding nondeterministic assignments $x := *$ to hybrid programs is to reconsider whether we even have to add a new construct for $x := *$ or whether it can already be expressed in other ways. That is, to understand whether $x := *$ is truly a new program construct or whether it can be defined in terms of the other hybrid program statements from (2). Is $x := *$ definable by a hybrid program?

Before you read on, see if you can find the answer for yourself.

According to the proof rules $[:*]$ and $\langle :* \rangle$, nondeterministic assignments $x := *$ can be expressed equivalently by suitable quantifiers. But that does not help at all in the middle of a program, where we can hardly write down a quantifier to express that the value of x now changes.

There is another way, though. Nondeterministic assignment $x := *$ assigns any real number to x . One hybrid program that has the same effect of giving x any arbitrary real value [Pla10b, Chapter 3] is:

$$x := * \stackrel{\text{def}}{=} x' = 1 \cup x' = -1 \quad (3)$$

That is not the only definition of $x := *$, though. An equivalent definition is [Pla14]:

$$x := * \stackrel{\text{def}}{=} x' = 1; x' = -1$$

When working through the intended semantics of the left-hand side $x := *$ shown in case 7 above and the actual semantics of the right-hand side of (3) according to [Lecture 3 on Choice & Control](#), it becomes clear that both sides of (3) mean the same, because they have the same reachability relation. Hence, the above definition (3) captures the intended concept of giving x any arbitrary real value, nondeterministically. And, in particular, just like if-then-else, nondeterministic assignments do not really have to be added to the language of hybrid programs, because they can already be defined. Likewise, no proof rules would have to be added for nondeterministic assignments, because there are already proof rules for the constructs used in the right-hand side of the definition of $x := *$ in (3). Since the above proof rules $\langle :* \rangle, [:*]$ for $x := *$ are particularly easy, though, it is usually more efficient to include them directly, which is what KeYmaera does.

What may, at first sight, appear slightly spooky about (3), however, is that the left-hand side $x := *$ is clearly an instant change in time where x changes its value instantaneously to some arbitrary new real number. That is not quite the case for the right-hand side of (3), which involves two differential equations, which take time to follow.

The clue is that this passage of time is not observable in the state of the system. Consequently, the left-hand side of (3) really means the same as the right-hand side of (3). Remember from earlier lectures that time is not special. If a CPS wants to refer to time, it would have a clock variable t with the differential equation $t' = 1$. With such an addition, however, the passage of time t becomes observable in the value of variable t and, hence, a corresponding variation of the right-hand side of (3) would not be equivalent to $x := *$ at all (indicated by $\not\equiv$):

$$x := * \not\equiv x' = 1, t' = 1 \cup x' = -1, t' = 1$$

Both sides differ, because the right side exposes the amount of time t it took to get the value of x to where it should be, which, secretly, records information about the absolute value of the change that x underwent from its old to its new value. That change is something that the left-hand side $x := *$ knows nothing about.

5. Differential-algebraic Ghosts

The transformation in Sect. 3 can eliminate all divisions, not just in assignments, but also in tests and all other hybrid programs, with the notable exception of differential equations. Eliminating divisions in differential equations turns out to be a little more involved.

The following elimination using a (discrete) arithmetic ghost q is correct:

$$x' = \frac{2x}{c} \ \& \ c \neq 0 \ \wedge \ \frac{x+1}{c} > 0 \quad \rightsquigarrow \quad q := *; ?qc = 1; x' = 2xq \ \& \ c \neq 0 \ \wedge \ (x+1)q > 0$$

where the extra ghost variable q is supposed to remember the value of $\frac{1}{c}$.

The following attempt with a (discrete) arithmetic ghost q , however, would change the semantics rather radically:

$$x' = \frac{c}{2x} \ \& \ 2x \neq 0 \ \wedge \ \frac{c}{2x} > 0 \quad \rightsquigarrow \quad q := *; ?q2x = 1; x' = cq \ \& \ 2x \neq 0 \ \wedge \ cq > 0$$

because q then only remembers the inverse of the initial value of $2x$, not the inverse of the value of $2x$ as x evolves along the differential equation $x' = \frac{c}{2x}$. That is q has a constant value during the differential equation but, of course, the quotient $\frac{c}{2x}$ changes over time since x does.

One way to proceed is to figure out how the value of the quotient $q = \frac{1}{2x}$ changes over time as x changes by $x' = \frac{c}{2x}$. By deriving what q stands for, that results in

$$q' = \left(\frac{1}{2x} \right)' = \frac{-2x'}{4x^2} = \frac{-2 \frac{c}{2x}}{4x^2} = -\frac{c}{4x^3}$$

Alas, we go unlucky here, because that gives yet another division to keep track of.

The other and entirely systematic way to proceed is to lift nondeterministic assignments q to differential equations $q' = *$ with the intended semantics that q changes arbitrarily over time while following that nondeterministic differential equation:¹

$$q' = \frac{b}{c} \quad \rightsquigarrow \quad q' = * \ \& \ qc = b \quad \rightsquigarrow \quad q' = * \ \& \ qc = b \ \wedge \ c \neq 0$$

While it is slightly more complicated to give a semantics to $q' = *$, the idea behind the transformation is completely analogous to the case of discrete arithmetic ghosts:

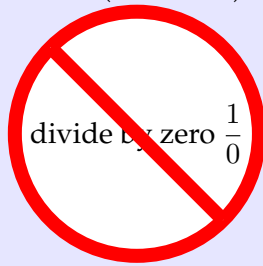
$$x' = 2 + \frac{b}{c} + e \quad \rightsquigarrow \quad x' = 2 + q + e, q' = * \ \& \ qc = b \quad \rightsquigarrow \quad x' = 2 + q + e, q' = * \ \& \ qc = b \ \wedge \ c \neq 0$$

Variable q is a *differential-algebraic ghost* in the sense of being an auxiliary variable in the differential-algebraic equation for the sake of defining the quotient $\frac{b}{c}$.

¹See [Pla10b, Chapter 3] for the precise meaning of the nondeterministic differential equation $q' = *$. It is the same as the differential-algebraic constraint $\exists d \ q' = d$, but differential-algebraic constraints have not been introduced in this course so far, either. The intuition of allowing arbitrary changes of the value of q over time is fine, though, for our purposes.

Together with the reduction of divisions in discrete assignments from Sect. 3, plus the insight that divisions in tests and evolution domain constraints can always be rewritten to division-free form, this gives a (rather sketchy) proof showing that hybrid programs and differential dynamic logic do not need divisions [Pla10b]. The advantage of eliminating divisions this way is that differential dynamic logic does not need special precautions for divisions and that the handling of zero divisors is made explicit in the way the divisions are eliminated from the formulas. In practice, however, it is still useful to use divisions, yet great care has to be exercised to make sure that no inadvertent divisions by zero could ever cause troublesome singularities.

Note 3 (Divisions).



Whenever dividing, exercise great care not to accidentally divide by zero, for that will cause quite some trouble. More often than not, this trouble corresponds to missing requirements in the system. For example $\frac{v^2}{2b}$ may be a good stopping distance when braking to a stop from initial velocity v , except when $b = 0$, which corresponds to having no brakes at all.

6. Discrete Ghosts

All the ghost variables so far were introduced to define operators such as divisions or nondeterministic assignments $x := *$. There are other reasons for using auxiliary alias ghost variables, though.

The discrete way of adding ghost variables is to introduce a new ghost variable y into a proof that remembers the value of a term θ . This can be useful in a proof in order to have a name, y , that recalls the value of θ later on in the proof, especially when the value of θ changes subsequently during the execution of hybrid programs α in the remaining modalities, which makes it possible to relate the value of θ before and after the run of that hybrid program α .

Lemma 1 (Discrete ghosts). *The following is a sound proof rule for introducing an auxiliary variable or (discrete) ghost y :*

$$(IA) \frac{\Gamma \vdash [y := \theta]\phi, \Delta}{\Gamma \vdash \phi, \Delta}$$

where y is a new program variable.

The fact that proof rule IA is sound can be explained easily based on the soundness of the substitution axiom $[:=]$ from [Lecture 5 on Dynamical Systems & Dynamic Axioms](#). The assignment axiom $[:=]$ proves validity of

$$\phi \leftrightarrow [y := \theta]\phi$$

because the fresh variable y does not occur in ϕ . Hence, discrete ghost rule [IA](#) just applies the substitution axiom backwards to introduce a ghost variable y that was not there before.

Discrete ghosts can be interesting when ϕ contains modalities that change variables in θ for y can then remember the value that θ had before that change. For example:

$$\frac{\text{IA} \frac{xy - 1 = 0 \vdash [c := xy][x' = x, y' = -y]xy = 1}{xy - 1 = 0 \vdash [x' = x, y' = -y]xy = 1}}{\rightarrow r \vdash xy - 1 = 0 \rightarrow [x' = x, y' = -y]xy = 1}$$

This sequent derivation memorizes the value that the interesting term xy had before the differential equation started in the ghost variable c . It is a bit hard to complete the proof, because substituting c away using the assignment rule [\[:=\]r](#) would undo the pleasant effect that the [IA](#) rule had, because the whole point of the fresh variable c is that it does not occur elsewhere.² So the only way the proof can make progress is by applying a proof rule to the differential equation. Unfortunately, the sequent calculus from [Lecture 6 on Truth & Proof](#) focuses the application of proof rules to the top-level of sequents. That is usually an advantage but now a disadvantage. For that reason, KeYmaera uses a concept called updates that postpone the application of the substitution rule [\[:=\]r](#) until all modalities are gone.

² This potentially surprising phenomenon happens in some form or other for other ghosts as well, because, the whole point of ghosts is to compute something that the original model and property do not depend on. So, sufficiently sophisticated forms of dead-code elimination would get rid of ghosts, which would be counterproductive for the proof.

Note 5 (Excursion: Updates). KeYmaera postpones the substitution resulting from an assignment according to rule $[:=]r,[:=]l,\langle := \rangle r,\langle := \rangle l$ if the postcondition is not a first-order formula but involves modalities with HPs. What this corresponds to is, essentially to leave the assignment as is and apply proof rules to the postcondition, but only in this particular case of a prefix of assignments! Because that would be a bit confusing without further notice, KeYmaera changes the notation slightly and turns an assignment into what it calls an update.

$$(R7) \frac{\{x := \theta\}\phi}{[x := \theta]\phi} \qquad (R8) \frac{\phi_x^\theta}{\{x := \theta\}\phi}$$

The meaning of the formula $\{x := \theta\}\phi$ in the premise of [R7](#) is exactly the same as the formula $[x := \theta]\phi$ in the conclusion of [R7](#). The notation $\{x := \theta\}\phi$ is only meant as a reminder for the user that KeYmaera decided to put the handling of the assignment by substitution on hold until the postcondition ϕ looks more civilized (meaning: first-order). KeYmaera collects all the state changes in such an update (or a list of updates). KeYmaera will then, essentially, just carry the update prefix $\{x := \theta\}$ around with it and apply the sequent proof rules directly to the respective postcondition ϕ after the update $\{x := \theta\}$ until the substitution that $\{x := \theta\}$ is waiting for can ultimately be applied ([R8](#)) which will make the update disappear again. Thus, KeYmaera splits the assignment rule $[:=]$ into two parts: the conversion of assignments to updates [R7](#) followed by the application of updates as substitutions [R8](#):

$$\frac{\Gamma \vdash \phi_x^\theta, \Delta}{\text{R8} \Gamma \vdash \{x := \theta\}\phi, \Delta} \\ \text{R7} \Gamma \vdash [x := \theta]\phi, \Delta$$

Similarly for $\langle x := \theta \rangle \phi$.

Observe that postponing the substitution of assignments in $[x := \theta]\phi$ may be necessary when the postcondition ϕ contains further modalities with loops assigning to x or differential equations binding x' . We also need to be careful to not leave updates lurking around for premises that need the sequent context Γ, Δ removed for soundness reasons, because both Γ, Δ and an update $\{x := \theta\}$ represent knowledge about the symbolic current state.

More information on updates can be found in [[Pla08](#), [Pla10b](#), Chapter 2.2,2.3,2.5].

Continuing the sequent proof above with updates to postpone the handling of the first assignment leads to the following proof:

$$\begin{array}{c} \text{R} \quad \frac{\begin{array}{c} * \\ \vdash 0 = xy + x(-y) \\ \vdash (0 = x'y + xy')_{x'y'}^{x-y} \end{array}}{\vdash 0 = xy + x(-y)} \\ \text{DI} \quad \frac{xy - 1 = 0 \vdash \{c := xy\}[x' = x, y' = -y]c = xy}{xy - 1 = 0 \vdash \{c := xy\}[x' = x, y' = -y]xy = 1} \triangleright \\ \text{[gen']} \quad \frac{xy - 1 = 0 \vdash \{c := xy\}[x' = x, y' = -y]xy = 1}{xy - 1 = 0 \vdash [c := xy][x' = x, y' = -y]xy = 1} \\ \text{R7} \quad \frac{xy - 1 = 0 \vdash [c := xy][x' = x, y' = -y]xy = 1}{xy - 1 = 0 \vdash [x' = x, y' = -y]xy = 1} \\ \text{IA} \quad \frac{xy - 1 = 0 \vdash [x' = x, y' = -y]xy = 1}{\vdash xy - 1 = 0 \rightarrow [x' = x, y' = -y]xy = 1} \\ \rightarrow r \end{array}$$

The generalization step $\llbracket gen' \rrbracket$ leads to a second premise that has been elided (marked by \triangleright) and proves, because $c = 1$ is an easily provable additional differential invariant, because the discrete ghost c starts out as 1 initially by the antecedent and never changes its value. This particular property also proves directly quite easily, but the proof technique of discrete ghosts is of more general interest.

See [«Proof using discrete ghosts»](#)

7. Proving Bouncing Balls with Sneaky Solutions

Recall a $d\mathcal{L}$ formula for the falling ball part of the bouncing ball proof from [Lecture 7 on Control Loops & Invariants](#), which was based on an argument in [Lecture 4](#):

$$2gx = 2gH - v^2 \wedge x \geq 0 \rightarrow [x' = v, v' = -g \wedge x \geq 0](2gx = 2gH - v^2 \wedge x \geq 0) \quad (4)$$

Recall the abbreviation

$$(x'' = -g \wedge x \geq 0) \equiv (x' = v, v' = -g \wedge x \geq 0)$$

[Lecture 7](#) proved $d\mathcal{L}$ formula (4) using the solutions of the differential equation with the solution proof rule $\llbracket \cdot \rrbracket$. Yet, $d\mathcal{L}$ formula (4) can also be proved with a mix of differential invariants, differential cuts and differential weakening, instead:

$$\text{DC} \frac{\text{DI} \frac{\mathbb{R} \frac{x \geq 0 \vdash 2gv = -2v(-g)}{x \geq 0 \vdash (2gx' = -2vv')_{x', v'}^{v, -g}}}{2gx = 2gH - v^2 \vdash [x'' = -g \wedge x \geq 0] 2gx = 2gH - v^2}}{2gx = 2gH - v^2 \vdash [x'' = -g \wedge x \geq 0](2gx = 2gH - v^2 \wedge x \geq 0)} \quad \text{DW} \frac{\text{ax} \frac{x \geq 0 \wedge 2gx = 2gH - v^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0}{2gx = 2gH - v^2 \vdash [x'' = -g \wedge x \geq 0 \wedge 2gx = 2gH - v^2](2gx = 2gH - v^2 \wedge x \geq 0)}}{2gx = 2gH - v^2 \vdash [x'' = -g \wedge x \geq 0](2gx = 2gH - v^2 \wedge x \geq 0)}$$

Note that differential weakening (DW) works for proving the postcondition $x \geq 0$, but DI would not work for proving $x \geq 0$, because its derivative is $(x \geq 0)' \equiv v \geq 0$, which is not an invariant of the bouncing ball since its velocity ultimately becomes negative when it is falling again according to gravity.

The above proof is very elegant and has notably easier arithmetic than the arithmetic requires when working with solutions of the bouncing ball in earlier lectures.

Note 6 (Differential invariants lower degrees). *Differential invariants DI work by differentiation, which lowers polynomial degrees. The solution proof rule $\llbracket \cdot \rrbracket$ works with solutions, which ultimately integrate the differential equation and, thus, increase the degrees. The computational complexity of the resulting arithmetic is, thus, often in favor of differential invariants even in cases where the differential equations can be solved so that the solution rule $\llbracket \cdot \rrbracket$ would be applicable.*

Besides the favorably simple arithmetic coming from differential invariants, the other reason why the proof worked so elegantly is that the invariant $2gx = 2gH - v^2 \wedge x \geq 0$ was a clever choice that we came up with in a creative way in [Lecture 4](#). There is nothing wrong with being creative. On the contrary!

But it also pays off to be systematic and develop a rich toolbox of techniques for proving properties of differential equations. Is there a way to prove (4) without such a distinctively clever invariant that works as a differential invariant right away? Yes, of course, because (4) can even be proved using solutions [7]. But it turns out that interesting things happen when we systematically try to understand how to make a proof happen that does not use the solution rule [7] and, yet, still uses solution-based arguments. Can you conceive a way to use solutions for differential equations without invoking the actual solution rule [7]?

Before you read on, see if you can find the answer for yourself.

8. Exploiting Differential Ghosts for Falling Balls

Note 7 (Ghost solutions). *Whenever there is a solution of a differential equation that we would like to make available to a proof without using the solution rule [']r, a differential cut and subsequent differential invariant can be used to cut the solution as an invariant into the system. The tricky part is that solutions depend on time, and time may not be part of the differential equation system. If there is no time variable, however, a differential ghost first needs to be added that pretends to be time.*

Consider dL formula (4) again, which turns into

$$A_{x,v} \vdash [x'' = -g \ \& \ x \geq 0] B_{x,v} \quad (4)$$

using the abbreviations:

$$\begin{aligned} A_{x,v} &\stackrel{\text{def}}{=} 2gx = 2gH - v^2 \wedge x \geq 0 \\ B_{x,v} &\stackrel{\text{def}}{=} 2gx = 2gH - v^2 \wedge x \geq 0 \\ (x'' = -g) &\stackrel{\text{def}}{=} (x' = v, v' = -g) \end{aligned}$$

The proof begins by introducing a discrete ghost v_0 remembering the initial velocity of the bouncing ball and proceeds by adding a differential ghost t for the time variable with derivative $t' = 1$ so that the solution $v = v_0 - tg$ can be differentially cut into the system and proved to be differentially invariant:

$$\begin{array}{c} \text{R} \\ \hline x \geq 0 \vdash -g = -1g \\ \hline x \geq 0 \vdash (v' = -t'g)_{v'}^{-g} \frac{1}{t'} \\ \text{DI} \frac{}{A_{x,v} \vdash \{v_0 := v\} [x'' = -g, t' = 1 \ \& \ x \geq 0] v = v_0 - tg} \quad \frac{}{A_{x,v} \vdash \{v_0 := v\} [x'' = -g, t' = 1 \ \& \ x \geq 0 \wedge v = v_0 - tg] B_{x,v}} \\ \text{DC} \frac{}{A_{x,v} \vdash \{v_0 := v\} [x'' = -g, t' = 1 \ \& \ x \geq 0] B_{x,v}} \\ \text{DA} \frac{}{A_{x,v} \vdash \{v_0 := v\} [x'' = -g \ \& \ x \geq 0] B_{x,v}} \\ \text{R7} \frac{}{A_{x,v} \vdash [v_0 := v] [x'' = -g \ \& \ x \geq 0] B_{x,v}} \\ \text{IA} \frac{}{A_{x,v} \vdash [x'' = -g \ \& \ x \geq 0] B_{x,v}} \end{array}$$

where the proof step marked **DA** (for differential auxiliaries or differential ghosts) introduces new variable t with derivative 1 as a differential ghost into the system.³ Observe how the differential invariant step **DI** made the sequent context as well as the update $\{v_0 := v\}$ disappear, which is generally important for soundness.

The left premise in the above proof has been closed by arithmetic. The right premise in the above proof proves as follows by first introducing yet another discrete ghost x_0

³ When discussing the differential ghost proof rule **DA** in a more general form later on, we will see that **DA** introduces an extra left premise, which is omitted in this proof (marked by \triangleleft). That additional premise, however, proves easily because $B_{x,v} \leftrightarrow \exists t B_{x,v}$ is rather trivially valid in first-order logic, as the fresh variable t does not even occur in $B_{x,v}$ at all here (vacuous quantification).

with **DA** that remembers the initial position so that it can be referred to in the solution. The solution $x = x_0 + v_0 t - \frac{g}{2} t^2$ can then be differentially cut into the system by **DC** and proved to be differentially invariant by **DI**:

$$\begin{array}{c}
 * \\
 \hline
 \text{ax} \frac{x \geq 0 \wedge v = v_0 - tg \vdash v = v_0 - 2\frac{g}{2}t}{x \geq 0 \wedge v = v_0 - tg \vdash (x' = v_0 t' - 2\frac{g}{2} t t') \frac{v}{x'} \frac{1}{t'}} \\
 \hline
 \text{DI} \frac{A_{x,v} \vdash \{x_0 := x; v_0 := v\} [x'' = -g, t' = 1 \& x \geq 0 \wedge v = v_0 - tg] x = x_0 + v_0 t - \frac{g}{2} t^2 \triangleright}{A_{x,v} \vdash \{x_0 := x; v_0 := v\} [x'' = -g, t' = 1 \& x \geq 0 \wedge v = v_0 - tg] B_{x,v}} \\
 \hline
 \text{DC} \frac{A_{x,v} \vdash \{x_0 := x; v_0 := v\} [x'' = -g, t' = 1 \& x \geq 0 \wedge v = v_0 - tg] B_{x,v}}{A_{x,v} \vdash \{v_0 := v\} [x'' = -g, t' = 1 \& x \geq 0 \wedge v = v_0 - tg] B_{x,v}} \\
 \hline
 \text{IA}
 \end{array}$$

The differential cut proof step marked **DC** has a second premise using the cut which is elided above (marked by \triangleright) and proves as follows:

$$\text{DW} \frac{x \geq 0 \wedge v = v_0 - tg \wedge x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash B_{x,v}}{A_{x,v} \vdash \{x_0 := x; v_0 := v\} [x'' = -g, t' = 1 \& x \geq 0 \wedge v = v_0 - tg \wedge x = x_0 + v_0 t - \frac{g}{2} t^2] B_{x,v}}$$

The resulting arithmetic can be proved by real arithmetic with enough care, but it has a twist! First of all, the arithmetic can be simplified substantially using the equality substitution rule **=r** from **Lecture 6** to replace v by $v_0 - tg$ and replace x by $x_0 + v_0 t - \frac{g}{2} t^2$ and use subsequent weakening (**WI**) to get rid of both equations after use. This simplification reduces the computational complexity of real arithmetic a lot:

$$\begin{array}{c}
 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2 \quad * \\
 \text{WI} \frac{x \geq 0 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2}{x \geq 0 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2 \wedge x \geq 0} \quad \text{ax} \frac{x \geq 0 \vdash x \geq 0}{x \geq 0 \vdash x \geq 0} \\
 \hline
 \text{Ar} \frac{x \geq 0 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2 \wedge x \geq 0}{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2 \wedge x \geq 0} \\
 \hline
 \text{WI} \frac{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2g(x_0 + v_0 t - \frac{g}{2} t^2) = 2gH - (v_0 - tg)^2 \wedge x \geq 0}{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - (v_0 - tg)^2 \wedge x \geq 0} \\
 \hline
 \text{=r} \frac{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - (v_0 - tg)^2 \wedge x \geq 0}{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0} \\
 \hline
 \text{=r} \frac{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0}{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0} \\
 \hline
 \text{AI} \frac{x \geq 0, v = v_0 - tg, x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0}{x \geq 0 \wedge v = v_0 - tg \wedge x = x_0 + v_0 t - \frac{g}{2} t^2 \vdash 2gx = 2gH - v^2 \wedge x \geq 0}
 \end{array}$$

Observe how this use of equality substitution and weakening helped simplify the arithmetic complexity of the formula substantially and even helped to eliminate a variable (v) right away. This can be useful to simplify arithmetic in many other cases as well. Both eliminating variables as well as applying and hiding equations right away can often simplify the complexity of handling real arithmetic. The arithmetic in the remaining left branch

$$2g \left(x_0 + v_0 t - \frac{g}{2} t^2 \right) = 2gH - (v_0 - tg)^2$$

expands by polynomial arithmetic and cancels as follows:

$$2g \left(x_0 + \cancel{v_0 t} - \frac{g}{2} \cancel{t^2} \right) = 2gH - v_0^2 + \cancel{2v_0 t g} + \cancel{t^2 g^2}$$

Those cancellations simplify the arithmetic, leaving the remaining condition:

$$2gx_0 = 2gH - v_0^2 \quad (5)$$

Indeed, this relation characterizes exactly how H , which turns out to have been the maximal height, relates to the initial height x_0 and initial velocity v_0 . In the case of initial velocity $v_0 = 0$, for example, the equation (5) collapses to $x_0 = H$, i.e. that H is the initial height in that case. Consequently, the computationally easiest way of proving the resulting arithmetic is to first prove by a differential cut **DC** that (5) is a trivial differential invariant, resulting in a proof of (4); see Exercise 2.

Yet, as we go through all proof branches again to check that we really have a proof, however, we notice a subtle but blatant oversight. Can you spot it, too?

The very first left-most branch with the initial condition for the differential invariant $v = v_0 = tg$ does not, actually, prove. The catch is that we silently assumed $t = 0$ to be the initial value for the new clock t , but that our proof did not actually say so. Oh my, what could be done about that now?

Before you read on, see if you can find the answer for yourself.

Discrete ghosts to the rescue! Even though we do not know the initial value of the differential ghost t , we can simply use a discrete ghost again to call it t_0 and get on with it. Will that work? Can you find it out? Or should we start a revision of the proof to find out?

$$\begin{array}{c}
 A_{x,v} \vdash \{v_0 := v\} \{t_0 := t\} [x'' = -g, t' = 1 \ \& \ x \geq 0] v = v_0 - (t - t_0)g \quad A_{x,v} \vdash \{v_0 := v\} \{t_0 := t\} [x'' = -g, t' = 1 \ \& \ x \geq 0 \wedge v = v_0 - (t - t_0)g] B_{x,v} \\
 \hline
 \text{DC} \quad A_{x,v} \vdash \{v_0 := v\} \{t_0 := t\} [x'' = -g, t' = 1 \ \& \ x \geq 0] B_{x,v} \\
 \hline
 \text{R}' \quad A_{x,v} \vdash \{v_0 := v\} \{t_0 := t\} [x'' = -g, t' = 1 \ \& \ x \geq 0] B_{x,v} \\
 \hline
 \text{IA} \quad A_{x,v} \vdash \{v_0 := v\} [x'' = -g, t' = 1 \ \& \ x \geq 0] B_{x,v}
 \end{array}$$

As this proof shows, everything works as expected as long as we realize that this requires a change of the invariants used for the differential cuts. The solution of the velocity to differentially cut in will be $v = v_0 - (t - t_0)g$ and the solution of the position to differentially cut in subsequently will be $x = x_0 + v_0(t - t_0) - \frac{g}{2}(t - t_0)^2$.

See [«Proof of falling balls»](#)

For the case of the bouncing ball, this proof was unnecessarily complicated, because the solution rule [\['\]](#) could have been used instead right away, instead. Yet, even if this particular proof was more involved, the arithmetic ended up being nearly trivial in the end (which [Note 6](#) already observed to hold in general for differential invariant proofs). But the same proof technique of adding differential ghosts and discrete ghosts as needed can be pretty useful in more complicated systems.

Note 8 (On the utility of ghosts). *Adding differential ghosts and discrete ghosts as needed can be useful in more complicated systems that do not have computable solutions, but in which other relations between initial (or intermediate) and final state can be proved. The same technique can also be useful for cutting in solutions when only part of a differential equation system admits a polynomial solution.*

For example, the differential equation system $d' = \omega e, e' = -\omega d, v' = a$ is difficult, because it has non-polynomial solutions. Still, one part of this differential equation, the velocity $v' = a$, is easily solved. Yet, the solution rule [\['\]r](#) is not applicable, because no real arithmetic solution of the whole differential equation system exists (except when $\omega = 0$). Regardless, after suitable discrete ghosts and differential ghosts for adding a clock $t' = 1$, a differential cut with the solution $v = v_0 + at$ of $v' = a$ adds this precise knowledge about the time-dependent change of the variable v to the evolution domain for subsequent use in the proof.

9. Differential Ghosts

The proof technique of differential ghosts is not limited to adding the differential equation $t' = 1$ for time, but can add other differential equations $y' = \eta$ into the differential equation system as well. Besides, the invariant to prove can very well be modified to make use of the additional ghost variable y by referring to it, which did not happen in the above proof, in which the postcondition $B_{x,v}$ remained unchanged (see [3](#)).

Lemma 2 (Differential ghosts). *The following is a sound proof rule differential auxiliaries (DA) for introducing auxiliary differential variables or differential ghosts [Pla12]:*

$$(DA) \frac{\Gamma \vdash F \leftrightarrow \exists y G, \Delta \quad \Gamma, G \vdash [x' = \theta, y' = \eta \& H]G, \Delta}{\Gamma, F \vdash [x' = \theta \& H]F, \Delta}$$

where y new and $y' = \eta, y(0) = y_0$ has a global solution y on H for each y_0 .

Rule DA is applicable if y is a new variable and the new differential equation $y' = \eta$ has global solutions on H (e.g., because term η satisfies a Lipschitz condition [Wal98, Proposition 10.VII], which is definable in first-order real arithmetic and thus decidable). Without that condition, adding $y' = \eta$ could limit the duration of system evolutions incorrectly. In fact, it would be sufficient for the domains of definition of the solutions of $y' = \eta$ to be no shorter than those of x . Soundness is easy to see, because precondition F implies G for some choice of y (left premise). Yet, for any y , G is an invariant of the extended dynamics (right premise). Thus, G always holds after the evolution for some y (its value can be different than in the initial state), which still implies F (left premise). Since y is fresh and its differential equation does not limit the duration of solutions of x on H , this implies the conclusion. Since y is fresh, y does not occur in H , and, thus, its solution does not leave H , which would incorrectly restrict the duration of the evolution as well.


Intuitively, rule DA can help proving properties, because it may be easier to characterize how x changes in relation to an auxiliary differential ghost variable y with a suitable differential equation ($y' = \eta$) compared to understanding the change of x in isolation.

10. Spooky Ghosts


In fact, differential ghosts even give us a, shockingly spooky, way of generating differential equations for differential ghosts on the fly as needed for proofs to work out. That might sound scary but is amazingly useful. To see how it works, invent your own differential ghost $y' = \text{cloud}$ with a still-unspecified right-hand side cloud , which is nothing but a common spooky cloud, and just keep “proving” as if nothing had happened:

$$\begin{array}{c} \text{could prove if } \text{cloud} = \frac{y}{2} \\ \hline \vdash -xy^2 + 2xy\text{cloud} = 0 \\ \hline \vdash (x'y^2 + x2yy' = 0)_{x' \ y'}^{-x \ \text{cloud}} \\ \hline \mathbb{R} \vdash x > 0 \leftrightarrow \exists y xy^2 = 1 \quad \text{DI } xy^2 = 1 \vdash [x' = -x, y' = \text{cloud}]xy^2 = 1 \\ \hline \text{DA} \quad x > 0 \vdash [x' = -x]x > 0 \end{array}$$

The right premise could prove if only cloud were chosen to be $\frac{y}{2}$, in which case the premise $-xy^2 + 2xy\text{cloud} = 0$ is quite easily proved. That, of course, was a bit too

spooky for the soundness-loving truth-connoisseur. So let's instantiate the spooky cloud  with its concrete choice $\frac{y}{2}$ and start all over with a proper proof:

$$\begin{array}{c}
 \text{DA} \quad \frac{\text{IR} \vdash x > 0 \leftrightarrow \exists y \, xy^2 = 1 \quad \frac{\text{DI} \, xy^2 = 1 \vdash [x' = -x, y' = \frac{y}{2}] xy^2 = 1}{\vdash (x'y^2 + x2yy' = 0)_{x' \frac{y}{2}}^x} \quad \frac{\vdash -xy^2 + 2xy \frac{y}{2} = 0}{\vdash (x'y^2 + x2yy' = 0)_{x' \frac{y}{2}}^x}}{\vdash x > 0 \vdash [x' = -x] x > 0}
 \end{array}$$

Fortunately, this proper sequent proof confirms the suspicion of a proof that we developed above. In that sense, all is fair in how we come up with a proof, even if we use spooky ghost arguments involving .⁴ But in the end, it is crucial to conduct a proper proof with sound proof rules to ensure the conclusion is valid.

It can be shown [Pla12] that there are properties such as this one that crucially need differential ghosts (or differential auxiliaries) to prove.

11. Summary


The major lesson from today's lecture is that it can sometimes be easier to relate a variable to its initial value or to other quantities than to understand its value in isolation. Ghosts, in their various forms, let us achieve that by adding auxiliary variables into the system dynamics, so that the values of the original variables of interest can be related to the values of the ghosts. Sometimes such ghosts are even necessary to prove properties. As a workaround, it might help to rewrite the original model so that it already includes the ghost variables, preferably marked as ghosts in the model. The phenomenon that relations between state and ghost variables are sometimes easier to prove than just standalone properties of state variables applies in either case. This lecture shines a light on the power of relativity theory in the sense of relating variables to one another.

This lecture also showed how properties of differential equations can be proved using solution-like arguments if only part of the differential equation system can be solved.

A. Axiomatic Ghosts

This section is devoted to yet another kind of ghosts: axiomatic ghosts. While less important for simple systems, axiomatic ghosts are the way to go for systems that involve special functions such as \sin , \cos etc.

When neglecting wind, gravitation, and so on, which is appropriate for analysing cooperation in air traffic control [TPS98], the in-flight dynamics of an aircraft at x can

⁴Of course,  is not quite as spooky as one might suspect. It can be made rigorous with term variables.

be described by the following differential equation system; see [TPS98] for details:

$$x'_1 = v \cos \vartheta \quad x'_2 = v \sin \vartheta \quad \vartheta' = \omega. \quad (6)$$

That is, the linear velocity v of the aircraft changes both positions x_1 and x_2 in the (planar) direction corresponding to the orientation ϑ the aircraft is currently heading toward. Further, the angular velocity ω of the aircraft changes the orientation ϑ of the aircraft.

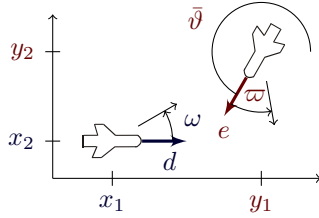


Figure 1: Aircraft dynamics

Unlike for straight-line flight ($\omega = 0$), the nonlinear dynamics in (6) is difficult to analyse [TPS98] for curved flight ($\omega \neq 0$), especially due to the trigonometric expressions which are generally undecidable. Solving (6) requires the Floquet theory of differential equations with periodic coefficients [Wal98, Theorem 18.X] and yields mixed polynomial expressions with multiple trigonometric functions. A true challenge, however, is the need to verify properties of the states that the aircraft reach by following these solutions, which requires proving that complicated formulas with mixed polynomial arithmetic and trigonometric functions hold true for all values of state variables and all possible evolution durations. However, quantified arithmetic with trigonometric functions is undecidable by Gödel's incompleteness theorem [Göd31].

To obtain polynomial dynamics, we axiomatize the trigonometric functions in the dynamics differentially and reparametrize the state correspondingly. Instead of angular orientation ϑ and linear velocity v , we use the linear speed vector

$$d = (d_1, d_2) := (v \cos \vartheta, v \sin \vartheta) \in \mathbb{R}^2$$

which describes both the linear speed $\|d\| := \sqrt{d_1^2 + d_2^2} = v$ and the orientation of the aircraft in space; see Figs. 1 and 2. Substituting this coordinate change into differential equations (6), we immediately have $x'_1 = d_1$ and $x'_2 = d_2$. With the coordinate change, we further obtain differential equations for d_1, d_2 from differential equation system (6) by simple symbolic differentiation:

$$\begin{aligned} d'_1 &= (v \cos \vartheta)' = v' \cos \vartheta + v(-\sin \vartheta)\vartheta' = -(v \sin \vartheta)\omega = -\omega d_2, \\ d'_2 &= (v \sin \vartheta)' = v' \sin \vartheta + v(\cos \vartheta)\vartheta' = (v \cos \vartheta)\omega = \omega d_1. \end{aligned}$$

The middle equality holds for constant linear velocity ($v' = 0$), which we assume, because only limited variations in linear speed are possible and cost-effective during the

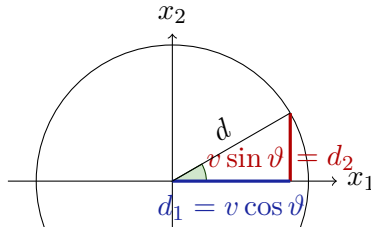


Figure 2: Reparametrize for differential axiomatization

flight [TPS98, LLL00] so that angular velocity ω is the primary control parameter in air traffic control. Hence, equations (6) can be restated as the following differential equation $\mathcal{F}(\omega)$:

$$x'_1 = d_1, x'_2 = d_2, d'_1 = -\omega d_2, d'_2 = \omega d_1 \quad (\mathcal{F}(\omega))$$

$$y'_1 = e_1, y'_2 = e_2, e'_1 = -\varrho e_2, e'_2 = \varrho e_1 \quad (\mathcal{G}(\varrho))$$

Differential equation $\mathcal{F}(\omega)$ expresses that position $x = (x_1, x_2)$ changes according to the linear speed vector $d = (d_1, d_2)$, which in turn rotates according to ω . Simultaneous movement together with a second aircraft at $y \in \mathbb{R}^2$ having linear speed $e \in \mathbb{R}^2$ (also indicated with angle ϑ in Fig. 1) and angular velocity ϱ corresponds to the differential equation $\mathcal{F}(\omega), \mathcal{G}(\varrho)$. Differential equations capture simultaneous dynamics of multiple traffic agents succinctly using conjunction.

By this *differential axiomatization*, we thus obtain polynomial differential equations. Note, however, that their solutions still involve the same complicated nonlinear trigonometric expressions so that solutions still give undecidable arithmetic [Pla10b, Appendix B]. Note that differential invariant type arguments work with the differential equations themselves and not with their solutions, so that differential axiomatization actually helps proving properties, because the solutions are still as complicated as they have always been, but the differential equations become easier

The same technique helps when handling other special functions in other cases by differential axiomatization.

Exercises

Exercise 1. Augment the discrete ghost proofs in Sect. 6 to a full sequent proof of

$$xy - 1 = 0 \rightarrow [x' = x, y' = -y]xy = 1$$

Exercise 2. Augment the proofs in this lecture as described to obtain a full sequent proof of (4). Your advised to find a big sheet of paper, first.

References

- [Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys.*, 38:173–198, 1931.
- [LLL00] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of TCAS. *Proc. IEEE - Special Issue on Hybrid Systems: Theory & Applications*, 88(7):926–947, 2000.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. [doi:10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. [doi:10.1093/logcom/exn070](https://doi.org/10.1093/logcom/exn070).
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. [doi:10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. [doi:10.2168/LMCS-8\(4:16\)2012](https://doi.org/10.2168/LMCS-8(4:16)2012).
- [Pla14] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014. [arXiv:1408.1980](https://arxiv.org/abs/1408.1980).
- [PQ08] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. [doi:10.1007/978-3-540-71070-7_15](https://doi.org/10.1007/978-3-540-71070-7_15).
- [TPS98] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE T. Automat. Contr.*, 43(4):509–521, 1998.
- [Wal98] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1998.

Lecture Notes on Differential Invariants & Proof Theory

[André Platzer](#)

Carnegie Mellon University
Lecture 13

1. Introduction

[Lecture 10 on Differential Equations & Differential Invariants](#) and [Lecture 11 on Differential Equations & Proofs](#) equipped us with powerful tools for proving properties of differential equations without having to solve them. *Differential invariants* (DI) [[Pla10a](#)] prove properties of differential equations by induction based on the right-hand side of the differential equation, rather than its much more complicated global solution. *Differential cuts* (DC) [[Pla10a](#)] made it possible to prove another property C of a differential equation and then change the dynamics of the system around so that it is restricted to never leave that region C . Differential cuts turned out to be very useful when stacking inductive properties of differential equations on top of each other, so that easier properties are proved first and then assumed during the proof of the more complicated properties. In fact, in some cases, differential cuts are crucial for proving properties in the first place [[Pla10a](#), [Pla12c](#), [GSP14](#)]. Differential weakening (DW) [[Pla10a](#)] proves simple properties that are entailed by the evolution domain, which becomes especially useful after the evolution domain constraint has been augmented sufficiently by way of a differential cut.

Just like in the case of loops, where the search for invariants is nontrivial, differential invariants also require some smarts (or good automatic procedures [[PC08](#), [Pla12b](#), [GP14](#), [GSP14](#)]) to be found. Once a differential invariant has been identified, however, the proof follows easily, which is a computationally attractive property.

Finding invariants of loops is very challenging. It can be shown to be the only fundamental challenge in proving safety properties of conventional discrete programs [[HMP77](#)]. Likewise, finding invariants and differential invariants is the only fundamental challenge in proving safety properties of hybrid systems [[Pla08](#), [Pla10b](#), [Pla12a](#)].

A more careful analysis even shows that just finding differential invariants is the only fundamental challenge for hybrid systems safety verification [Pla12a].

That is reassuring, because we know that the proofs will work¹ as soon as we find the right differential invariants. But it also tells us that we can expect the search for differential invariants (and invariants) to be challenging, because cyber-physical systems are extremely challenging, albeit very important. Yet, differential equations also enjoy many pleasant properties that we can exploit to help us find differential invariants.

Since, at the latest after this revelation, we fully realize the importance of studying and understanding differential invariants, we subscribe to developing a deeper understanding of differential invariants right away. The part of their understanding that today's lecture develops is how various classes of differential invariants relate to each other in terms of what they can prove. That is, are there properties that only differential invariants of the form \mathcal{A} can prove, because differential invariants of the form \mathcal{B} cannot ever succeed in proving them? Or are all properties provable by differential invariants of the form \mathcal{A} also provable by differential invariants of the form \mathcal{B} ?

These relations between classes of differential invariants tell us which forms of differential invariants we need to search for and which forms of differential invariants we don't need to bother considering. A secondary goal of today's lecture besides this theoretical understanding is the practical understanding of developing more intuition about differential invariants and seeing them in action more thoroughly.

This lecture is based on [Pla12c] and strikes a balance between comprehensive handling of the subject matter and core intuition. The lecture mostly focuses on the core intuition at the heart of the proofs and leaves a more comprehensive argument and further study for the literature [Pla12c]. Many proofs in this lecture are simplified and only prove the core argument, while leaving out other aspects. Those—very important—further details are beyond the scope of this course, however, and can be found elsewhere [Pla12c]. For example, this lecture will not study whether indirect proofs could conclude the same properties. With a more careful analysis [Pla12c], it turns out that indirect proofs do not change the results reported in this lecture, but the proofs become significantly more complicated and require a more precise choice of the sequent calculus formulation. In this lecture, we will also not always prove all statements conjectured in a theorem. The remaining proofs can be found in the literature [Pla12c].

Note 1 (Proof theory of differential equations). *The results in this lecture are part of the proof theory of differential equations, i.e. the theory of what can be proved about differential equations and with what techniques. They are proofs about proofs, because they prove relations between the provability of logical formulas with different proof calculi. That is, they relate “formula ϕ can be proved using \mathcal{A} ” and “formula ϕ can be proved using \mathcal{B} .”*

The most important learning goals of this lecture are:

Modeling and Control: This lecture helps in understanding the core argumentative

¹Although it may still be a lot of work in practice to make the proofs work. At least they become possible.

principles behind CPS and sheds more light on the question how to tame their analytic complexity.

Computational Thinking: An important part of computer science studies questions about the *limits of computation* or, more generally, develops an understanding of *what can be done* and *what cannot be done*. Either in absolute terms (*computability theory* studies what is computable and what is not) or in relative terms (*complexity theory* studies what is computable in a characteristically quicker way or within classes of resource bounds on time and space). Often times, the most significant understanding of a problem space starts with what cannot be done (the theorem of Rice says that all nontrivial properties of programs are not computable) or what can be done (every problem that can be solved with a deterministic algorithm in polynomial time can also be solved with a nondeterministic algorithm in polynomial time, with the converse being the P versus NP problem).

The primary purpose of this lecture is to develop such an understanding of the limits of what can and what cannot be done in the land of *proofs about differential equations* with what techniques. Not all aspects of this deep question will be possible to answer in one lecture, but it will feature the beginning of the *proof theory of differential equations*, i.e. the theory of provability and proofs about differential equations. Proof theory is, of course, also of interest in other cases, but we will study it in the case that is most interesting and illuminating: the case of proofs about differential equations.

The primary, scientific learning goals of this lecture are, thus, to develop a fundamental understanding of what can and cannot be proved in which way about differential equations. This helps us in our search for differential invariants for applications, because such an understanding prevents us from asking the same analytic question again in equivalent ways (if two different classes of differential invariants prove the same properties and one of them already failed) and guides our search toward the required classes of differential invariants (by next choosing a class that can prove fundamentally more, and of properties of the requisite form). The secondary, pragmatic learning goals are to practice inductive proofs about differential equations using differential invariants and to develop an intuition which verification question to best address in which way. In these ways, both fundamentally and pragmatically, the primary direct impact of this lecture is on understanding rigorous reasoning about CPS models as well as helping to verify CPS models of appropriate scale, in which more than one mode of reasoning is often needed for the various parts and aspects of the system.

CPS Skills: This lecture serves no purpose in CPS Skills that the author could think of, except indirectly via its impact on their analysis.

2. Recap

Recall the following proof rules for differential equations from [Lecture 11 on Differential Equations & Proofs](#):

Note 2 (Proof rules for differential equations).

$$\begin{array}{ll}
 \text{(DI)} \frac{H \vdash F'_{x'}^\theta}{F \vdash [x' = \theta \& H]F} & \text{(DW)} \frac{H \vdash F}{\Gamma \vdash [x' = \theta \& H]F, \Delta} \\
 \text{(DC)} \frac{\Gamma \vdash [x' = \theta \& H]C, \Delta \quad \Gamma \vdash [x' = \theta \& (H \wedge C)]F, \Delta}{\Gamma \vdash [x' = \theta \& H]F, \Delta}
 \end{array}$$

With cuts and generalizations, earlier lectures have also shown that the following can be proved:

$$\frac{A \vdash F \quad F \vdash [x' = \theta \& H]F \quad F \vdash B}{A \vdash [x' = \theta \& H]B} \quad (1)$$

This is useful for replacing a precondition A and postcondition B by another invariant F that implies postcondition B and is implied by precondition A , which will be done frequently in this lecture.

3. Comparative Deductive Study: Relativity Theory for Proofs

In order to find out what we can do when we have been unsuccessfully searching for a differential invariant of one form, we need to understand which other form of differential invariants could work out better. If we have been looking for differential invariants of the form $p = 0$ with a term p without success and then move on to search for differential invariants of the form $p = q$, then we cannot expect to be any more successful than before, because $p = q$ can be rewritten as $p - q = 0$, which is of the first form again. So we should, for example, try finding inequational differential invariants of the form $p \geq 0$, instead. In general, this begs the question which generalizations would be silly (because differential invariants of the form $p = q$ cannot prove any more than those of the form $p = 0$) and when it might be smart (because $p \geq 0$ could still succeed even if everything of the form $p = 0$ failed).

As a principled answer to questions like these, we study the relations of classes of differential invariants in terms of their relative deductive power. That is, we study whether some properties are only provable using differential invariants from the class \mathcal{A} , not using differential invariants from the class \mathcal{B} , or whether all properties provable with differential invariants from class \mathcal{A} are also provable with class \mathcal{B} .

As a basis, we consider a propositional sequent calculus with logical cuts (which simplify glueing derivations together) and real-closed field arithmetic (we denote all uses of real arithmetic by proof rule \mathbb{R}) along the lines of what we say in [Lecture 6 on](#)

Truth & Proof; see [Pla12c] for precise details. By \mathcal{DI} we denote the proof calculus that, in addition, has general differential invariants (rule **DI** with arbitrary quantifier-free first-order formula F) but no differential cuts (rule **DC**). For a set $\Omega \subseteq \{\geq, >, =, \wedge, \vee\}$ of operators, we denote by \mathcal{DI}_Ω the proof calculus where the differential invariant F in rule **DI** is further restricted to the set of formulas that uses only the operators in Ω . For example, $\mathcal{DI}_{=, \wedge, \vee}$ is the proof calculus that allows only and/or-combinations of equations to be used as differential invariants. Likewise, \mathcal{DI}_{\geq} is the proof calculus that only allows atomic weak inequalities $p \geq q$ to be used as differential invariants.

We consider classes of differential invariants and study their relations. If \mathcal{A} and \mathcal{B} are two classes of differential invariants, we write $\mathcal{A} \leq \mathcal{B}$ if all properties provable using differential invariants from \mathcal{A} are also provable using differential invariants from \mathcal{B} . We write $\mathcal{A} \not\leq \mathcal{B}$ otherwise, i.e., when there is a valid property that can only be proven using differential invariants of $\mathcal{A} \setminus \mathcal{B}$. We write $\mathcal{A} \equiv \mathcal{B}$ if $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \leq \mathcal{A}$. We write $\mathcal{A} < \mathcal{B}$ if $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \not\leq \mathcal{A}$. Classes \mathcal{A} and \mathcal{B} are incomparable if $\mathcal{A} \not\leq \mathcal{B}$ and $\mathcal{B} \not\leq \mathcal{A}$.

4. Equivalences of Differential Invariants

Before we go any further, let us study whether there are equivalence transformations on formulas that preserve differential invariance. Every equivalence transformation that we have for differential invariant properties helps us with structuring the proof search space and also helps simplifying the meta-proofs in the proof theory. For example, we should not expect $F \wedge G$ to be a differential invariant for proving a property when $G \wedge F$ was not. Neither would $F \vee G$ be any better as a differential invariant than $G \vee F$.

Lemma 1 (Differential invariants and propositional logic). *Differential invariants are invariant under propositional equivalences. That is, if $F \leftrightarrow G$ is an instance of a propositional tautology then F is a differential invariant of $x' = \theta \ \& \ H$ if and only if G is.*

Proof. In order to prove this, we consider any property that proves with F as a differential invariant and show that G also works. Let F be a differential invariant of a differential equation system $x' = \theta \ \& \ H$ and let G be a formula such that $F \leftrightarrow G$ is an instance of a propositional tautology. Then G is a differential invariant of $x' = \theta \ \& \ H$, because of the following formal proof:

$$\frac{\displaystyle \frac{*}{H \vdash G'_{x'}^\theta}}{\text{DI } G \vdash [x' = \theta \ \& \ H]G} \quad \frac{}{F \vdash [x' = \theta \ \& \ H]F}$$

The bottom proof step is easy to see using (1), because precondition F implies the new precondition G and postcondition F is implied by the new postcondition G propositionally. Subgoal $H \vdash G'_{x'}^\theta$ is provable, because $H \vdash F'_{x'}^\theta$ is provable and G' is defined

as a conjunction over all literals of G . The set of literals of G is identical to the set of literals of F , because the literals do not change by using propositional tautologies. Furthermore, dL uses a propositionally complete base calculus [Pla12c]. \square

In all subsequent proofs, we can use propositional equivalence transformations by Lemma 1. In the following, we will also implicitly use equivalence reasoning for pre- and postconditions *à la* (1) as we have done in Lemma 1. Because of Lemma 1, we can, without loss of generality, work with arbitrary propositional normal forms for proof search.

5. Differential Invariants & Arithmetic

Depending on the reader's exposure to differential structures, it may come as a shock that not all logical equivalence transformations carry over to differential invariants. Differential invariance is not necessarily preserved under real arithmetic equivalence transformations.

Lemma 2 (Differential invariants and arithmetic). *Differential invariants are not invariant under equivalences of real arithmetic. That is, if $F \leftrightarrow G$ is an instance of a first-order real arithmetic tautology then F may be a differential invariant of $x' = \theta \ \& \ H$ yet G may not.*

Proof. There are two formulas that are equivalent over first-order real arithmetic but, for the same differential equation, one of them is a differential invariant, the other one is not (because their differential structures differ). Since $5 \geq 0$, the formula $x^2 \leq 5^2$ is equivalent to $-5 \leq x \wedge x \leq 5$ in first-order real arithmetic. Nevertheless, $x^2 \leq 5^2$ is a differential invariant of $x' = -x$ by the following formal proof:

$$\begin{array}{c} \text{R} \quad \frac{\frac{*}{\vdash -2x^2 \leq 0}}{\vdash (2xx' \leq 0)_{x'}^{-x}} \\ \text{DI} \quad \frac{x^2 \leq 5^2}{\vdash [x' = -x]x^2 \leq 5^2} \end{array}$$

but $-5 \leq x \wedge x \leq 5$ is not a differential invariant of $x' = -x$:

$$\begin{array}{c} \text{not valid} \\ \vdash 0 \leq -x \wedge -x \leq 0 \\ \vdash (0 \leq x' \wedge x' \leq 0)_{x'}^{-x} \\ \text{DI} \quad \frac{-5 \leq x \wedge x \leq 5}{\vdash [x' = -x](-5 \leq x \wedge x \leq 5)} \end{array}$$

\square

For proving the property in the proof of Lemma 2 we need to use the principle (1) with the differential invariant $F \equiv x^2 \leq 5^2$ and cannot use $-5 \leq x \wedge x \leq 5$ directly.

By Lemma 2, we cannot just use arbitrary equivalences when investigating differential invariance, but have to be more careful. Not just the *elementary real arithmetical equivalence* of having the same set of satisfying assignments matters, but also the differential structures need to be compatible. Some equivalence transformations that preserve the solutions still destroy the differential structure. It is the equivalence of *real differential structures* that matters. Recall that differential structures are defined locally in terms of the behavior in neighborhoods of a point, not the point itself.

Lemma 2 illustrates a notable point about differential equations. Many different formulas characterize the same set of satisfying assignments. But not all of them have the same differential structure. Quadratic polynomials have inherently different differential structure than linear polynomials even when they have the same set of solutions over the reals. The differential structure is a more fine-grained information. This is similar to the fact that two elementary equivalent models of first-order logic can still be non-isomorphic. Both the set of satisfying assignments and the differential structure matter for differential invariance. In particular, there are many formulas with the same solutions but different differential structures. The formulas $x^2 \geq 0$ and $x^6 + x^4 - 16x^3 + 97x^2 - 252x + 262 \geq 0$ have the same solutions (all of \mathbb{R}), but very different differential structure; see Fig. 1.

The first two rows in Fig. 1 correspond to the polynomials from the latter two cases. The third row is a structurally different degree 6 polynomial with again the same set of solutions (\mathbb{R}) but a rather different differential structure. The differential structure also depends on what value x' assumes according to the differential equation. Fig. 1 illustrates that p' alone can already have a very different characteristic even if the respective sets of satisfying assignments of $p \geq 0$ are identical.

We can, however, always normalize all atomic subformulas to have right-hand side 0, that is, of the form $p = 0$, $p \geq 0$, or $p > 0$. For instance, $p \leq q$ is a differential invariant if and only if $q - p \geq 0$ is, because $p \leq q$ is equivalent (in first-order real arithmetic) to $q - p \geq 0$ and, moreover, for any variable x and term θ , $(p' \leq q')_{x'}^\theta$ is equivalent to $(q' - p' \geq 0)_{x'}^\theta$ in first-order real arithmetic.

6. Differential Invariant Equations

For equational differential invariants $p = 0$, a.k.a. differential invariant equations, propositional operators do not add to the deductive power.

Proposition 3 (Equational deductive power [Pla10a, Pla12c]). *The deductive power of differential induction with atomic equations is identical to the deductive power of differential induction with propositional combinations of polynomial equations: That is, each formula is provable with propositional combinations of equations as differential invariants iff it is provable with only atomic equations as differential invariants:*

$$\mathcal{DI}_= \equiv \mathcal{DI}_{=, \wedge, \vee}$$

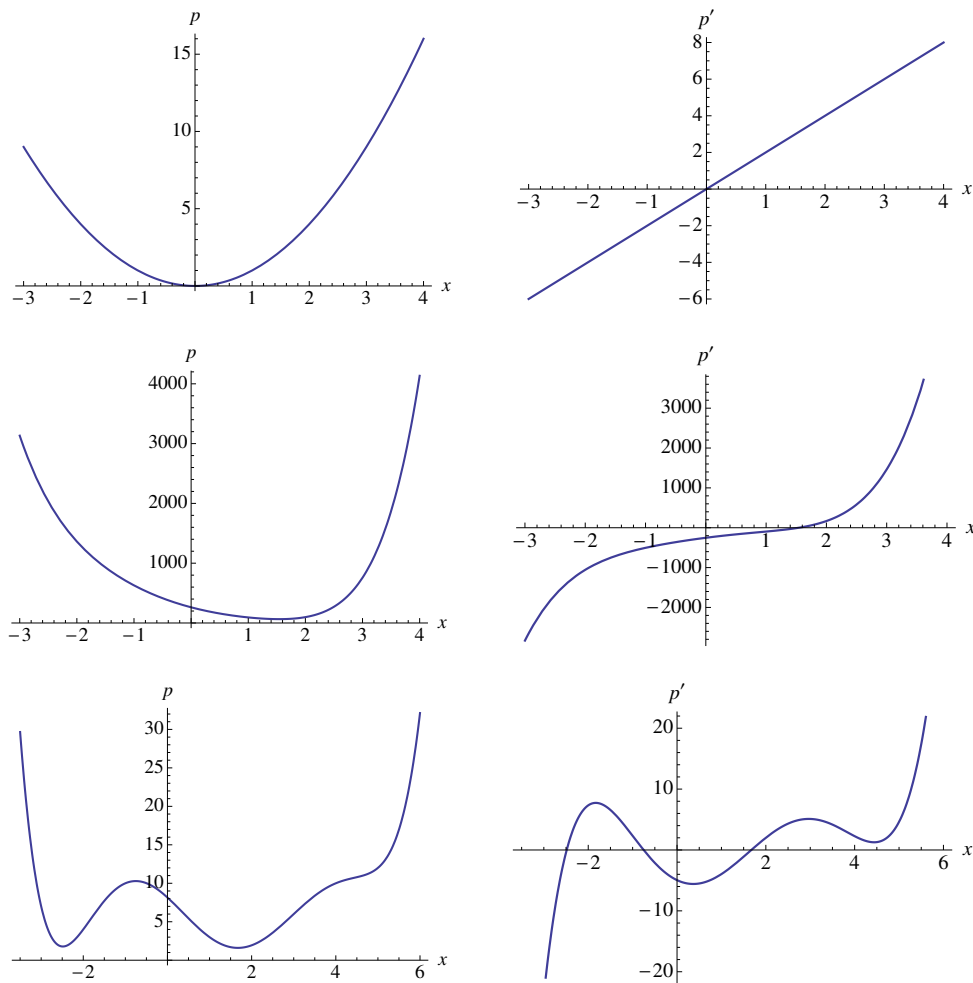


Figure 1: Equivalent solutions ($p \geq 0$ on the left) with quite different differential structure (p' plotted on the right)

How could we prove that?

Before you read on, see if you can find the answer for yourself.

One direction is simple. Proving $\mathcal{DI}_= \leq \mathcal{DI}_{=,\wedge,\vee}$ is obvious, because every proof using a differential invariant equation $p_1 = p_2$ also is a proof using a propositional combination of differential invariant equations. The propositional combination that just consists of the only conjunct $p_1 = p_2$ without making use of any propositional operators.

The other way around $\mathcal{DI}_= \geq \mathcal{DI}_{=,\wedge,\vee}$ is more difficult. If a formula can be proved using a differential invariant that is a propositional combination of equations, such as $p_1 = p_2 \wedge q_1 = q_2$, how could it possibly be proved using just a single equation?

Note 6 (Proofs of equal provability). *A proof of Proposition 3 needs to show that every such provable property is also provable with a structurally simpler differential invariant. It effectively needs to transform proofs with propositional combinations of equations as differential invariants into proofs with just differential invariant equations. And, of course, the proof of Proposition 3 needs to prove that the resulting equations are actually provably differential invariants and prove the same properties as before. This is a general feature of proof theory. It often involves proof transformations at the heart of the arguments.*

Proof of Proposition 3. Let $x' = \theta$ be the (vectorial) differential equation to consider. We show that every differential invariant that is a propositional combination F of polynomial equations is expressible as a single atomic polynomial equation (the converse inclusion is obvious). We can assume F to be in negation normal form by Lemma 1 (recall that negations are resolved and \neq can be assumed not to appear). Then we reduce F inductively to a single equation using the following transformations:

- If F is of the form $p_1 = p_2 \vee q_1 = q_2$, then F is equivalent to the single equation $(p_1 - p_2)(q_1 - q_2) = 0$. Furthermore, $F'_{x'}^\theta \equiv (p'_1 = p'_2 \wedge q'_1 = q'_2)_{x'}^\theta$ directly implies

$$(((p_1 - p_2)(q_1 - q_2))' = 0)_{x'}^\theta \equiv ((p'_1 - p'_2)(q_1 - q_2) + (p_1 - p_2)(q'_1 - q'_2) = 0)_{x'}^\theta$$

which implies that the differential structure is the same so that the inductive steps are equivalent (either both succeed or both fail).

- If F is of the form $p_1 = p_2 \wedge q_1 = q_2$, then F is equivalent to the single equation $(p_1 - p_2)^2 + (q_1 - q_2)^2 = 0$. Furthermore, $F'_{x'}^\theta \equiv (p'_1 = p'_2 \wedge q'_1 = q'_2)_{x'}^\theta$ implies

$$(((p_1 - p_2)^2 + (q_1 - q_2)^2)' = 0)_{x'}^\theta \equiv (2(p_1 - p_2)(p'_1 - p'_2) + 2(q_1 - q_2)(q'_1 - q'_2) = 0)_{x'}^\theta$$

Consequently propositional connectives of equations can be replaced by their equivalent arithmetic equations in pre- and postconditions, and the corresponding induction steps are equivalent. \square

Note that the polynomial degree increases quadratically by the reduction in Proposition 3, but, as a trade-off, the propositional structure simplifies. Consequently, differential invariant search for the equational case can either exploit propositional structure

with lower degree polynomials or suppress the propositional structure at the expense of higher degrees. This trade-off depends on the real arithmetic decision procedure, but is often enough in favor of keeping propositional structure, because the proof calculus can still exploit the logical structure to decompose the verification question before invoking real arithmetic. There are cases, however, where such reductions are formidably insightful [Pla12b].

Equational differential invariants, thus, enjoy a lot of beautiful properties, including characterizing invariant functions [Pla12b] and generalizing to a decision procedure for algebraic invariants of algebraic differential equations [GP14].

7. Equational Incompleteness

Focusing exclusively on differential invariants with equations reduces the deductive power, because sometimes only differential invariant inequalities can prove properties.

Proposition 4 (Equational incompleteness). *The deductive power of differential induction with equational formulas is strictly less than the deductive power of general differential induction, because some inequalities cannot be proven with equations.*

$$\begin{aligned} \mathcal{DI}_= &\equiv \mathcal{DI}_{=,\wedge,\vee} < \mathcal{DI} \\ \mathcal{DI}_{\geq} &\not\equiv \mathcal{DI}_= \equiv \mathcal{DI}_{=,\wedge,\vee} \\ \mathcal{DI}_{>} &\not\equiv \mathcal{DI}_= \equiv \mathcal{DI}_{=,\wedge,\vee} \end{aligned}$$

How could such a proposition be proved?

Before you read on, see if you can find the answer for yourself.

The proof strategy for the proof of Proposition 3 involved transforming proofs into proofs to prove the inclusion $\mathcal{DI}_= \geq \mathcal{DI}_{=,\wedge,\vee}$. Could the same strategy prove Proposition 4? No, because we need to show the opposite! Proposition 4 conjectures $\mathcal{DI}_\geq \not\leq \mathcal{DI}_{=,\wedge,\vee}$, which means that there are true properties that are only provable using a differential invariant inequality $p_1 \geq p_2$ and not using any differential invariant equations or propositional combinations thereof.

For one thing, this means that we ought to find a property that a differential invariant inequality can prove. That ought to be easy enough, because [Lecture 11 on Differential Equations & Proofs](#) showed us how useful differential invariants are. But then a proof of Proposition 4 also requires a proof why that very same formula cannot possibly ever be proved with any way of using only differential invariant equations or their propositional combinations. That is a proof about nonprovability. Proving provability in proof theory amounts to producing a proof (in sequent calculus). Proving nonprovability most certainly does not mean it would be enough to write something down that is not a proof. After all, just because one proof attempt fails does not mean that other attempts would not be successful. You have experienced this while you were working on proving your labs for this course. The first proof attempt might have failed miserably and was impossible to ever work out. But, come next day, you had a better idea with a different proof, and suddenly the same property turned out to be perfectly provable even if the first proof attempt failed.

How could we prove that all proof attempts do not work?

Before you read on, see if you can find the answer for yourself.

One way of showing that a logical formula cannot be proved is by giving a counterexample, i.e. a state which assigns values to the variables that falsify the formula. That is, of course, not what can help us proving Proposition 4, because a proof of Proposition 4 requires us to find a formula that can be proved with \mathcal{DI}_{\geq} (so it cannot have any counterexamples, since it is perfectly valid), just cannot be proved with $\mathcal{DI}_{=,\wedge,\vee}$. Proving that a valid formula cannot be proved with $\mathcal{DI}_{=,\wedge,\vee}$ requires us to show that all proofs in $\mathcal{DI}_{=,\wedge,\vee}$ do not prove that formula.

Expedition 1 (Proving differences in set theory and linear algebra). Recall sets. The way to prove that two sets M, N have the same “number” of elements is to come up with a pair of functions $\Phi : M \rightarrow N$ and $\Psi : N \rightarrow M$ between the sets and then prove that Φ, Ψ are inverses of each other, i.e. $\Phi(\Psi(y)) = y$ and $\Psi(\Phi(x)) = x$ for all $x \in M, y \in N$. Proving that two sets M, N do not have the same “number” of elements works entirely differently, because that has to prove for all pairs of functions $\Phi : M \rightarrow N$ and $\Psi : N \rightarrow M$ that there is an $x \in M$ such that $\Psi(\Phi(x)) \neq x$ or an $y \in N$ such that $\Phi(\Psi(y)) \neq y$. Since writing down every such pair of functions Φ, Ψ is a lot of work (an infinite amount of work if M and N are infinite), indirect criteria such as cardinality or countability are used instead, e.g. for proving that the reals \mathbb{R} and rationals \mathbb{Q} cannot possibly have the same number of elements, because \mathbb{Q} are countable but \mathbb{R} are not (by Cantor’s diagonal argument).

Recall vector spaces from linear algebra. The way to prove that two vector spaces V, W are isomorphic is to think hard and construct a function $\Phi : V \rightarrow W$ and a function $\Psi : W \rightarrow V$ and then prove that Φ, Ψ are linear functions and inverses of each other. Proving that two vector spaces V, W are *not* isomorphic works entirely differently, because that has to prove that all pairs of functions $\Phi : V \rightarrow W$ and $\Psi : W \rightarrow V$ are either not linear or not inverses of each other. Proving the latter literally is again a lot (usually infinite) amount of work. So instead, indirect criteria are being used. One proof that two vector spaces V, W are not isomorphic could show that both have different dimensions and then prove that isomorphic vector spaces always have the same dimension, so V and W cannot possibly be isomorphic.

By analogy, proving non-provability leads to a study of indirect criteria about proofs of differential equations.

Note 8 (Proofs of different provability). *Proving non-reducibility $\mathcal{A} \not\leq \mathcal{B}$ for classes of differential invariants requires an example formula ϕ that is provable in \mathcal{A} plus a proof that no proof using \mathcal{B} proves ϕ . The preferred way of doing that is finding an indirect criterion that all proofs in \mathcal{B} possess but that ϕ does not have, so that the proofs using \mathcal{B} cannot possibly succeed in proving ϕ .*

Proof of Proposition 4. Consider any positive term $a > 0$ (e.g., 5 or $x^2 + 1$ or $x^2 + x^4 + 2$). The following proof proves a formula by differential induction with the weak inequal-

ity $x \geq 0$:

$$\frac{\mathbb{R} \frac{*}{\vdash a \geq 0}}{\text{DI} \frac{x \geq 0 \vdash [x' = a]x \geq 0}{}}$$

The same formula is not provable with an equational differential invariant, however. Any univariate polynomial p that is zero on all $x \geq 0$ is the zero polynomial and, thus, an equation of the form $p = 0$ cannot be equivalent to the half space $x \geq 0$. By the equational deductive power theorem 3, the above formula then is not provable with any Boolean combination of equations as differential invariant either, because propositional combinations of equational differential invariants prove the same properties that single equational differential invariants do, and the latter cannot succeed in proving $x \geq 0 \rightarrow [x' = a]x \geq 0$.

The other parts of the theorem that involve generalizations of the non-provability argument to other indirect proofs using cuts and the like are proved elsewhere [Pla12c]. \square

It might be tempting to think that at least equational postconditions only need equational differential invariants for proving them. But that is not the case either [Pla12c]. So even if the property you care to prove involves only equations, you may still need to generalize your proof arguments to consider inequalities instead.

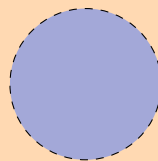
8. Strict Differential Invariant Inequalities

We show that, conversely, focusing on strict inequalities $p > 0$ also reduces the deductive power, because equations are obviously missing and there is at least one proof where this matters. That is, what are called strict barrier certificates do not prove (non-trivial) closed invariants.

Expedition 2 (Topology in real analysis). The following proofs distinguish open sets from closed sets, which are concepts from real analysis (or topology).cd Roughly: A closed set is one whose boundary belongs to the set. For example the solid unit disk of radius 1. An open set is one for which no point of the boundary belongs to the set, for example the unit disk of radius 1 without the outer circle of radius 1.



closed solid disk
 $x^2 + y^2 \leq 1$
with boundary



open disk
 $x^2 + y^2 < 1$
without
boundary

A set $O \subseteq \mathbb{R}^n$ is *open* iff there is a small neighborhood that is contained in O around every point of O . That is, for all points $a \in O$ there is an $\varepsilon > 0$ such that every point b of distance at most ε from a is still in O . A set $C \subseteq \mathbb{R}^n$ is *closed* iff its complement is open. Because \mathbb{R}^n is what is called a complete metric space, a set $C \subseteq \mathbb{R}^n$ is closed iff every convergent sequence of elements in C converges to a limit in C .

Proposition 5 (Strict barrier incompleteness). *The deductive power of differential induction with strict barrier certificates (formulas of the form $p > 0$) is strictly less than the deductive power of general differential induction.*

$$\begin{aligned} DI_{>} &< DI \\ DI_{=} &\not\leq DI_{>} \end{aligned}$$

Proof. The following proof proves a formula by equational differential induction:

$$\frac{\mathbb{R} \quad \frac{*}{\vdash 2xy + 2y(-x) = 0}}{DI_{x^2 + y^2 = c^2} \vdash [x' = y, y' = -x] x^2 + y^2 = c^2}$$

But the same formula is not provable with a differential invariant of the form $p > 0$. An invariant of the form $p > 0$ describes an open set and, thus, cannot be equivalent to the (nontrivial) closed set where $x^2 + y^2 = c^2$ holds true. The only sets that are both open and closed in (the Euclidean space) \mathbb{R}^n are the empty set \emptyset and the full space \mathbb{R}^n .

The other parts of the theorem are proved elsewhere [Pla12c]. \square

One takeaway message is that it makes sense to check whether the desired invariant is an open or a closed set and use differential invariants of the suitable type for the job. Of course, both $p = 0$ and $p \geq 0$ might still work for closed sets.

9. Differential Invariant Equations as Differential Invariant Inequalities

Weak inequalities $p \geq 0$, however, do subsume the deductive power of equational differential invariants $p = 0$. This is obvious on the algebraic level but we will see that it also does carry over to the differential structure.

Proposition 6 (Equational definability). *The deductive power of differential induction with equations is subsumed by the deductive power of differential induction with weak inequalities:*

$$DI_{=, \wedge, \vee} \leq DI_{\geq}$$

Proof. By Proposition 3, we only need to show that $\mathcal{DI}_= \leq \mathcal{DI}_{\geq}$, as $\mathcal{DI}_{=,\wedge,\vee} = \mathcal{DI}_=$. Let $p = 0$ be an equational differential invariant of a differential equation $x' = \theta \ \& \ H$. Then we can prove the following:

$$\frac{\frac{*}{H \vdash (p' = 0)_{x'}}{\text{DI } p = 0 \vdash [x' = \theta \ \& \ H] p = 0}}$$

Then, the inequality $-p^2 \geq 0$, which is equivalent to $p = 0$ in real arithmetic, also is a differential invariant of the same dynamics by the following formal proof:

$$\frac{\frac{*}{H \vdash (-2pp' \geq 0)_{x'}}{\text{DI } -p^2 \geq 0 \vdash [x' = \theta \ \& \ H] (-p^2 \geq 0)}}$$

The subgoal for the differential induction step is provable: if we can prove that H implies $(p' = 0)_{x'}$ according to the first sequent proof, then we can also prove that H implies $(-2pp' \geq 0)_{x'}$ for the sequent proof, because $(p' = 0)_{x'}$ implies $(-2pp' \geq 0)_{x'}$ in first-order real arithmetic. \square

Note that the local state-based view of differential invariants is crucial to make the last proof work. By Proposition 6, differential invariant search with weak inequalities can suppress equations. Note, however, that the polynomial degree increases quadratically with the reduction in Proposition 6. In particular, the polynomial degree increases quadratically when using the reductions in Proposition 3 and Proposition 6 one after another to turn propositional equational formulas into single inequalities. This quartic increase of the polynomial degree is likely a too serious computational burden for practical purposes even if it is a valid reduction in theory.

10. Differential Invariant Atoms

Next we see that, with the notable exception of pure equations (Proposition 3), propositional operators increase the deductive power.

Theorem 7 (Atomic incompleteness). *The deductive power of differential induction with propositional combinations of inequalities exceeds the deductive power of differential induction with atomic inequalities.*

$$\begin{aligned} \mathcal{DI}_{\geq} &< \mathcal{DI}_{\geq,\wedge,\vee} \\ \mathcal{DI}_{>} &< \mathcal{DI}_{>,\wedge,\vee} \end{aligned}$$

Proof. Consider any term $a \geq 0$ (e.g., 1 or x^2+1 or x^2+x^4+1 or $(x-y)^2+2$). Then the formula $x \geq 0 \wedge y \geq 0 \rightarrow [x' = a, y' = y^2](x \geq 0 \wedge y \geq 0)$ is provable using a conjunction in the differential invariant:

$$\begin{array}{c}
 * \\
 \hline
 \mathbb{R} \quad \vdash a \geq 0 \wedge y^2 \geq 0 \\
 \hline
 \vdash (x' \geq 0 \wedge y' \geq 0)_{x' y'}^a y^2 \\
 \hline
 \text{DI} \quad x \geq 0 \wedge y \geq 0 \vdash [x' = a, y' = y^2](x \geq 0 \wedge y \geq 0)
 \end{array}$$

By a sign argument similar to that in the proof of [Pla10a, Theorem 2] and [Pla10b, Theorem 3.3], no atomic formula is equivalent to $x \geq 0 \wedge y \geq 0$. Basically, no formula of the form $p(x, y) \geq 0$ for a polynomial p can be equivalent to $x \geq 0 \wedge y \geq 0$, because that would imply that $p(x, 0) \geq 0 \leftrightarrow x \geq 0$ for all x , which, as $p(x, 0)$ is a univariate polynomial with infinitely many roots (for every $x \geq 0$), which implies that $p(x, 0)$ is the zero polynomial, which is not equivalent to $x \geq 0$, because the zero polynomial is also zero on $x < 0$. Similar arguments work for $p(x, y) > 0$ and $p(x, y) = 0$. Thus, the above property cannot be proven using a single differential induction. The proof for a postcondition $x > 0 \wedge y > 0$ is similar.

The other—quite substantial—parts of the proof are proved elsewhere [Pla12c]. \square

Note that the formula in the proof of Theorem 7 is provable, e.g., using differential cuts (DC) with two atomic differential induction steps, one for $x \geq 0$ and one for $y \geq 0$. Yet, a similar, yet much more involved, argument can be made to show that the deductive power of differential induction with atomic formulas (even when using differential cuts) is strictly less than the deductive power of general differential induction; see [Pla10a, Theorem 2].

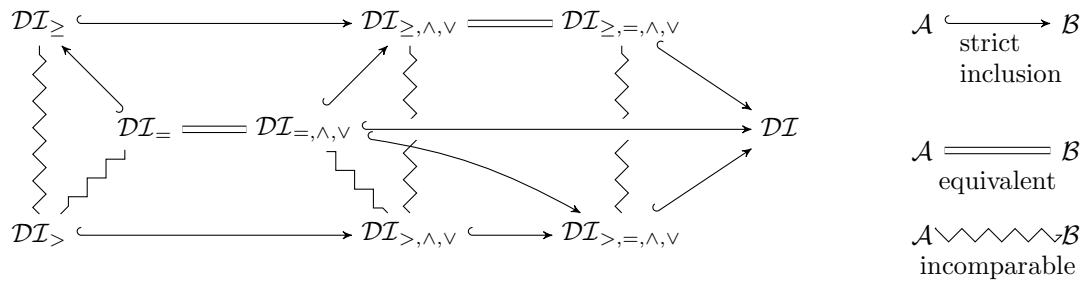
Consequently, in the case of inequalities, propositional connectives can be quite crucial when looking for differential invariants.

11. Summary

Fig. 2 summarizes the findings on provability relations of differential equations explained in this lecture and others reported in the literature [Pla12c]. We have considered the differential invariance problem, which, by a relative completeness argument [Pla12a], is at the heart of hybrid systems verification. To better understand structural properties of hybrid systems, we have identified and analyzed more than a dozen (16) relations between the deductive power of several (9) classes of differential invariants, including subclasses that correspond to related approaches. An understanding of these relations helps guide the search for suitable differential invariants and also provides an intuition for exploiting indirect criteria such as open/closedness of sets as a guide.

The results require a symbiosis of elements of logic with real arithmetical, differential, semialgebraic, and geometrical properties. Future work includes investigating this new

field further called *real differential semialgebraic geometry*, whose development has only just begun [Pla12c, GSP14, GSP15].



DI_{Ω} : properties verifiable using differential invariants built with operators from Ω

Figure 2: Differential invariance chart

(strict inclusions $\mathcal{A} < \mathcal{B}$, equivalences $\mathcal{A} \equiv \mathcal{B}$, and incomparabilities $\mathcal{A} \not\leq \mathcal{B}$, $\mathcal{B} \not\leq \mathcal{A}$ for classes of differential invariants are indicated)

A. Curves Playing with Norms and Degrees

The proof of Lemma 2 showed a case where a formula with a higher-degree polynomial was needed to prove a property that a lower-degree polynomial could not prove. The conclusion from the proof of Lemma 2 is not that it is always better to use differential invariants of higher degrees, just because that worked in this particular proof.

For example, the following proof for an upper bound t on the supremum norm $\|(x, y)\|_{\infty}$ of the vector (x, y) defined as

$$\|(x, y)\|_{\infty} \leq t \stackrel{\text{def}}{=} -t \leq x \leq t \wedge -t \leq y \leq t \quad (2)$$

is significantly easier for the curved dynamics:

$$\begin{array}{c} \mathbb{R} \\ \hline * \\ d^2 + e^2 \leq 1 \vdash -1 \leq d \leq 1 \wedge -1 \leq e \leq 1 \\ \hline d^2 + e^2 \leq 1 \vdash (-t' \leq x' \leq t' \wedge -t' \leq y' \leq t') \frac{d}{x'} \frac{e}{y'} \frac{\omega e}{d'} \frac{-\omega d}{e'} \frac{1}{t'} \\ \hline \text{DI} \quad \triangleleft d^2 + e^2 \leq 1 \wedge x = y = t = 0 \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d, t' = 1 \& d^2 + e^2 \leq 1] \|(x, y)\|_{\infty} \leq t \\ \text{DC} \quad d^2 + e^2 \leq 1 \wedge x = y = t = 0 \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d, t' = 1] \|(x, y)\|_{\infty} \leq t \end{array}$$

where the first premise of the differential cut (DC) above is elided (marked \triangleleft) and proves as in [Lecture 11 on Differential Invariants & Proofs](#). This proof shows that a point (x, y) starting with linear velocity at most 1 and angular velocity ω from the origin will not move further than the time t in supremum norm.

This simple proof is to be contrasted with the following proof attempt for a corresponding upper bound on the Euclidean norm $\|(x, y)\|_2$ defined as

$$\|(x, y)\|_2 \leq t \stackrel{\text{def}}{=} x^2 + y^2 \leq t^2 \quad (3)$$

for which a direct proof fails:

not valid

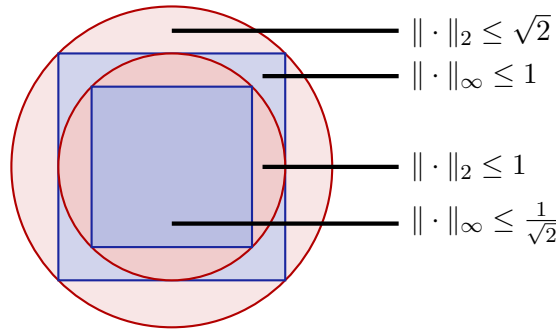
$$\begin{array}{c} \hline d^2 + e^2 \leq 1 \vdash 2xd + 2ye \leq 2t \\ \hline d^2 + e^2 \leq 1 \vdash (2xx' + 2yy' \leq 2tt')_{x' y' d' e' t'}^{d e \omega e -\omega d 1} \\ \hline \text{DI } d^2 + e^2 \leq 1 \wedge x = y = t = 0 \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d, t' = 1 \ \& \ d^2 + e^2 \leq 1] \|(x, y)\|_2 \leq t \\ \text{DC } d^2 + e^2 \leq 1 \wedge x = y = t = 0 \vdash [x' = d, y' = e, d' = \omega e, e' = -\omega d, t' = 1] \|(x, y)\|_2 \leq t \end{array}$$

An indirect proof is still possible but much more complicated. But the proof using the supremum norm (2) is much easier than the proof using the Euclidean norm (3) in this case. In addition, the arithmetic complexity decreases, because supremum norms are definable in linear arithmetic (2) unlike the quadratic arithmetic required for Euclidean norms (3). Finally, the simpler proof is, up to a factor of $\sqrt{2}$ just as good, because quantifier elimination easily proves that the supremum norm $\|\cdot\|_\infty$ and the standard Euclidean norm $\|\cdot\|_2$ are equivalent, i.e., their values are identical up to constant factors:

$$\forall x \forall y (\|(x, y)\|_\infty \leq \|(x, y)\|_2 \leq \sqrt{n} \|(x, y)\|_\infty) \quad (4)$$

$$\forall x \forall y \left(\frac{1}{\sqrt{n}} \|(x, y)\|_2 \leq \|(x, y)\|_\infty \leq \|(x, y)\|_2 \right) \quad (5)$$

where n is the dimension of the vector space, here 2. That makes sense, because if, e.g., the coordinate with maximal absolute value is at most 1, then the Euclidean distance can be at most 1. And the extra factor of $\sqrt{2}$ is easily justified by Pythagoras' theorem.



Exercises

Exercise 1. Prove the relation $\mathcal{DI}_> \leq \mathcal{DI}_{>, \wedge, \vee}$, i.e., that all properties provable using differential invariants of the form $p > q$ are also provable using propositional combinations of these formulas as differential invariants.

Exercise 2. Prove the relation $\mathcal{DI}_\geq \equiv \mathcal{DI}_{\leq, \wedge, \vee}$.

Exercise 3. Prove the relation $\mathcal{DI}_{\geq, \wedge, \vee} \equiv \mathcal{DI}_{\geq, =, \wedge, \vee}$.

Exercise 4. Let \mathcal{DI}_{true} denote the proof calculus in which only the formula *true* is allowed as a differential invariant. Prove the relation $\mathcal{DI}_{true} < \mathcal{DI}_=$.

Exercise 5. Let \mathcal{DI}_{false} denote the proof calculus in which only the formula *false* is allowed as a differential invariant. Prove the relation $\mathcal{DI}_{false} < \mathcal{DI}_>$.

Exercise 6. Prove the relation $\mathcal{DI}_{=,\wedge,\vee} < \mathcal{DI}_{\geq,\wedge,\vee}$.

Exercise 7. Prove the relation $\mathcal{DI}_{>,\wedge,\vee} < \mathcal{DI}_{>,\leq,\wedge,\vee}$.

Exercise 8. Prove the norm relations (4) and (5). Use these relations in a sequent proof to relate the successful proof with a bound on the supremum norm $\|(x, y)\|_\infty$ to a result about a bound on the Euclidean norm $\|(x, y)\|_2$.

References

- [GP14] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. doi:[10.1007/978-3-642-54862-8_19](https://doi.org/10.1007/978-3-642-54862-8_19).
- [GSP14] Khalil Ghorbal, Andrew Sogokon, and André Platzer. Invariance of conjunctions of polynomial equalities for algebraic differential equations. In Markus Müller-Olm and Helmut Seidl, editors, *SAS*, volume 8723 of *LNCS*, pages 151–167. Springer, 2014. doi:[10.1007/978-3-319-10936-7_10](https://doi.org/10.1007/978-3-319-10936-7_10).
- [GSP15] Khalil Ghorbal, Andrew Sogokon, and André Platzer. A hierarchy of proof rules for checking differential invariance of algebraic sets. In Deepak D’Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *VMCAI*, *LNCS*. Springer, 2015.
- [HMP77] David Harel, Albert R. Meyer, and Vaughan R. Pratt. Computability and completeness in logics of programs (preliminary report). In *STOC*, pages 261–268. ACM, 1977.
- [PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. doi:[10.1007/978-3-540-70545-1_17](https://doi.org/10.1007/978-3-540-70545-1_17).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. doi:[10.1093/logcom/exn070](https://doi.org/10.1093/logcom/exn070).
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. A differential operator approach to equational differential invariants. In Lennart Beringer and Amy Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 28–48. Springer, 2012. doi:[10.1007/978-3-642-32347-8_3](https://doi.org/10.1007/978-3-642-32347-8_3).

- [Pla12c] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. [doi:10.2168/LMCS-8\(4:16\)2012](https://doi.org/10.2168/LMCS-8(4:16)2012).

Lecture Notes on Differential & Temporal Logics

André Platzer

Carnegie Mellon University
Lecture 16

1 Introduction

This course is devoted to the study of the [Foundations of Cyber-Physical Systems](#) [Pla12c, Pla12b]. [Lecture 3 on Choice & Control](#) explained hybrid programs, a program notation for hybrid systems [Pla08, Pla10, Pla12c, Pla12a]. [Lecture 4 on Safety & Contracts](#) defined differential dynamic logic [Pla08, Pla10, Pla12c, Pla12a] as a specification and verification logic for hybrid programs. [Lecture 5 on Dynamical Systems & Dynamic Axioms](#) and subsequent lectures studied proof principles for differential dynamic logic with which we can prove correctness properties of hybrid systems. In your labs, you have demonstrated aptly how you can model, specify, and verify quite sophisticated and challenging robots.

Yet, there was one rather puzzling phenomenon that we noticed in [Lecture 4](#) only then did not have a chance to consider any further. For a hybrid program α and differential dynamic logic formula ϕ , the modal formula

$$[\alpha]\phi$$

expresses that all *final* states reached by all runs of α satisfy the logical formula ϕ . The modal formula $[\alpha]\phi$ is, consequently, false exactly in those states from which α can reach a final state that violates the safety condition ϕ . Yet, what about states from which the final state reached by running α is safe but some intermediate state along the execution of α was not safe?

Shouldn't systems that violate safety condition ϕ at an intermediate state be considered unsafe as well?

The short answer is: that depends.

Does it even make a difference whether we study intermediate states as well or only worry about final states?

The short answer is again: that depends.

What exactly it depends on and how to systematically approach the general case of safety *throughout* the system execution is what today's lecture studies. The key to the answer will be understanding the temporal behavior of hybrid programs. The hybrid trace semantics of hybrid programs will also give us a deeper understanding of the hybrid aspect of time in hybrid systems.

This lecture is based on [Pla10, Chapter 4], which is a significant extension of [Pla07], and incorporates some aspects of follow-up work [JP14] to which we refer for a more general account to temporal aspects in hybrid systems verification. The thoughts on time in this lecture are related to an upcoming article [Pla14].

The most important learning goals of this lecture are:

Modeling and Control: We find identify one additional dynamical aspect, the aspect of temporal dynamics, i.e. how state changes over time throughout a system execution. It is important to learn to judge under which circumstance temporal dynamics is important for understanding a CPS and when it can be neglected without loss. Part of today's lecture is also about understanding time, never a bad goal to have.

Computational Thinking: This lecture addresses subtle aspects with identifying specifications and critical properties of CPS. We will also see how to express temporal variations of postconditions for CPS models. This lecture introduces *differential temporal dynamic logic* dTL [Pla10] extending the differential dynamic logic that is used as the specification and verification language for CPS in the other parts of this course by temporal aspects. Secondary goals in this lecture are practicing the first half of the logical trinity consisting of the relationship of syntax and semantics.

CPS Skills: We add a new dimension into our understanding of the semantics of a CPS model: the temporal dimension corresponding to how exactly a system changes state as a function of time. Such temporal changes have been implicit in the semantics of hybrid programs so far, because that was based on reachability relations. Today's lecture will make the temporal change explicit as a function of time. This helps understanding nuances in the semantics of hybrid systems either based on state reachability or on temporal traces, which further helps sharpen our intuition for the operational effects of CPS as dynamic functions over time.

2 Temporalizing Hybrid Systems

In order to be able to distinguish whether a CPS is safe at the end of its run or safe always throughout its run, differential dynamic logic $\text{d}\mathcal{L}$ will be extended with additional temporal modalities. The resulting logic extends $\text{d}\mathcal{L}$ and is called *differential temporal dynamic logic* (dTL) [Pla10, Chapter 4]. The modal formula

$$[\alpha]\phi$$

of $d\mathcal{L}$ [Pla08, Pla12c] expresses that all *final* states reached by all runs of α satisfy the logical formula ϕ . The same $d\mathcal{L}$ formula $[\alpha]\phi$ is allowed in the logic dTL and has the same semantics [Pla10, Chapter 4]. The new temporal modal dTL formula

$$[\alpha]\Box\phi$$

instead, expresses that all states reached *all along* all traces of α satisfy ϕ . Those two modalities can be used to distinguish systems that are always throughout from those that are only safe in final states. For example, if the dTL formula

$$[\alpha]\phi \wedge \neg[\alpha]\Box\phi$$

is true in an initial state ν , then the system α will be safe (in the sense of satisfying ϕ) in all final states reached after running α from ν , but is not safe always throughout all traces of all runs of α from ν . Can that happen?

You should try to answer this question before it is discussed in a later part of these lecture notes.

3 Syntax of Differential Temporal Dynamic Logic

The *differential temporal dynamic logic* dTL extends differential dynamic logic [Pla08, Pla10, Pla12c] with temporal modalities for verifying temporal specifications of hybrid systems. Hence, dTL has two kinds of modalities:

Modal operators. Modalities of dynamic logic express statements about all possible behaviour $([\alpha]\pi)$ of a system α , or about the existence of a trace $(\langle\alpha\rangle\pi)$, satisfying condition π . Unlike in standard dynamic logic, α is a model of a hybrid system. The logic dTL uses hybrid programs to describe α as in previous lectures. Yet, unlike in standard dynamic logic [HKT00] or $d\mathcal{L}$, π is a *trace formula* in dTL, and π can refer to all states that occur *during* a trace using temporal operators.

Temporal operators. For dTL, the temporal trace formula $\Box\phi$ expresses that the formula ϕ holds all along a trace selected by $[\alpha]$ or $\langle\alpha\rangle$. For instance, the state formula $\langle\alpha\rangle\Box\phi$ says that the state formula ϕ holds at every state along at least one trace of α . Dually, the trace formula $\Diamond\phi$ expresses that ϕ holds at some point during such a trace. It can occur in a state formula $\langle\alpha\rangle\Diamond\phi$ to express that there is such a state in some trace of α , or as $[\alpha]\Diamond\phi$ to say that along each trace there is a state satisfying ϕ . The primary focus of attention in today's lecture is on homogeneous combinations of path and trace quantifiers like $[\alpha]\Box\phi$ or $\langle\alpha\rangle\Diamond\phi$.

The formulas of dTL are defined similarly to differential dynamic logic. However, the modalities $[\alpha]$ and $\langle\alpha\rangle$ accept trace formulas that refer to the temporal behavior of *all* states along a trace. Inspired by CTL and CTL* [EC82, EH86], dTL distinguishes between state formulas, which are true or false in states, and trace formulas, which are true or false for system traces. The sets Fml of state formulas and Fml_T of trace formulas with variables in Σ are simultaneously inductively defined in Def. 1.

Definition 1 (dTL formula). The (state) formulas of differential temporal dynamic logic (dTL) are defined by the grammar (where ϕ, ψ are dTL state formulas, π is a dTL trace formula, θ_1, θ_2 (polynomial) terms, x a variable, α a HP):

$$\phi, \psi ::= \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\pi \mid \langle \alpha \rangle \pi$$

The trace formulas of dTL are defined by the grammar (where ϕ is a dTL state formula):

$$\pi ::= \phi \mid \Box\phi \mid \Diamond\phi$$

Operators $>, \leq, <, \leftrightarrow$ can be defined as usual, e.g., $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

Formulas without \Box and \Diamond are *nontemporal formulas* and have the same semantics as the corresponding dL formulas. Unlike in CTL, dTL state formulas are true on a trace if they hold for the *last* state of a trace, not for the first. Thus, when ϕ is a state formula, dTL formula $[\alpha]\phi$ expresses that ϕ is true at the end of each trace of α , which is the same as the dL formula $[\alpha]\phi$. In contrast, $[\alpha]\Box\phi$ expresses that ϕ is true *all along* all states of every trace of α . This combination gives a smooth embedding of nontemporal dL into dTL and makes it possible to define a compositional calculus. Like CTL, dTL allows nesting with a branching time semantics [EC82], e.g., $[\alpha]\Box(x \geq 2 \rightarrow \langle \beta \rangle \Diamond x \leq 0)$.

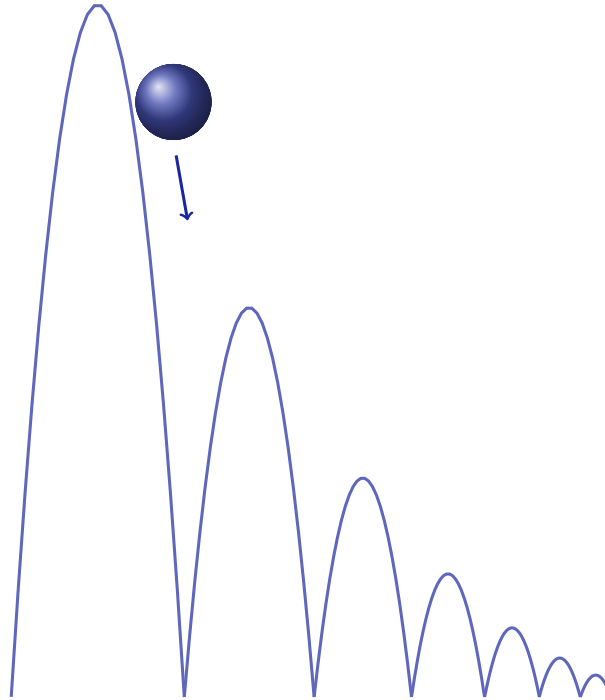


Figure 1: Sample trajectory of a bouncing ball (plotted as position over real time)

4 Hybrid Time

The semantics of differential temporal dynamic logic refers to the temporal behavior of hybrid programs along a trace over time. Our first goal will therefore be to find the right notion of time for the job.

Let us consider the familiar bouncing ball; see Fig. 1. The bouncing ball is flying through the air toward the ground, bounces back up when it hits the ground, and will again fly up. Then, as gravity wins over, it will fly down again for a second bounce, and so forth, leading to a lot of interesting physics including questions of how the kinetic energy transforms into potential energy as the ball deforms by an elastic collision on the ground and then reverses the deformation to gain kinetic energy [Cro00].

Alternatively, we decided in [Lecture 4 on Safety & Contracts](#) to put our multi-dynamical systems glasses on [Pla12c] and realized that the bouncing ball dynamics consists of two phases that, individually, are easy to describe and interact to form a hybrid system. There is the flying part, where the ball does not do anything but move according to gravity.¹ And then there is the bouncing part, where the ball bounces back from the ground. While there is more physics involved in the bouncing, a simple description is that the bounce on the ground will make the ball invert its velocity vector (from down to up) and slow down a little (since the friction loses energy). Both aspects separately, the flying and the bouncing, are easy to understand. They interact as a hybrid system, where the ball flies continuously through the air until it hits the ground where it bounces back up by a discrete jump of its velocity from negative to positive. These thoughts led us to a hybrid program model for the bouncing ball along with its specification in differential dynamic logic from [Lecture 4 on Safety & Contracts](#):

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ \left[(x' = v, v' = -g \ \& \ x \geq 0; \text{if}(x = 0) \ v := -cv)^* \right] (0 \leq x \wedge x \leq H) \quad (1)$$

A typical run of the bouncing ball program in (1) will follow an alternating succession of a continuous trajectory for the differential equation $x' = v, v' = -g$ within the evolution domain $x \geq 0$ for a certain nonzero duration and an instantaneous discrete jump following the discrete change $\text{if}(x = 0) \ v := -cv$ at a discrete instant of time. This succession gives rise to a subtlety. If we ask what value the velocity and height of the bouncing ball have at a certain point in time t_1 , chances are that that is not going to have an unambiguous answer. Whenever the ball is on the ground ($x = 0$) bouncing back up, there are two velocities we could be referring to. The negative velocity where the ball was still flying downwards and the subsequent positive velocity after the bounce which reverted direction by $v := -cv$. So if we are trying to define a trace as a function $\sigma : \mathbb{R} \rightarrow \mathcal{S}$ from real time \mathbb{R} to the state space \mathcal{S} , we will utterly fail producing a function, because the velocity values at a time t_1 with $\sigma(t)(x) = 0$ are not unique. What could we do about that?

Before you read on, see if you can find the answer for yourself.

¹Taking the usual models of air resistance into account turned out to be easy as well as we saw in [Lecture 11 on Differential Equations & Proofs](#).

The way out of this dilemma is to blow up time. Ever since [Lecture 12 on Ghosts & Differential Ghosts](#), we are used to seeing extra dimensions everywhere. Time is one of those cases where a spooky extra dimension can help clarify what is going on. Let's add a second dimension to time so that we can distinguish between the first and the second time that the ball was at the ordinary real time t_1 . The first such time (maybe denoted $t_{1.1}$) will have negative velocity, the second one (denoted $t_{1.2}$) positive velocity. Think of this as buying a pair of chronophotographic hyper-time spectacles and looking at the same world as before with a more fine-grained notion of time to discover that there is succession in things that looked indistinguishable before.²

It turns out, however, that rather than suffixing real points in time t_1 with a natural number j to form $t_{1.j}$, it is more convenient to turn it around and consider time T as a cartesian product $\mathbb{N} \times \mathbb{R}$ such that a point in time (j, t) consists of a natural number $j \in \mathbb{N}$ counting how many discrete steps have happened so far and a real number $t \in \mathbb{R}, t \geq 0$ indicating the real amount of time it took to get there.

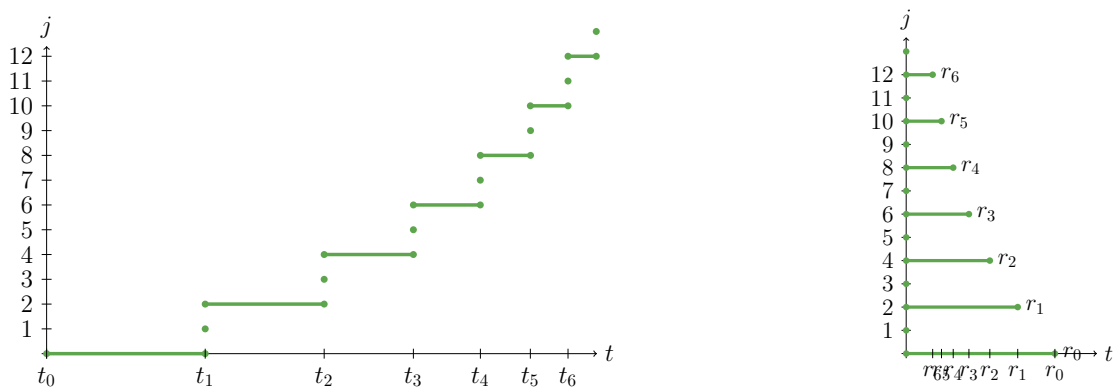


Figure 2: Two possible hybrid time domains for the sample trajectory of a bouncing ball with discrete time step j and continuous time t

This succession of continuous and discrete transitions in Fig. 1 gives rise to the hybrid time domain T shown in Fig. 2(left). Here, the intervals are either compact intervals $[t_i, t_{i+1}]$ of positive duration $t_{i+1} - t_i > 0$ during which the ball is flying through the air continuously according to the differential equation, or they are point intervals $[t_i, t_i]$ and a discrete transition happens at that single point in time that changes the sign and magnitude of the ball's velocity by a bounce described in the assignment. For example, $[t_1, t_2]$ is the time interval during which the ball is flying after its first bounce. And the point interval $[t_2, t_2]$ represents the point in time during which the subsequent discrete transition of bouncing happened, while $[t_2, t_3]$ would be the flying phase after the second bounce.

While this choice of a hybrid time domain gives a nice visual representation of the overall progress of time, the lower bound of the intervals is not particularly informa-

²This is one of the many amazing cases where we follow Wheeler's expression of Henri Poincaré's thoughts: "Time is defined so that motion looks simple" [MTW73, p. 23].

tive, because it coincides with the upper bound of the previous interval of time. It gets notationally easier if all lower bounds of all intervals are normalized to 0 and only the duration $r_i = t_{i+1} - t_i$ is retained as the upper bound; see Fig. 2(right). Both hybrid time domains in Fig. 2 are ultimately equivalent but the one on the right is easier to work with. Fig. 3 shows the particular sample trajectory of the bouncing ball from Fig. 1 plotted on its corresponding hybrid time domain T from Fig. 2(right). That illustration separates out the various discrete and continuous pieces of the trajectory of the bouncing ball into separate fragments of the two-dimensional hybrid time.

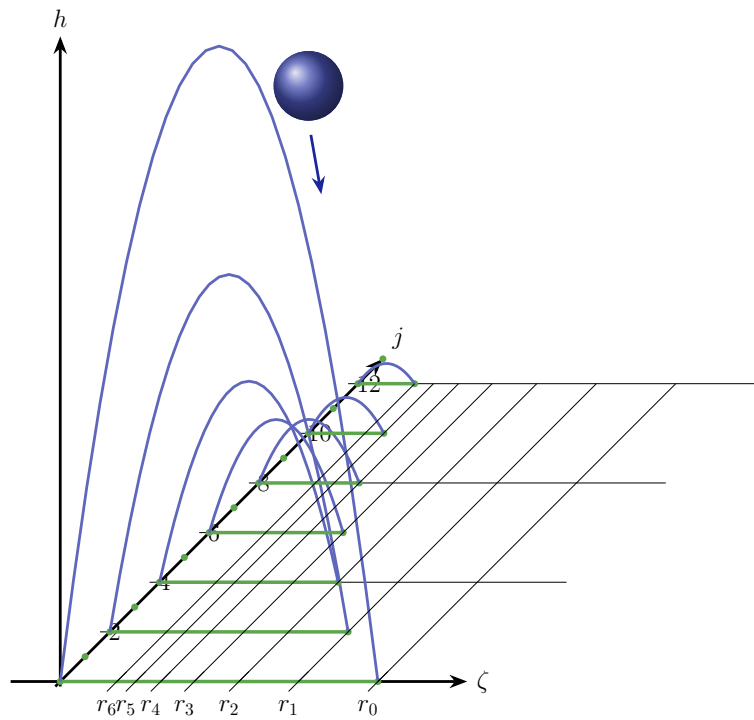


Figure 3: Sample trajectory of a bouncing ball plotted as position h over its hybrid time domain with discrete time step j and continuous time ζ

This particular illustration nicely highlights the hybrid nature of the bouncing ball dynamics. The downside, however, is that the hybrid domain T shown in Fig. 2 is specific to the particular bouncing ball trajectory from Fig. 1 and Fig. 3 and does not fit to any other bouncing ball trajectories.

Before we proceed, we illustrate two more phenomena that are worth noticing: subdivision and super-dense computations. While Fig. 2 shows one hybrid time domain for the sample trajectory in Fig. 1, there are infinitely many other hybrid time domains that fit to the original sample trajectory shown in Fig. 1 and just subdivide one of the intervals of a flying phase into two subintervals during which the ball just keeps on flying the way it did before. The first flying phase, for example, could just as well be subdivided into the continuous phase where the ball is flying up according to the dif-

ferential equation followed by a continuous phase where the ball is flying down, still according to the same differential equation. This happens whenever the continuous evolution stops before the ball was on the ground, in which case the hybrid program from (1) will loop around without actually changing any variables. That would yield a different hybrid time domain with multiple intervals of positive duration in immediate succession but still essentially the same behavior of the hybrid system in the end. So subdivision of time domains does not yield characteristically different behavior. Likewise, there can be hybrid systems that have multiple discrete steps (corresponding to point intervals in the hybrid time domain) in immediate succession before a continuous transition happens again. For example, a car could, successively, switch gears and disable the adaptive cruise control system and engage a warning light to alert the driver before it ceases control again to the continuous driving behavior. Hence, while strict alternation of discrete and continuous transitions may be the canonical example to have in mind, it is most definitely not the only relevant scenario.

5 Trace Semantics of Hybrid Programs

In differential dynamic d \mathcal{L} [Pla08, Pla12c] from Lecture 4, modalities only refer to the final states of system runs and the semantics is a reachability relation on states: State ω is reachable from state ν using system α if there is a run of α which terminates in ω when started in ν . For dTL, however, formulas can refer to intermediate states of runs as well. To capture this, we change the semantics of a hybrid system α to be the set of its possible *traces*, i.e., successions of states that occur during the evolution of α . The relation between the initial and the final state alone is not sufficient.

States define the values of system variables during a hybrid evolution. A *state* is a map $\nu : \Sigma \rightarrow \mathbb{R}$. In addition, we distinguish a separate state Λ to denote the failure of a system run when it is *aborted* due to a test $?H$ that yields *false*. In particular, Λ can only occur at the end of an aborted system run and marks that no further extension of that trace is possible because of a failed test. The set of all states is denoted by S .

Hybrid systems evolve along piecewise continuous traces in multi-dimensional space as time passes. Continuous phases are governed by differential equations, whereas discontinuities are caused by discrete jumps in state space. Unlike in discrete cases [Pra79, BS01], traces are not just sequences of states, since hybrid systems pass through uncountably many states even in bounded time. Beyond that, continuous changes are more involved than in pure real time [ACD90, HNSY92], because all variables can evolve along differential equations with different slopes. Generalizing the real-time traces of [HNSY92], the following definition captures hybrid behaviour by splitting the uncountable succession of states into periods σ_i that are regulated by the same control law. For discrete jumps, some of those periods are point flows of duration 0.

The (trace) semantics of hybrid programs is compositional, that is, the semantics of a complex program is defined as a simple function of the trace semantics of its parts. What a hybrid trace captures is the full temporal evolution of a hybrid system. Hybrid systems can behave in different ways, so their trace semantics will be a set of hybrid

traces, each of which describes one particular temporal evolution over time. Time, however, is hybridized to a pair (i, ζ) of a discrete time index $i \in \mathbb{N}$ and a real time point $\zeta \in \mathbb{R}$. A single time component $\zeta \in \mathbb{R}$ itself would be an inadequate model of time for hybrid systems, because hybrid systems can make progress by a discrete transition without continuous time passing. That happens whenever discrete controls take action. Continuous time only passes during continuous evolutions along differential equations. Discrete actions only make discrete time index i pass.

Definition 2 (Hybrid trace). A *trace* is a (nonempty) finite sequence

$$\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_n)$$

of functions $\sigma_i : [0, r_i] \rightarrow \mathcal{S}$ with their respective durations $r_i \in \mathbb{R}$ (for $i \in \mathbb{N}$). A *position* of σ is a pair (i, ζ) with $i \in \mathbb{N}, i \leq n$ and ζ in the interval $[0, r_i]$; the state of σ at (i, ζ) is $\sigma_i(\zeta)$. Positions of σ are ordered lexicographically by $(i, \zeta) \prec (j, \xi)$ iff either $i < j$, or $i = j$ and $\zeta < \xi$. Further, for a state $\nu \in \mathcal{S}$, $\hat{\nu} : [0, 0] \rightarrow \mathcal{S}; 0 \mapsto \nu$ is the *point flow* at ν with duration 0, which is only defined at the time 0 as $\hat{\nu}(0) = \nu$.

A trace *terminates* if it is a finite sequence $(\sigma_0, \sigma_1, \dots, \sigma_n)$ with $\sigma_n(r_n) \neq \Lambda$. In that case, the last state $\sigma_n(r_n)$ is denoted by *last* σ , otherwise *last* σ is undefined. The first state $\sigma_0(0)$ is denoted by *first* σ . A trace is an *error trace* if it is a finite sequence $(\sigma_0, \sigma_1, \dots, \sigma_n)$ with $\sigma_n(r_n) = \Lambda$. Error traces of hybrid programs cannot be continued any further, so if a trace has the error state Λ anywhere, then only as the last state $\sigma_n(r_n)$.

Unlike in [ACD90, HNSY92], the definition of traces also admits finite traces of bounded duration, which is necessary for compositionality of traces in $\alpha; \beta$. The semantics of hybrid programs α as the set $\tau(\alpha)$ of its possible traces depends on valuations $\llbracket \cdot \rrbracket_\nu$ of formulas and terms at intermediate states ν . The valuation of terms and interpretations of function and predicate symbols are as for real arithmetic (Lecture 4). The valuation of formulas will be defined in Def. 6. Again, we use ν_x^d to denote the *modification* that agrees with state ν on all variables except for the symbol x , which is changed to $d \in \mathbb{R}$.

Definition 3 (Trace semantics of hybrid programs). The *trace semantics*, $\tau(\alpha)$, of a hybrid program α , is the set of all its possible hybrid traces and is defined inductively as follows:

1. $\tau(x := \theta) = \{(\hat{\nu}, \hat{\omega}) : \omega = \nu \text{ except that } \llbracket x \rrbracket_{\omega} = \llbracket \theta \rrbracket_{\nu} \text{ for } \nu \in \mathcal{S}\}$
2. $\tau(x' = \theta \ \& \ H) = \{(\varphi) : \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \text{ for a solution } \varphi : [0, r] \rightarrow \mathcal{S} \text{ of any duration } r\} \cup \{(\hat{\nu}, \hat{\Lambda}) : \nu \not\models H\};$
i.e., with $\varphi(t)(x') \stackrel{\text{def}}{=} \frac{d\varphi(\zeta)(x)}{d\zeta}(t)$, φ solves the differential equation and satisfies H at all times, see [Lecture 2](#). If even the initial state ν does not satisfy H , there can be no evolution except from the current state ν to an error state Λ .
3. $\tau(?H) = \{(\hat{\nu}) : \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) : \nu \not\models H\}$
4. $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$
5. $\tau(\alpha; \beta) = \{\sigma \circ \varsigma : \sigma \in \tau(\alpha), \varsigma \in \tau(\beta) \text{ when } \sigma \circ \varsigma \text{ is defined}\};$
the composition of $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_n)$ and $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \dots, \varsigma_m)$ is

$$\sigma \circ \varsigma := \begin{cases} (\sigma_0, \dots, \sigma_n, \varsigma_0, \varsigma_1, \dots, \varsigma_m) & \text{if } \sigma \text{ terminates and last } \sigma = \text{first } \varsigma \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$

6. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$, where $\alpha^{n+1} \stackrel{\text{def}}{=} (\alpha^n; \alpha)$ for $n \geq 1$, as well as $\alpha^1 \stackrel{\text{def}}{=} \alpha$ and $\alpha^0 \stackrel{\text{def}}{=} (?true)$.

Time passes differently during discrete and continuous change. During continuous evolutions, the discrete step index i of positions (i, ζ) remains constant, whereas the continuous duration ζ remains 0 during discrete point flows. This permits multiple discrete state changes to happen at the same (super-dense) continuous time, unlike in other approaches [[ACD90](#)].

Example 4. For comparing the transition semantics of hybrid programs for \mathbf{dL} from [Lecture 3](#) and the trace semantics of hybrid programs for \mathbf{dTL} from Def. 3, consider the following simple hybrid program α :

$$a := -2a; \ a := a^2.$$

The transition semantics is just the relation between initial and final states:

$$\rho(\alpha) \equiv \{(\nu, \omega) : \omega \text{ is like } \nu \text{ except that } \omega(a) = 4\nu(a)^2\}.$$

In particular, the \mathbf{dL} formula $[\alpha]a \geq 0$ is valid, because all final states have a square as

the value of a . In contrast, the trace semantics of α retains all intermediate states:

$$\tau(\alpha) \equiv \{(\hat{\nu}, \hat{s}, \hat{\omega}) : s \text{ is like } \nu \text{ except } s(a) = -2\nu(a) \\ \text{and } \omega \text{ is like } s \text{ except } \omega(a) = s(a)^2 = 4\nu(a)^2\}.$$

During these traces, $a \geq 0$ does not hold at all states. If the trace starts with a positive value ($\nu \models a > 0$), then it will become negative at the point flow s (where $s \models a < 0$), yet recover to a positive value ($\omega \models a > 0$) at the end.

Example 5. The previous example only had discrete jumps, and, thus, the traces only involved point flows. Now consider the hybrid program β from the train context:

$$a := -b; z' = v, v' = a; ?v \geq 0; a := A; z' = v, v' = a.$$

The transition semantics of this program only considers successful runs to completion. In particular, if $A > 0$, the velocity v will always be nonnegative at the end (otherwise the test $?v \geq 0$ in the middle fails and the program aborts), because the last differential equation will accelerate and increase the velocity again. Thus, the position z at the end of the program run will never be smaller than at the beginning.

If, instead, we consider the trace semantics of β , all intermediate states are in the set of traces:

$$\begin{aligned} \tau(\beta) \equiv & \{(\hat{\mu}_0, \hat{\mu}_1, \varphi_1, \hat{\mu}_2, \hat{\mu}_3, \varphi_2) : \mu_1 = \mu_0[a \mapsto -\mu_0(b)] \text{ and} \\ & \varphi_1 \text{ is a state flow of some duration } r_1 \geq 0 \text{ with } \varphi_1 \models z' = v \wedge v' = a \\ & \text{starting in } \varphi_1(0) = \mu_1 \text{ and ending in a state with } \varphi_1(r_1)(v) \geq 0 \\ & \text{and } \mu_2 = \varphi_1(r_1), \mu_3 = \varphi_1(r_1)[a \mapsto \varphi_1(r_1)(A)] \text{ and} \\ & \varphi_2 \text{ is a state flow of some duration } r_2 \geq 0 \text{ with } \varphi_2 \models z' = v \wedge v' = a \\ & \text{starting in } \varphi_2(0) = \mu_3 \text{ and ending in state } \varphi_2(r_2)\} \\ \cup & \{(\hat{\mu}_0, \hat{\mu}_1, \varphi_1, \hat{\mu}_2, \hat{\Lambda}) : \mu_1 = \mu_0[a \mapsto -\mu_0(b)] \text{ and} \\ & \varphi_1 \text{ is a state flow of some duration } r \geq 0 \text{ with } \varphi_1 \models z' = v \wedge v' = a \\ & \text{starting in } \varphi_1(0) = \mu_1 \text{ and ending in a state with } \varphi_1(r)(v) < 0 \\ & \text{further } \mu_2 = \varphi_1(r)\}. \end{aligned}$$

The first set is the set of traces where the test $?v \geq 0$ in the middle succeeds and the system continues. The second set (after the union) is the set of traces that are aborted with $\hat{\Lambda}$ during their execution, because the middle test fails. Note that the traces in the first set have two continuous flows φ_1, φ_2 and four point flows $\hat{\mu}_0, \hat{\mu}_1, \hat{\mu}_2, \hat{\mu}_3$ in each trace. The traces in the second set have only one continuous flow φ_1 and three point flows $\hat{\mu}_0, \hat{\mu}_1, \hat{\mu}_2$, because the subsequent aborting point flow $\hat{\Lambda}$ does not terminate and aborts all further execution. In the trace semantics, $v < 0$ is possible in the middle of some traces, which is a fact that the transition semantics does not notice. Combining traces for $\alpha \cup \beta$, that is, for

$$(a := -2a; a := a^2) \cup (a := -b; z' = v, v' = a; ?v \geq 0; a := A; z' = v, v' = a)$$

is just the union $\tau(\alpha) \cup \tau(\beta)$ of the traces $\tau(\alpha)$ and $\tau(\beta)$ from Examples 4 and 5. Note that $a \leq 0$ will hold at least once during every trace of $\alpha \cup \beta$, either in the beginning, or after setting $a := -2a$ or $a := -b$, respectively, when we assume $b > 0$.

6 Semantics of State and Trace Formulas

In the semantics of dTL formulas, the dynamic modalities determine the set of traces according to the trace semantics of hybrid programs, and, independently, the temporal modalities determine at which points in time the respective postcondition needs to hold. The semantics of formulas is compositional and denotational, that is, the semantics of a complex formula is defined as a simple function of the semantics of its subformulas.

Definition 6 (dTL semantics). The *satisfaction relation* $\nu \models \phi$ for a dTL (state) formula ϕ in state ν is defined inductively:

- $\nu \models (\theta_1 = \theta_2)$ iff $\llbracket \theta_1 \rrbracket_\nu = \llbracket \theta_2 \rrbracket_\nu$.
- $\nu \models (\theta_1 \geq \theta_2)$ iff $\llbracket \theta_1 \rrbracket_\nu \geq \llbracket \theta_2 \rrbracket_\nu$.
- $\nu \models \neg\phi$ iff $\nu \not\models \phi$, i.e. if it is not the case that $\nu \models \phi$.
- $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$.
- $\nu \models \phi \vee \psi$ iff $\nu \models \phi$ or $\nu \models \psi$.
- $\nu \models \phi \rightarrow \psi$ iff $\nu \not\models \phi$ or $\nu \models \psi$.
- $\nu \models \phi \leftrightarrow \psi$ iff $(\nu \models \phi \text{ and } \nu \models \psi)$ or $(\nu \not\models \phi \text{ and } \nu \not\models \psi)$.
- $\nu \models \forall x \phi$ iff $\nu_x^d \models \phi$ for all $d \in \mathbb{R}$.
- $\nu \models \exists x \phi$ iff $\nu_x^d \models \phi$ for some $d \in \mathbb{R}$.
- $\nu \models [\alpha]\pi$ iff for each trace $\sigma \in \tau(\alpha)$ that starts in first $\sigma = \nu$, if $\llbracket \pi \rrbracket_\sigma$ is defined, then $\llbracket \pi \rrbracket_\sigma = \text{true}$.
- $\nu \models \langle \alpha \rangle \pi$ iff there is a trace $\sigma \in \tau(\alpha)$ starting in first $\sigma = \nu$ such that $\llbracket \pi \rrbracket_\sigma$ is defined and $\llbracket \pi \rrbracket_\sigma = \text{true}$.

For trace formulas, the *valuation* $\llbracket \cdot \rrbracket_\sigma$ with respect to trace σ is defined inductively as:

1. If ϕ is a state formula, then $\llbracket \phi \rrbracket_\sigma = \llbracket \phi \rrbracket_{\text{last } \sigma}$ if σ terminates, whereas $\llbracket \phi \rrbracket_\sigma$ is *not defined* if σ does not terminate.
2. $\llbracket \Box \phi \rrbracket_\sigma = \text{true}$ iff $\sigma_i(\zeta) \models \phi$ holds for all positions (i, ζ) of σ with $\sigma_i(\zeta) \neq \Lambda$.
3. $\llbracket \Diamond \phi \rrbracket_\sigma = \text{true}$ iff $\sigma_i(\zeta) \models \phi$ holds for some position (i, ζ) of σ with $\sigma_i(\zeta) \neq \Lambda$.

As usual, a (state) formula is *valid* if it is true in all states. If $\nu \models \phi$, then we say that dTL state formula ϕ is true at ν or that ν is a model of ϕ . A (state) formula ϕ is *valid*, written $\models \phi$, iff $\nu \models \phi$ for all states ν . A formula ϕ is a *consequence* of a set of formulas Γ , written $\Gamma \models \phi$, iff, for each ν : ($\nu \models \psi$ for all $\psi \in \Gamma$) implies that $\nu \models \phi$. Likewise, for trace formula π and trace σ we write $\sigma \models \pi$ iff $\llbracket \pi \rrbracket_\sigma = \text{true}$ and $\sigma \not\models \pi$ iff $\llbracket \pi \rrbracket_\sigma = \text{false}$. In particular, we only write $\sigma \models \pi$ or $\sigma \not\models \pi$ if $\llbracket \pi \rrbracket_\sigma$ is defined, which it is not the case if π is a state formula and σ does not terminate. The points where a dTL property ϕ has to hold for the various combinations of temporal and dynamic modalities are illustrated in Fig. 4.

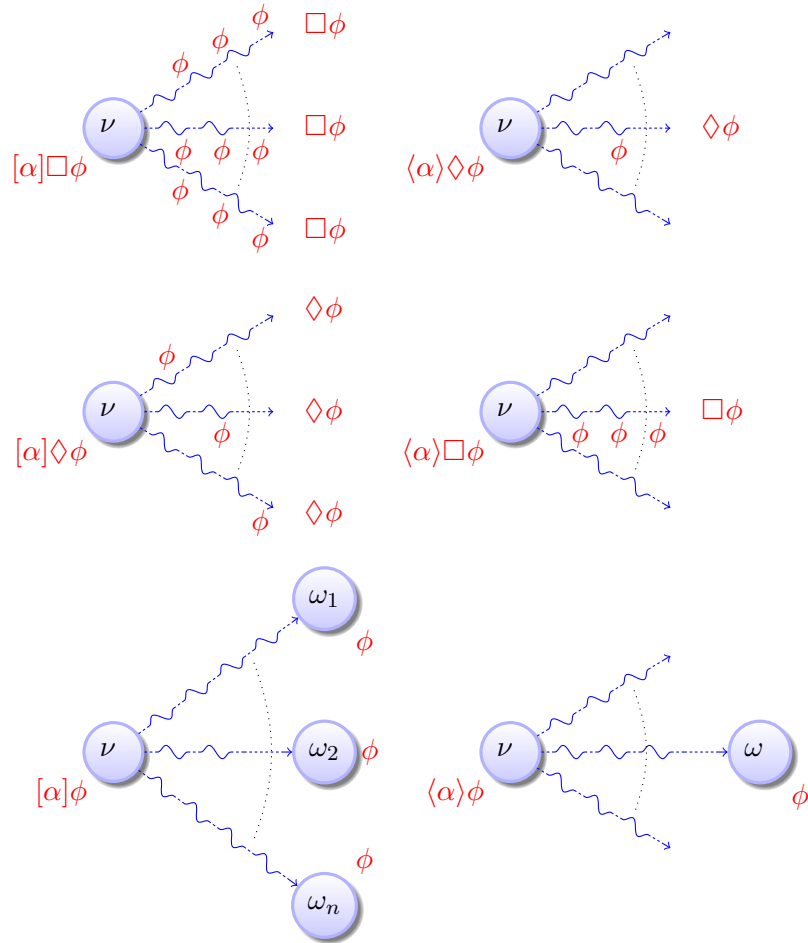


Figure 4: Trace semantics of dTL formulas

7 Conservative Temporal Extension

The following result shows that the extension by temporal operators that dTL provides does not change the meaning of nontemporal dL formulas. The trace semantics given in Def. 6 is equivalent to the final state reachability relation semantics given in Lecture 4 for the sublogic dL of dTL.

Proposition 7 (Conservative temporal extension [Pla10, Proposition 4.1]). *The logic dTL is a conservative extension of nontemporal dL, i.e., the set of valid dL formulas is the same with respect to transition reachability semantics of dL (Lecture 4) as with respect to the trace semantics of dTL (Def. 6).*

The proof is by induction using that the reachability relation fits to the trace semantics. That is, the reachability relation semantics of hybrid programs agrees with the first and last states of the traces in the trace semantics.

Lemma 8 (Trace relation [Pla10, Lemma 4.1]). *For hybrid programs α :*

$$\rho(\alpha) = \{(\text{first } \sigma, \text{last } \sigma) : \sigma \in \tau(\alpha) \text{ terminates}\}.$$

In particular, the trace semantics from today's lecture fits seamlessly to the original reachability semantics that was the basis for the previous lectures. The trace semantics exactly satisfies the objective of characterizing the same reachability relation between initial and final states, while, in addition, keeping a trace of all intermediate states around. For nontemporal dTL formulas and for dL formulas, this full trace with intermediate states is not needed, because the reachability relation between initial and final states is sufficient to define the meaning. For temporal dTL formulas, instead, the trace is crucial to give a meaning to \Box and \Diamond .

8 Summary

This lecture introduced a temporal extension of the logic dL and a trace semantics of hybrid programs. This extends the syntax and semantics to the presence of temporal modalities. The next lecture investigates how to prove temporal properties of hybrid systems. Part of the value of today's lecture was to learn about how to state temporal properties of hybrid systems in differential temporal dynamic logic. An indirect aspect is, however, that it gave us a deeper understanding of the temporal behavior of hybrid systems even in cases where we continue to operate in differential dynamic logic.

Exercises

Exercise 1. Can you give a formula of the following form that is valid?

$$[\alpha]\Box\phi \wedge \neg[\alpha]\phi$$

Exercise 2. Plot the counterpart of the sample trajectory from Fig. 3 for the alternative hybrid time domain in Fig. 2(left).

Exercise 3. In which case does the temporal $[\alpha]\Box\phi$ differ from the nontemporal $[\alpha]\phi$.

Exercise 4. Def. 3 defined the trace semantics of tests as

$$\tau(?H) = \{(\hat{\nu}) : \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) : \nu \not\models H\}$$

What would change if this definition would be modified to include an extra state $\hat{\nu}$ in the case of successful tests to record the fact that a test has happened:

$$\tau(?H) = \{(\hat{\nu}, \hat{\nu}) : \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) : \nu \not\models H\}$$

Is there a dTL formula that is true in one semantics but not in the other? Would the semantics be the same when, instead, modifying the semantics of tests to elide the initial state:

$$\tau(?H) = \{(\hat{\nu}) : \nu \models H\} \cup \{(\hat{\Lambda}) : \nu \not\models H\}$$

Is there a dTL formula that is true in one semantics but not in the other?

Exercise 5. No traces ever start in error states. Is the semantics of sequential composition the same when dropping the requirement of termination and using the following definition instead:

$$\sigma \circ \varsigma := \begin{cases} (\sigma_0, \dots, \sigma_n, \varsigma_0, \varsigma_1, \dots, \varsigma_m) & \text{if last } \sigma = \text{first } \varsigma \\ \sigma & \text{if last } \sigma = \Lambda \\ \text{not defined} & \text{otherwise} \end{cases}$$

This exercise assumes that last σ is defined as $\sigma_n(r_n)$ whether it terminates without error or with error.

Exercise 6. Is the formula (1) equivalent to the following dTL formula?

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ \left[(x' = v, v' = -g \ \& \ x \geq 0; \text{ if } (x = 0) \ v := -cv)^* \right] \Box (0 \leq x \wedge x \leq H)$$

What if the differential equation is replaced by

$$x' = v, v' = -g; ?x \geq 0$$

Are the corresponding temporal and nontemporal formulas equivalent in that case?

References

- [ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.

- [BS01] Bernhard Beckert and Steffen Schlager. A sequent calculus for first-order dynamic logic with trace modalities. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 626–641. Springer, 2001.
- [Cro00] Rod Cross. The coefficient of restitution for collisions of happy balls, unhappy balls, and tennis balls. *Am. J. Phys.*, 68(11):1025–1031, 2000. doi:[10.1119/1.1285945](https://doi.org/10.1119/1.1285945).
- [EC82] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
- [EH86] E. Allen Emerson and Joseph Y. Halpern. “Sometimes” and “Not Never” revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.
- [HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT Press, 2000.
- [HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.
- [JP14] Jean-Baptiste Jeannin and André Platzer. dTL²: Differential temporal dynamic logic with nested temporalities for hybrid systems. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR*, volume 8562 of *LNCS*, pages 292–306. Springer, 2014. doi:[10.1007/978-3-319-08587-6_22](https://doi.org/10.1007/978-3-319-08587-6_22).
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [MTW73] Charles W. Misner, Kip S. Thorne, and John Archibald Wheeler. *Gravitation*. W. H. Freeman, New York, 1973.
- [Pla07] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007. doi:[10.1007/978-3-540-72734-7_32](https://doi.org/10.1007/978-3-540-72734-7_32).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:[10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In *LICS [LIC12]*, pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:[1205.4788](https://arxiv.org/abs/1205.4788).

- [Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:10.1109/LICS.2012.13.
- [Pla14] André Platzer. Analog and hybrid computation: Dynamical systems and programming languages. *Bulletin of the EATCS*, 114, 2014. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/viewFile/292/274>.
- [Pra79] Vaughan R. Pratt. Process logic. In *POPL*, pages 93–100, 1979.

Lecture Notes on Differential & Temporal Proofs

[André Platzer](#)

Carnegie Mellon University
Lecture 17

1 Introduction

This lecture continues the study of temporal aspects of cyber-physical systems that [Lecture 16 on Differential & Temporal Logics](#) started. The trace semantics of hybrid programs as well as the semantics of differential temporal dynamic logic (dTL) [[Pla10](#)], a temporal extension of differential dynamic logic d \mathcal{L} [[Pla08](#), [Pla12](#)], have been discussed in said lecture. That was very useful, because dTL gives us a way of expressing CPS correctness properties that depend on their temporal behavior, with the most prominent one being that a CPS is supposed to be safe always at all times in the future. But that alone is not enough, unless we also find a way to verify such temporal properties of CPS to find out whether they are true, not just specify them to state what we would like to be true. This is what today's lecture focuses on: how to prove temporal properties of cyber-physical systems. Needless to say that the axiomatics of temporal properties of CPS will give us a complementary understanding of their syntactic and semantic representation from [Lecture 16](#) by the general principles of the logical trinity of syntax, semantics, and axiomatics.

This lecture is based on [[Pla10](#), Chapter 4], which extends [[Pla07](#)]. Proof rules for more general temporal properties of hybrid systems are handled elsewhere [[JP14](#)].

The most important learning goals of this lecture are:

Modeling and Control: We get a more detailed understanding for the difference between temporal and nontemporal properties of CPS models. An understanding for the temporal behavior of CPS also has an immediate impact on their simulation challenges, because a number of differences in how suitable CPS models are for simulation purposes are only reflected in their temporal behavior and corresponding temporal properties.

Computational Thinking: This lecture focuses on rigorous reasoning techniques for the temporal aspects of CPS. It also addresses subtle aspects with identifying specifications and critical properties of CPS. Since *Differential temporal dynamic logic* dTL [Pla10] extends differential dynamic logic, we can continue to use the familiar proof rules for its nontemporal parts, but need to develop new proof rules for its new temporal operators. A secondary goal in this lecture is practicing the logical trinity consisting of the relationship of syntax, semantics, and axiomatics. Temporal properties of CPS cause some new phenomena in proof rules that we have not seen before.

CPS Skills: Another secondary but useful goal of today's lecture is to develop an intuition for the question which parts of a proof (and, by duality, which parts of a system) will be affected by temporal properties. This can be helpful for identifying the relevant dynamical aspects for each part of a CPS and understanding the analytic impact of modeling tradeoffs concerning temporal behaviors of CPS.

2 Temporal Proof Rules

When extending a logic, it is not enough to extend just the syntax (Lecture 16) and semantics (Lecture 16). The third part of the logical trinity, the proof rules, also need to be extended to handle the new concepts, that is the temporal modalities of dTL.

This section shows a sequent calculus for verifying temporal specifications of hybrid systems in differential temporal dynamic logic dTL. With the basic idea being to perform a symbolic decomposition, the calculus transforms hybrid programs successively into simpler logical formulas describing their effects. Statements about the temporal behaviour of a hybrid program are successively reduced to corresponding nontemporal statements about the intermediate states. This lecture shows a proof calculus for differential temporal dynamic logic dTL that inherits the proof rules of d \mathcal{L} from previous lectures and adds new proof rules for temporal modalities.

Inherited Nontemporal Rules The dTL calculus is presented in Fig. 1 and inherits the (nontemporal) d \mathcal{L} proof rules, i.e., the propositional, first-order, dynamic, and global rules from d \mathcal{L} . That is, it includes the propositional and quantifier rules from Lecture 6. The dynamic rules ($\langle \cdot \rangle$ – $[\cdot]$) and global rules ($\llbracket gen, \langle \cdot \rangle gen, ind, con \rrbracket$) for handling nontemporal dynamic modalities are also inherited directly from Lecture 6.

The only minor additional observation is that the rules $[\cup], \langle \cup \rangle$ for nondeterministic choices can be generalized to apply to formulas of the form $[\alpha \cup \beta]\pi$ where π is an arbitrary trace formula, and not just a state formula as in d \mathcal{L} . Thus, π may begin with \Box or \Diamond , which is why the rules are repeated in this generalized form as $[\cup]\Box$ and $\langle \cup \rangle\Diamond$ in Fig. 1.

Temporal Rules The new temporal rules in Fig. 1 for the dTL calculus successively transform temporal specifications of hybrid programs into nontemporal d \mathcal{L} formulas.

Note 1.

$([\cup]\Box) \frac{[\alpha]\pi \wedge [\beta]\pi}{[\alpha \cup \beta]\pi} \text{ }^1$	$(\langle \cup \rangle \Diamond) \frac{\langle \alpha \rangle \pi \vee \langle \beta \rangle \pi}{\langle \alpha \cup \beta \rangle \pi} \text{ }^1$
$([\cdot];)\Box \frac{[\alpha]\Box\phi \wedge [\alpha][\beta]\Box\phi}{[\alpha;\beta]\Box\phi}$	$(\langle \cdot \rangle \Diamond) \frac{\langle \alpha \rangle \Diamond\phi \vee \langle \alpha \rangle \langle \beta \rangle \Diamond\phi}{\langle \alpha;\beta \rangle \Diamond\phi}$
$([\cdot]?)\Box \frac{\phi}{[?]\chi]\Box\phi}$	$(\langle \cdot \rangle?)\Diamond \frac{\phi}{\langle ?\chi \rangle \Diamond\phi}$
$([\cdot]:=)\Box \frac{\phi \wedge [x := \theta]\phi}{[x := \theta]\Box\phi}$	$(\langle \cdot \rangle :=)\Diamond \frac{\phi \vee \langle x := \theta \rangle \phi}{\langle x := \theta \rangle \Diamond\phi}$
$([\cdot]')\Box \frac{[x' = \theta]\phi}{[x' = \theta]\Box\phi}$	$(\langle \cdot \rangle')\Diamond \frac{\langle x' = \theta \rangle \phi}{\langle x' = \theta \rangle \Diamond\phi}$
$([\cdot]^*n)\Box \frac{[\alpha; \alpha^*]\Box\phi}{[\alpha^*]\Box\phi}$	$(\langle \cdot \rangle^*n)\Diamond \frac{\langle \alpha; \alpha^* \rangle \Diamond\phi}{\langle \alpha^* \rangle \Diamond\phi}$
$([\cdot]^*)\Box \frac{[\alpha^*][\alpha]\Box\phi}{[\alpha^*]\Box\phi}$	$(\langle \cdot \rangle^*)\Diamond \frac{\langle \alpha^* \rangle \langle \alpha \rangle \Diamond\phi}{\langle \alpha^* \rangle \Diamond\phi}$

¹ π is a trace formula and—unlike the state formulas ϕ and ψ —may thus begin with a temporal modality \Box or \Diamond . That is, π could be of the form ϕ or $\Box\phi$ or $\Diamond\phi$ for a state formula ϕ .

Figure 1: Axiomatization of differential temporal dynamic logic dTL

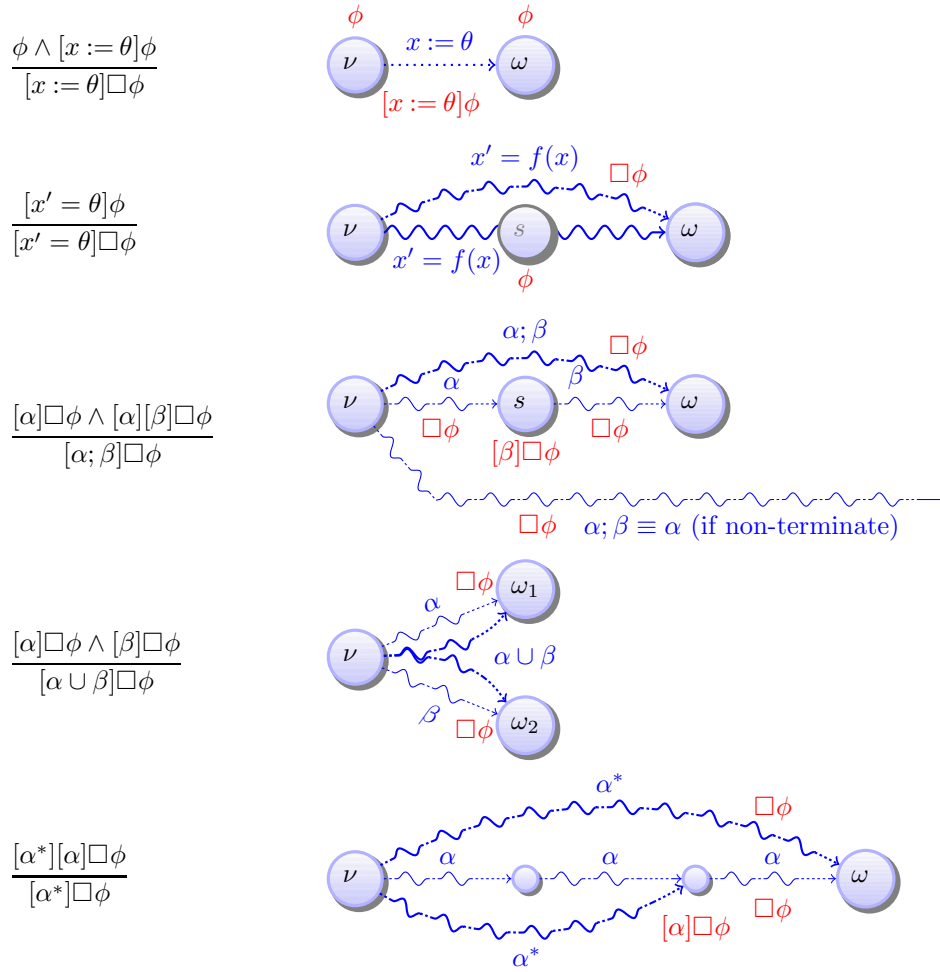


Figure 2: Correspondence of temporal proof rules and trace semantics

The idea underlying this transformation is to decompose hybrid programs and recursively augment intermediate state transitions with appropriate correctness properties that need to be shown for the original temporal property of the whole system. An illustration of the correspondence of a representative set of proof rules for temporal modalities to the trace semantics of hybrid programs (Lecture 16) is shown in Fig. 2 and will be explained in detail subsequently.

Since a property of a form $[\alpha]\Box\phi$ expresses that formula ϕ is true at all times always throughout every execution of α , we also say informally that ϕ is a (temporal) *invariant* of the hybrid program α . The reason for this name is that ϕ then plays a role somewhat similar to a loop invariant or a differential invariant, except that it holds always throughout the execution of the system.

The most fundamental rule for temporal properties is the one for sequential compositions. Rule $[\alpha; \beta]\Box\phi$ decomposes invariants of $\alpha; \beta$ (i.e., $[\alpha; \beta]\Box\phi$ holds) into a temporal

invariant that holds throughout α (i.e., $[\alpha]\Box\phi$) and a temporal invariant of β that holds when β is started in *any* final state of α (i.e., $[\alpha]([\beta]\Box\phi)$). The difference of rule $[\Box]$ compared to the $\text{d}\mathcal{L}$ rule $[\Box]$ thus is that the dTL rule $[\Box]$ also checks the safety invariant ϕ at all (symbolic) states in between the execution of α and β , and recursively in all possible intermediate states because the temporal modality \Box remains both on α (in $[\alpha]\Box\phi$) and on β (in $[\beta]\Box\phi$). See Fig. 2 for an illustration of this proof principle. As an aside, note that the same reasoning principle also works for nonterminating traces of α , because β will just never start in those cases but the subformula $[\alpha]\Box\phi$ already expresses that all nonterminating behaviors of α also satisfy ϕ all the time.

Rule $[\Box]$ expresses that invariants of assignments just need to hold before and after the discrete change, because there are no other intermediate states. The rule $[\Box]$ is similar, except that tests do not actually lead to any state change, so that ϕ holding before the test is all there is to it, because ϕ will already have to hold after the test $?H$ if it held before. Rule $[\Box]$ can directly reduce invariants of continuous evolutions to nontemporal formulas, because restrictions of solutions of differential equations are themselves solutions of different duration and thus already included in the evolutions of $x' = \theta$. If a property ϕ is true after all solutions of $x' = \theta$ of any duration, then it is also true always throughout every solution of $x' = \theta$ of every duration. In particular, observe that the handling of differential equations within hybrid systems is fully encapsulated within the fragment of dynamic rules from $\text{d}\mathcal{L}$, which makes differential invariants, differential cuts, differential weakening, and differential ghosts available for temporal properties right away.

The (optional) iteration rule $[\Box^*]$ can partially unwind loops. It relies on rule $[\Box]$ and is simpler than $\text{d}\mathcal{L}$ rule $[\Box^*]$, because the other rules will inductively produce a premise to show that ϕ holds in the current state, because of the temporal modality $\Box\phi$. The dual rules $\langle\Box\rangle, \langle\Box^*\rangle, \langle\Box\rangle, \langle\Box^*\rangle, \langle\Box\rangle, \langle\Box^*\rangle, \langle\Box\rangle, \langle\Box^*\rangle$ work similarly.

In $\text{d}\mathcal{L}$ (Lecture 7 on Control Loops & Invariants), the primary means for handling loops are the invariant induction (*ind*) and variant convergence (*con*) rules. Because dTL is built on top of $\text{d}\mathcal{L}$, the logic dTL takes a different, completely modular approach for verifying temporal properties of loops based on the $\text{d}\mathcal{L}$ capabilities for verifying nontemporal properties of loops. Rules $[\Box^*]$ and $\langle\Box^*\rangle$ actually *define* temporal properties of loops inductively. Rule $[\Box^*]$ expresses that ϕ holds at all times during repetitions of α (i.e., $[\alpha^*]\Box\phi$) iff, *after* repeating α *any* number of times, ϕ holds at all times *during one* execution of α (i.e., $[\alpha^*]([\alpha]\Box\phi)$). See Fig. 2 for an illustration. Dually, $\langle\Box^*\rangle$ expresses that α holds at some time during repetitions of α (i.e., $\langle\alpha^*\rangle\Box\phi$) iff, after some number of repetitions of α , formula ϕ holds at some point during one execution of α (i.e., $\langle\alpha^*\rangle(\langle\alpha\rangle\Box\phi)$). In this context, the nontemporal modality $\langle\alpha^*\rangle$ can be thought of as skipping over to the iteration of α during which ϕ actually occurs, as expressed by the nested dTL formula $\langle\alpha\rangle\Box\phi$.

Note 2. The inductive definition rules $[*]\Box$ and $\langle *\rangle\Diamond$ completely reduce temporal properties of loops to dTL properties of standard nontemporal dL-modalities such that standard induction (*ind*) or convergence rules (*con*) can be used for the outer nontemporal modality of the loop. Hence, after applying the inductive loop definition rules $[*]\Box$ and $\langle *\rangle\Diamond$, the standard dL loop invariant and variant rules can be used for verifying temporal properties of loops without change, except that the postcondition contains temporal modalities.

Overall, the temporal repetition rule $[*]\Box$ makes it possible to reduce a temporal property of a repetition to a nontemporal property of a repetition and a temporal property of a non-repetition, both of which are easier. Combining the temporal repetition definition rule $[*]\Box$ with the nontemporal loop induction rule *ind'* using a loop invariant φ leads to:

$$\begin{array}{c} \text{ind}' \frac{\Gamma \vdash \varphi \quad \varphi \vdash [\alpha]\varphi \quad \varphi \vdash [\alpha]\Box\phi}{\Gamma \vdash [\alpha^*][\alpha]\Box\phi} \\ [*]\Box \frac{}{\Gamma \vdash [\alpha^*]\Box\phi} \end{array}$$

Note that all proof rules for atomic hybrid programs α (i.e., for an α of the form $x := \theta$, $?H$, or $x' = \theta$) could be summarized into a single rule:

$$\frac{\phi \wedge [\alpha]\phi}{[\alpha]\Box\phi} \quad \frac{\phi \vee \langle \alpha \rangle \phi}{\langle \alpha \rangle \Diamond\phi} \quad \text{for atomic } \alpha \quad (1)$$

because the atomic hybrid programs have no temporal behavior that is not already captured at the initial or final states. But the separate proof rules are more efficient, because they leave out parts that are already implied (the postcondition check in the case of $[?]\Box, \langle ? \rangle\Diamond$ and the precondition check in the case of $[']\Box, \langle ' \rangle\Diamond$).

Even though not necessary, other rules generalize to the temporal case as well such as the generalization rule $[gen]$ whose temporal counterpart is:

$$(\text{gen}\Box) \frac{\psi \vdash \phi}{[\alpha]\Box\psi \vdash [\alpha]\Box\phi}$$

Similarly for the practical form $[gen']$ of the generalization rule with the temporal counterpart:

$$(\text{gen}\Box) \frac{\Gamma \vdash [\alpha]\Box\psi, \Delta \quad \psi \vdash \phi}{\Gamma \vdash [\alpha]\Box\phi, \Delta}$$

Rules for handling $[\alpha]\Diamond\phi$ and $\langle \alpha \rangle\Box\phi$ are briefly discussed in [Pla10] and elaborated in much more detail along with many extensions to more general temporal formulas elsewhere [JP14]. The core challenge is that there is no obvious analogue of the compositional proof rules $[\cdot]\Box, \langle \cdot \rangle\Diamond$ for the alternation cases $[\alpha]\Diamond\phi$ and $\langle \alpha \rangle\Box\phi$.

3 Temporal Bouncing Ball

Recall the bouncing ball that has served us so well in [previous lectures](#).

$$(2gx \leq 2gH - v^2 \wedge x \geq 0 \wedge g > 0 \wedge H \geq 0 \wedge 1 \geq c \geq 0) \rightarrow [ball^*](0 \leq x \leq H). \quad (2)$$

Use the following abbreviations and the invariant φ from [Lecture 11](#) refining the invariant [Lecture 7](#):

$$\begin{aligned} \text{ball} &\equiv x' = v, v' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)) \\ \Gamma &\equiv g > 0 \wedge H \geq 0 \wedge 1 \geq c \geq 0, \\ \varphi &\equiv 2gx \leq 2gH - v^2 \wedge x \geq 0 \\ \langle x := ..(t) \rangle F &\equiv \langle x := x + vt - \frac{g}{2}t^2; v := v - tg \rangle F. \end{aligned}$$

When simplifying the *ball* dynamics to remove evolution domain constraints:

$$x' = v, v' = -g; (?x > 0 \cup (?x = 0; v := -cv))$$

the proof for the simplified bouncing ball property without evolution domain constraint is shown in [Fig. 3](#). The $\text{d}\mathcal{L}$ proof for the original non temporal bouncing ball property (2) with an evolution domain constraint is shown in [Fig. 4](#). Note that $\langle x := ..(t) \rangle F$ is used as an update in the proof, i.e., proof rules are applied directly to its postcondition F leaving the update around.

4 Verification Example

Recall the bouncing ball. The proofs from previous lectures or [Fig. 4](#) can be generalized easily to a proof of the temporal property

$$\begin{aligned} 2gx \leq 2gH - v^2 \wedge x \geq 0 \wedge g > 0 \wedge H \geq 0 \wedge 1 \geq c \geq 0 \\ \rightarrow [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \Box (0 \leq x \leq H). \end{aligned} \quad (3)$$

The only aspect of the proof that changes is that the temporal proof rules in [Fig. 1](#) are used instead of the dynamic proof rules for $\text{d}\mathcal{L}$, and that the resulting extra proof goals for the invariance property at intermediate steps have to be proven.

In contrast, the proof in [Fig. 3](#) for the simplified dynamics without evolution domain restriction $x \geq 0$ cannot be generalized to a proof of the temporal property

$$\begin{aligned} 2gx \leq 2gH - v^2 \wedge x \geq 0 \wedge g > 0 \wedge H \geq 0 \wedge 1 \geq c \geq 0 \\ \rightarrow [(x'' = -g; (?x > 0 \cup (?x = 0; v := -cv)))^*] \Box (0 \leq x \leq H). \end{aligned} \quad (4)$$

because the premise $\Gamma, \varphi \vdash [x'' = -g] \varphi$ resulting from the bottom-most occurrence of a sequential composition rule cannot be shown. This difference in provability is for good reasons. The property in (3) is valid, but the property in (4) is not! While there was no noticeable semantical difference between the nontemporal $\text{d}\mathcal{L}$ counterparts of the properties (3) versus (4), there is a decisive difference between the corresponding temporal properties (4) and (3). Because there is no evolution domain restriction in (4), its hybrid program does not prevent continuous evolution to a negative height under the floor ($x < 0$), for which $0 \leq x \leq H$ does not hold.

$$\begin{array}{c}
\text{iv} \frac{*}{\Gamma, \varphi, s \geq 0, x + vs - \frac{g}{2}s^2 = 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (-cv + cgs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0} \\
\langle := \rangle \frac{}{\Gamma, \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle \langle v := -cv \rangle \quad \varphi} \\
[:=] \frac{}{\Gamma, \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle [v := -cv] \quad \varphi} \\
\rightarrow r \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle (x = 0 \rightarrow [v := -cv] \quad \varphi)} \\
[?] \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0][v := -cv] \quad \varphi} \\
[i] \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \quad \varphi}
\end{array}$$

$$\begin{array}{c}
\text{iv} \frac{*}{\Gamma, \varphi, s \geq 0, x + vs - \frac{g}{2}s^2 > 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (v - gs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0} \\
\langle := \rangle \frac{}{\Gamma, \varphi, s \geq 0, \langle x := ..(s) \rangle x > 0 \vdash \langle x := ..(s) \rangle \quad \varphi} \\
\rightarrow r \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle (x > 0 \rightarrow \quad \varphi)} \\
[?] \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \quad \varphi}
\end{array}$$

$$\begin{array}{c}
\text{Ar} \frac{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \quad \varphi \quad \Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \quad \varphi}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle ([?x > 0] \quad \varphi \wedge [?x = 0; v := -cv] \quad \varphi)} \\
[\cup] \frac{}{\Gamma, \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \quad \varphi} \\
\rightarrow r \frac{}{\Gamma, \varphi \vdash s \geq 0 \rightarrow \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \quad \varphi} \\
\forall r \frac{}{\Gamma, \varphi \vdash \forall t \geq 0 \langle x := ..(t) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \quad \varphi} \\
['] \frac{}{\Gamma, \varphi \vdash [x'' = -g][?x > 0 \cup (?x = 0; v := -cv)] \quad \varphi} \\
[i] \frac{}{\Gamma, \varphi \vdash [x'' = -g; (?x > 0 \cup (?x = 0; v := -cv))] \quad \varphi} \\
ind' \frac{}{\Gamma, \varphi \vdash [(x'' = -g; (?x > 0 \cup (?x = 0; v := -cv)))^*] \quad (0 \leq x \leq H)} \\
\rightarrow r, \wedge l \frac{}{\vdash \Gamma \wedge \varphi \rightarrow [(x'' = -g; (?x > 0 \cup (?x = 0; v := -cv)))^*] \quad (0 \leq x \leq H)}
\end{array}$$

Figure 3: Non-temporal bouncing ball proof (no evolution domain)

$$\begin{array}{c}
\text{iv} \quad \frac{*}{\Gamma \wedge \varphi, s \geq 0, x + vs - \frac{g}{2}s^2 = 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (-cv + cgs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0} \\
\text{<:=>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle \langle v := -cv \rangle \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle [v := -cv] \varphi} \\
\text{<:=>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle [v := -cv] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle (x = 0 \rightarrow [v := -cv] \varphi)} \\
\text{<?>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0] [v := -cv] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \varphi} \\
\text{[i]} \quad \Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \varphi
\end{array}$$

$$\begin{array}{c}
\text{iv} \quad \frac{*}{\Gamma \wedge \varphi, s \geq 0, x + vs - \frac{g}{2}s^2 > 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (v - gs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0} \\
\text{<:=>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x > 0 \vdash \langle x := ..(s) \rangle \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle (x > 0 \rightarrow \varphi)} \\
\text{<?>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \varphi}
\end{array}$$

$$\begin{array}{c}
\text{...} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle [?x > 0] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \varphi} \quad \frac{\text{...} \quad \Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle ([?x > 0] \varphi \wedge [?x = 0; v := -cv] \varphi)} \\
\text{<?>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \varphi \wedge [?x = 0; v := -cv] \varphi}{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \varphi} \\
\text{<?>} \quad \frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \varphi}{\Gamma \wedge \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle x \geq 0 \rightarrow \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \varphi} \\
\text{<?>} \quad \frac{\Gamma \wedge \varphi \vdash s \geq 0 \rightarrow (\langle x := ..(s) \rangle x \geq 0 \rightarrow \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \varphi)}{\Gamma \wedge \varphi \vdash \forall t \geq 0 (\langle x := ..(t) \rangle x \geq 0 \rightarrow \langle x := ..(t) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \varphi)} \\
\text{[i]} \quad \frac{\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0] [?x > 0 \cup (?x = 0; v := -cv)] \varphi}{\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \varphi} \\
\text{[i]} \quad \frac{\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \varphi}{\Gamma \wedge \varphi \vdash [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \quad (0 \leq x \leq H)} \\
\text{ind'} \quad \frac{\Gamma \wedge \varphi \vdash [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \quad (0 \leq x \leq H)}{\vdash \Gamma \wedge \varphi \rightarrow [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \quad (0 \leq x \leq H)}
\end{array}$$

Figure 4: Nontemporal bouncing ball proof (with evolution domain)

The reason for this discrepancy of the temporal version compared to the nontemporal versions thus is that the nontemporal modalities do not “see” the temporary violation of $0 \leq x \leq H$. Such a temporary violation of $0 \leq x$ during the continuous evolution does not produce a successful run of the hybrid program, because it is blocked by the subsequent tests $?x = 0$ and $?x > 0$. A state with negative height fails both tests. While this behaviour does not give a successful program transition of $(\nu, \omega) \in \rho(\text{ball})$ by [Lecture 3](#) so that the proof in [Fig. 3](#) is correct, the behaviour still gives a valid trace $\sigma \in \tau(\text{ball})$ by [Lecture 16](#). This trace σ is a partial trace, because it ends in a failure state Λ , but it is still one of the traces that $[\text{ball}] \Box (0 \leq x \leq H)$ quantifies over (quite unlike $[\text{ball}](0 \leq x \leq H)$, which only considers final states of successful traces).

The proof of the temporal bouncing ball formula (4) is shown in [Fig. 5](#), which is a direct counterpart of the nontemporal proof in [Fig. 4](#). Some additional premises are elided for space reasons (marked as \triangleleft). The bottom-most use of the rule [gen](#) \Box has an additional premise $\varphi \rightarrow 0 \leq x \leq H$, which proves easily. The bottom-most use of rule [\[;\]](#) \Box has an additional premise which proves as follows:

$$\frac{\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0] \varphi}{[\text{?}] \Box \Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0] \Box \varphi}$$

making crucial use of the evolution domain constraint $x \geq 0$. The use of rule [\[;\]](#) \Box near the top has an additional premise proving as follows:

$$\frac{\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle \varphi}{[\text{?}] \Box \Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [\text{?} x = 0] \Box \varphi}$$

Finally, the top-most use of the $\langle := \rangle$ rule has a second premise for proving the left conjunct, which is once again:

$$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle \varphi$$

Finally note that the temporal proof in [Fig. 5](#) was conducted to retain the closest possible resemblance with the nontemporal proof in [Fig. 4](#). But the temporal induction rule [\[*\]](#) \Box followed by the nontemporal induction rule [ind'](#) does not actually require a proof of the loop invariant φ to hold throughout one iteration of the loop. It would have been sufficient to prove

$$\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \Box (0 \leq x \leq H)$$

directly without generalizing the temporal postcondition $\Box (0 \leq x \leq H)$ to $\Box \varphi$ via [gen](#) \Box . That would have led to a proof of the same shape as in [Fig. 5](#), just without [gen](#) \Box and with easier arithmetic in the end, because $0 \leq x \leq H$ is only a linear constraint on position.

Observe how the temporal analogue of the proof in [Fig. 4](#) shows how every intermediate symbolic state in the hybrid program for the bouncing ball is checked for safety as illustrated by the checkpoint symbol \wedge in the following hybrid program:

$$\wedge (\wedge x' = v, v' = -g \ \& \ x \geq 0 \wedge; \wedge (\wedge ?x > 0 \cup (\wedge ?x = 0; \wedge v := -cv \wedge)) \wedge)^*$$

	*
iV	$\Delta \Gamma \wedge \varphi, s \geq 0, x + vs - \frac{g}{2}s^2 = 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (-cv + cgs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0$
$\langle := \rangle$	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle (\varphi \wedge \langle v := -cv \rangle \varphi)$
$[:]=\Box$	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x = 0 \vdash \langle x := ..(s) \rangle [v := -cv] \Box \varphi$
$\rightarrow r$	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle (x = 0 \rightarrow [v := -cv] \Box \varphi)$
[?]	$\Delta \Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0] [v := -cv] \Box \varphi$
[;]□	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x = 0; v := -cv] \Box \varphi$
	*
iV	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash 2g(H - (x + vs - \frac{g}{2}s^2)) \leq 2gH - (v - gs)^2 \wedge (H - (x + vs - \frac{g}{2}s^2)) \geq 0$
$\langle := \rangle$	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle \varphi$
[?]□	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0] \Box \varphi$
	...
$\wedge r$	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle ([?x > 0] \Box \varphi \wedge [?x = 0; v := -cv] \Box \varphi)$
[U]□	$\Gamma \wedge \varphi, s \geq 0, \langle x := ..(s) \rangle x \geq 0 \vdash \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \Box \varphi$
$\rightarrow r$	$\Gamma \wedge \varphi, s \geq 0 \vdash \langle x := ..(s) \rangle x \geq 0 \rightarrow \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \Box \varphi$
$\rightarrow r$	$\Gamma \wedge \varphi \vdash s \geq 0 \rightarrow (\langle x := ..(s) \rangle x \geq 0 \rightarrow \langle x := ..(s) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \Box \varphi)$
$\forall r$	$\Gamma \wedge \varphi \vdash \forall t \geq 0 (\langle x := ..(t) \rangle x \geq 0 \rightarrow \langle x := ..(t) \rangle [?x > 0 \cup (?x = 0; v := -cv)] \Box \varphi)$
[']	$\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0] [?x > 0 \cup (?x = 0; v := -cv)] \Box \varphi$
[;]□	$\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \Box \varphi$
gen□	$\Gamma \wedge \varphi \vdash [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \Box (0 \leq x \leq H)$
ind'	$\Gamma \wedge \varphi \vdash [ball^*] [x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv))] \Box (0 \leq x \leq H)$
[*]□	$\Gamma \wedge \varphi \vdash [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \Box (0 \leq x \leq H)$
$\rightarrow r$	$\vdash \Gamma \wedge \varphi \rightarrow [(x'' = -g \ \& \ x \geq 0; (?x > 0 \cup (?x = 0; v := -cv)))^*] \Box (0 \leq x \leq H)$

Figure 5: Temporal bouncing ball proof (with evolution domain)

This is another reminder why temporal proofs will check some states repeatedly. There are ways of avoiding such redundancy in the proofs to simplify the computational complexity, but they increase the conceptual complexity of the proof rules, which makes them useful for automation but not necessarily for humans [JP14].

5 Summary

This lecture showed a systematic way of specifying and verifying temporal properties of hybrid systems. The focus was on safety properties that hold always throughout the evolution of the system and are specified as $[\alpha]\Box\phi$ with a mix of a temporal and a dynamic modality instead of just a dynamic modality as in $[\alpha]\phi$. The difference is that $[\alpha]\Box\phi$ includes that safety condition ϕ holds at all intermediate states during all traces of α , whereas $[\alpha]\phi$ only specifies that ϕ holds at the end of each trace of α . This difference matters in systems that have more intermediate states than final states. The difference is insignificant for systems that can “stop anytime”, because those will already include all intermediate states of longer system runs as the final state of a corresponding shorter system run. This has been the case in almost all systems studied in this course and is frequently the case in practice.

The systematic way of ensuring safety always throughout the execution of hybrid systems is the use of the dynamic and temporal modality $[\alpha]\Box\phi$, which works whether or not the system has the special structure that allows it to stop anytime. In a nutshell, the temporal proof rules for $[\alpha]\Box\phi$ properties lead to additional branches that correspond to the safety conditions at the respective intermediate state. It can be shown that temporal dTL properties reduce to nontemporal dL properties completely [Pla10, Chapter 4], justifying the intimate relation of temporal and nontemporal properties. That completeness result made crucial use of the clever form of the $[\ast]\Box$ proof rule.

Properties whose temporal counterpart does not hold such as (4) not only indicate that the safety property does not hold throughout the system, but also that the systems might be hard to simulate, because they can run into unsafe dead ends during execution.

Other temporal modalities are more involved and discussed elsewhere [JP14].

Exercises

Exercise 1. Can you give a formula of the following form that is valid?

$$[\alpha]\Box\phi \wedge \neg[\alpha]\phi$$

Exercise 2. Can you give a temporal box version of the differential invariant proof rule?

Exercise 3. The proof rules in (1) were argued to hold for any atomic hybrid program α . Yet, differential equations with evolution domain constraints were not captured in Fig. 1. Is the case where α is of the form $x' = \theta \ \& \ H$ sound for (1)? Justify why and correct the statement if necessary.

Exercise 4. Augment the nontemporal proof shown in Fig. 4 to a proof of (3). Which of the steps fails when trying to turn Fig. 3 into a proof attempt of (4) and why does that happen?

Exercise 5. Which of the following proof rule attempts are sound? Discuss carefully.

$$\frac{\Gamma \vdash \varphi \quad \varphi \vdash [\alpha]\varphi \quad \varphi \vdash \phi}{\Gamma \vdash [\alpha^*]\Box\phi} \quad \frac{\Gamma \vdash \varphi \quad \varphi \vdash [\alpha]\varphi \quad \varphi \vdash \Box\phi}{\Gamma \vdash [\alpha^*]\Box\phi} \quad \frac{\Gamma \vdash \varphi \quad \varphi \vdash [\alpha]\Box\varphi \quad \varphi \vdash \phi}{\Gamma \vdash [\alpha^*]\Box\phi}$$

Exercise 6. In which case does the temporal $[\alpha]\Box\phi$ differ from the nontemporal $[\alpha]\phi$. Hint: consider a number of different forms that α could have.

References

- [JP14] Jean-Baptiste Jeannin and André Platzer. dTL²: Differential temporal dynamic logic with nested temporalities for hybrid systems. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR*, volume 8562 of *LNCS*, pages 292–306. Springer, 2014. doi:10.1007/978-3-319-08587-6_22.
- [Pla07] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007. doi:10.1007/978-3-540-72734-7_32.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.
- [Pla12] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. doi:10.1109/LICS.2012.13.

Lecture Notes on Virtual Substitution & Real Equations

André Platzer

Carnegie Mellon University
Lecture 18

1. Introduction

Cyber-physical systems are important technical concepts for building better systems around us. Their safe design requires careful specification and verification, which this course provides using differential dynamic logic and its proof calculus [Pla08, Pla10, Pla12b]. The proof calculus for differential dynamic logic has a number of powerful axioms and proof rules (especially in [Lecture 5](#), [Lecture 6](#), [Lecture 11](#), and [Lecture 12](#)). In theory, the *only* difficult problem in proving hybrid systems safety is finding their invariants or differential invariants [Pla08, Pla12a] ([Lecture 13 on Differential Invariants & Proof Theory](#)). In practice, however, the handling of real arithmetic is another challenge that you have faced in your labs, even though the problem is easier in theory. How arithmetic interfaces with proofs by way of the proof rules \forall, \exists has already been discussed in [Lecture 6 on Truth & Proof](#). But how does the handling of real arithmetic by quantifier elimination really work?

Today's lecture shows one technique for deciding interesting formulas of first-order real arithmetic. Understanding how such techniques for real arithmetic work is interesting for at least two reasons. First of all, it is important to understand why this miracle happens at all that something as complicated and expressive as first-order logic of real arithmetic is decidable. But this lecture is also helpful to get an intuition about how real arithmetic decision procedures work. With such an understanding, you are better prepared to identify the limitations of these techniques, learn when they are likely not to work out in due time, and get a sense of what you can do to help arithmetic prove more complicated properties. For complex proofs, it is often very important to use your insights and intuitions about the system to help the prover along to scale your verification results to more challenging systems in feasible amounts of time. An understanding how arithmetic decision procedures work helps to focus such insights on the parts of

the arithmetic analysis that has a big computational impact. Quite substantial impact has been observed for handling the challenges of real arithmetic [Pla07, dMP13].

There are a number of different approaches to understanding real arithmetic and its decision procedures besides Tarski's original seminal breakthrough [Tar51]. There is an algebraic approach using cylindrical algebraic decompositions [Col75], which leads to practical procedures, but is highly nontrivial. There are simple and elegant model-theoretic approaches using semantic properties of logic and algebra [Rob77], which are easy to understand, but do not lead to any particularly useful algorithms. There is a reasonably simple Cohen-Hörmander algorithm [Coh69, Hör83] that, unfortunately, does not generalize well into a practical algorithm. Other simple but inefficient decision procedures are also described elsewhere [KK71, Eng93]. And there is virtual substitution [Wei97], a syntactical approach that fits well to the understanding of logic that we have developed in this course and leads to highly efficient algorithms (although not in the most general cases). As a good compromise of accessibility and practicality, this lecture focuses on virtual substitution [Wei97].

These lecture notes are loosely based on [Wei97, Pla10, Appendix D]. They add substantial intuition and motivation that is helpful for following the technical development. More information about virtual substitution can be found in the literature [Wei97]. See, e.g., [PQR09, Pas11] for an overview of other techniques for real arithmetic.

The most important learning goals of this lecture are:

Modeling and Control: This lecture has an indirect impact on CPS models and controls by informing the reader about the consequences of the analytic complexity resulting from different arithmetical modeling tradeoffs. There is always more than one way of writing down a model. It becomes easier to find the right tradeoffs for expressing a CPS model with some knowledge of and intuition for the working principles of the workhorse of quantifier elimination that will handle the resulting arithmetic.

Computational Thinking: The primary purpose of today's lecture is to understand how arithmetical reasoning, which is crucial for CPS, can be done rigorously and automatically. Developing an intuition for the working principles of real arithmetic decision procedures can be very helpful for developing strategies to verify CPS models at scale. The lecture also serves the purpose of learning to appreciate the miracle that quantifier elimination in real arithmetic provides by contrasting it with closely related problems that have fundamentally different challenges. We will also see a conceptually very important device in the logical trinity: the flexibility of moving back and forth between syntax and semantics at will. We have seen this principle in action already in the case of differential invariants in [Lecture 10 on Differential Equations & Differential Invariants](#), where we moved back and forth between analytic differentiation $\frac{d}{dt}$ and syntactic derivations $(\cdot)'$ by way of the derivation lemma and the differential substitution lemma as we saw fit. This time, we leverage the same conceptual device for real arithmetic (rather than differential arithmetic) by working with virtual substitutions to bridge the gap between semantic operations that are inexpressible otherwise in first-order logic

of real arithmetic. Virtual substitutions will again allow us to move back and forth at will between syntax and semantics.

CPS Skills: The author is not aware of any impact that this lecture has on CPS skills, because the primary point of this lecture is to get a sense of the pragmatics of CPS analysis.

2. Framing the Miracle

First-order logic is an expressive logic in which many interesting properties and concepts can be expressed, analyzed, and proven. It is certainly significantly more expressive than propositional logic, which is decidable by NP-complete SAT solving.

In classical (uninterpreted) *first-order logic* (FOL), no symbol (except possibly equality) has a special meaning. There are only predicate symbols p, q, r, \dots and function symbols f, g, h, \dots whose meaning is subject to interpretation. And the domain that quantifiers range over is subject to interpretation. In particular, a formula of first-order logic is only valid if it holds true for all interpretations of all predicate and function symbols and all domains.

In contrast, *first-order logic of real arithmetic* ($\text{FOL}_{\mathbb{R}}$ or the theory of real-closed field arithmetic FOL_{RCF} [Pla10, Appendix D]) is interpreted, because its symbols have a special fixed interpretation. The only predicate symbols are $=, \geq, >, \leq, <, \neq$ and they mean exactly equality, greater-or-equals, greater-than, etc., and the only function symbols are $+, -, \cdot$, which mean exactly addition, subtraction, and multiplication of real numbers. Furthermore, the quantifiers quantify over the set \mathbb{R} of all real numbers.¹

The first special interpretation for symbols that comes to mind may not necessarily be the real numbers but maybe the natural numbers \mathbb{N} with $+$ for addition and \cdot for multiplication on natural numbers and where quantifiers range over the natural numbers. That gives the *first-order logic of natural numbers* ($\text{FOL}_{\mathbb{N}}$). Is $\text{FOL}_{\mathbb{N}}$ easier or harder than FOL? How do both compare to $\text{FOL}_{\mathbb{R}}$? What would happen compared to $\text{FOL}_{\mathbb{Q}}$, the first-order logic of rational numbers? $\text{FOL}_{\mathbb{Q}}$ is like $\text{FOL}_{\mathbb{R}}$ and $\text{FOL}_{\mathbb{N}}$, except that the rational numbers \mathbb{Q} are used as the domain of quantification and interpretation of variables, rather than \mathbb{R} and \mathbb{N} , respectively. How do those different flavors of first-order logic compare? How difficult is it to prove validity of logical formulas in each case?

Before you read on, see if you can find the answer for yourself.

¹Respectively over another real-closed field, but that has been shown not to change validity [Tar51].

Uninterpreted first-order logic FOL is semidecidable, because there is a (sound and complete [Göd30]) proof procedure that is algorithmic and able to prove all true sentences of first-order logic [Her30]. The natural numbers are more difficult. Actually much more difficult! By Gödel's incompleteness theorem [Göd31], first-order logic $\text{FOL}_{\mathbb{N}}$ of natural numbers does not have a sound and complete effective axiomatization. $\text{FOL}_{\mathbb{N}}$ is neither semidecidable nor cosemidecidable [Chu36]. There is neither an algorithm that can prove all valid formulas of $\text{FOL}_{\mathbb{N}}$ nor one that can disprove all formulas of $\text{FOL}_{\mathbb{N}}$ that are not valid. One way of realizing the inherent challenge of the logic of natural numbers in retrospect is to use that not all questions about programs can be answered effectively (for example the halting problem of Turing machines is undecidable) [Chu36, Tur37], in fact "none" can [Ric53], and then encode questions about classical programs into the logic of natural numbers.

Yet, a miracle happened. Alfred Tarski proved in 1930 [Tar31, Tar51] that reals are much better behaved and that $\text{FOL}_{\mathbb{R}}$ is decidable, even though this seminal result remained unpublished for many years and only appeared in full in 1951 [Tar51].

The first-order logic $\text{FOL}_{\mathbb{Q}}$ of rational numbers, however, was shown to be undecidable [Rob49], even though rational numbers may appear to be so close to real numbers. Rationals are lacking something important: completeness (in the topological sense). The square root $\sqrt{2}$ of 2 is a perfectly good witness for $\exists x x^2 = 2$ but only a real number, not a rational one.

The first-order logic $\text{FOL}_{\mathbb{C}}$ of complex numbers, though, is again perfectly decidable [Tar51, CC56].

Note 1 (The miracle of reals. Overview of validity problems of first-order logics).

Logic	Validity
FOL	<i>semidecidable</i>
$\text{FOL}_{\mathbb{N}}$	<i>not semidecidable nor cosemidecidable</i>
$\text{FOL}_{\mathbb{Q}}$	<i>not semidecidable nor cosemidecidable</i>
$\text{FOL}_{\mathbb{R}}$	<i>decidable</i>
$\text{FOL}_{\mathbb{C}}$	<i>decidable</i>

3. Quantifier Elimination

Alfred Tarski's seminal insight for deciding real arithmetic is based on quantifier elimination, i.e. the successive elimination of quantifiers from formulas so that the remaining formula is equivalent but structurally significantly easier, because it has less quantifiers. Why does eliminating quantifiers help? When evaluating a logical formula for whether it is true or false in a given state (i.e. an assignment of real numbers to all its free variables), arithmetic comparisons and polynomial terms are easy, because all we need to do is plug the numbers in and compute according to their semantics (recall [Lecture 2 on Differential Equations & Domains](#)). For example, for a state ν with $\nu(x) = 2$, we can

easily evaluate the logical formula

$$x^2 > 2 \wedge 2x < 3 \vee x^3 < x^2$$

to *false* by following the semantics, which ultimately just plugs in 2 for x :

$$\llbracket x^2 > 2 \wedge 2x < 3 \vee x^3 < x^2 \rrbracket_\nu = 2^2 > 2 \wedge 2 \cdot 2 < 3 \vee 2^3 < 2^2 = \text{false}$$

Similarly, in a state ω with $\omega(x) = -1$, the same formula evaluates to *true*:

$$\llbracket x^2 > 2 \wedge 2x < 3 \vee x^3 < x^2 \rrbracket_\omega = (-1)^2 > 2 \wedge 2 \cdot (-1) < 3 \vee (-1)^3 < (-1)^2 = \text{true}$$

But quantifiers are a difficult matter, because they require us to check for all possible values of a variable (in the case $\forall x F$) or to find exactly the right value for a variable that makes the formula true (in the case of $\exists x F$). The easiest formulas to evaluate are the ones that have no free variables (because then their value does not depend on the state ν) and that also have no quantifiers (because then there are no choices for the values of the quantified variables during the evaluation). Quantifier elimination can take a logical formula that is closed, i.e. has no free variables, and equivalently remove its quantifiers, so that it becomes easy to evaluate the formula to *true* or *false*. Quantifier elimination also works for formulas that still have free variables. Then it will eliminate all quantifiers in the formula but the original free variables will remain in the resulting formula, unless it simplifies in the quantifier elimination process.

Definition 1 (Quantifier elimination). A first-order theory admits *quantifier elimination* if, with each formula ϕ , a quantifier-free formula $\text{QE}(\phi)$ can be associated effectively that is equivalent, i.e. $\phi \leftrightarrow \text{QE}(\phi)$ is valid (in that theory).

That is, a first-order theory that admits quantifier elimination if there is a computer program that outputs a quantifier-free formula $\text{QE}(\phi)$ for any input formula ϕ in that theory such that the input and output are equivalent ($\phi \leftrightarrow \text{QE}(\phi)$ is valid) and such that the output $\text{QE}(\phi)$ is quantifier-free.

Theorem 2 (Tarski [Tar51]). *The first-order logic of real arithmetic admits quantifier elimination and is, thus, decidable.*

The operation QE is further assumed to evaluate ground formulas (i.e., without variables), yielding a decision procedure for closed formulas of this theory (i.e., formulas without free variables). For a closed formula ϕ , all it takes is to compute its quantifier-free equivalent $\text{QE}(\phi)$ by quantifier elimination. The closed formula ϕ is closed, so has no free variables or other free symbols, and neither will $\text{QE}(\phi)$. Hence, ϕ as well as its equivalent $\text{QE}(\phi)$ are either equivalent to *true* or to *false*. Yet, $\text{QE}(\phi)$ is quantifier-free, so which one it is can be found out simply by evaluating the (variable-free) concrete arithmetic in $\text{QE}(\phi)$ as in the above examples.

Example 3. Quantifier elimination uses the special structure of real arithmetic to express quantified arithmetic formulas equivalently without quantifiers and without using more free variables. For instance, QE yields the following equivalence:

$$\text{QE}(\exists x (2x^2 + c \leq 5)) \equiv c \leq 5.$$

In particular, the formula $\exists x (2x^2 + c \leq 5)$ is not valid, but only true if $c \leq 5$ holds, as has been so aptly described by the outcome of the above quantifier elimination result.

Example 4. Quantifier elimination can be used to find out whether a first-order formula of real arithmetic is valid. Take $\exists x (2x^2 + c \leq 5)$, for example. A formula is valid iff its universal closure is, i.e. the formula obtained by universally quantifying all free variables. After all, valid means that a formula is true for all interpretations. Hence, consider the universal closure $\forall c \exists x (2x^2 + c \leq 5)$, which is a closed formula, because it has no free variables. Quantifier elimination could, for example, lead to

$$\text{QE}(\forall c \exists x (2x^2 + c \leq 5)) \equiv \text{QE}(\forall c \text{QE}(\exists x (2x^2 + c \leq 5))) \equiv \text{QE}(\forall c (c \leq 5)) \equiv -100 \leq 5 \wedge 5 \leq 5 \wedge 100 \leq 5$$

The resulting formula still has no free variables but is now quantifier-free, so it can simply be evaluated arithmetically. Since the conjunct $100 \leq 5$ evaluates to *false*, the universal closure $\forall c \exists x (2x^2 + c \leq 5)$ is equivalent to *false* and, hence, the original formula $\exists x (2x^2 + c \leq 5)$ is not valid (although still satisfiable for $c = 1$).

Geometrically, quantifier elimination corresponds to projection, see Fig. 1.

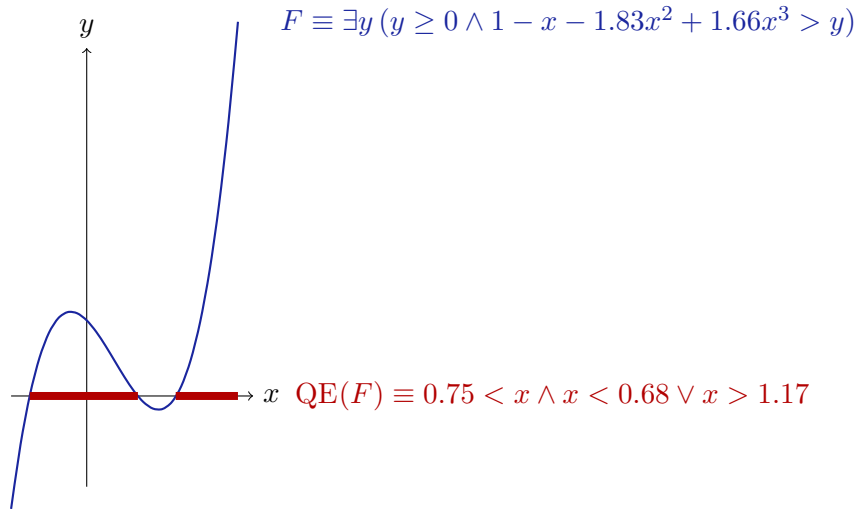


Figure 1: The geometric counterpart of quantifier elimination for $\exists y$ is projection onto the x axis

Note that, when using QE, we usually assume it would already evaluate ground arithmetic, so that the only two possible outcomes of applying QE to a closed formula are *true* and *false*.

Alfred Tarski's result that quantifier elimination over the reals is possible and that real arithmetic is decidable was groundbreaking. The only issue is that the complexity of Tarski's decision procedure is non-elementary, i.e. cannot be bounded by any tower of exponentials $2^{2^{\dots^n}}$, which made it quite impractical. Still, it was a seminal breakthrough because it showed reals to be decidable at all. It was not until another seminal result in 1949 by Julia Robinson, who proved the rationals to be undecidable [Rob49]. It took many further advances [Sei54, Coh69, KK71, Hör83, Eng93] and a major breakthrough by George Collins in 1975 [Col75] until more practical procedures had been found [Col75, CH91, Wei97]. The virtual substitution technique shown in this lecture has been implemented in Redlog [DS97], which has an interface for KeYmaera [PQ08]. There is also a recent approach of combining ideas from SMT solving with nonlinear real arithmetic [JdM12] implemented in the SMT solver Z3, which has an interface for KeYmaera.

4. Homomorphic Normalization for Quantifier Elimination

The first insight for defining quantifier elimination is to understand that the quantifier elimination operation commutes with almost all logical connectives, so that QE only needs to be defined for existential quantifiers. Consequently, as soon as we understand how to eliminate existential quantifiers, universal quantifiers can be eliminated as well just by double negation.

$$\text{QE}(A \wedge B) \equiv \text{QE}(A) \wedge \text{QE}(B)$$

$$\text{QE}(A \vee B) \equiv \text{QE}(A) \vee \text{QE}(B)$$

$$\text{QE}(\neg A) \equiv \neg \text{QE}(A)$$

$$\text{QE}(\forall x A) \equiv \text{QE}(\neg \exists x \neg A)$$

These transformations isolate existential quantifiers for quantifier elimination. In particular, it is sufficient if quantifier elimination focuses on existentially quantified variables. When using the QE operation inside out, i.e. when using it repeatedly to eliminate the inner-most quantifier to a quantifier-free equivalent and then again eliminating the inner-most quantifier, the quantifier elimination is solved if only we manage to solve it for $\exists x A$ with a quantifier-free formula A . If A is not quantifier-free, its quantifiers can be eliminated from inside out:

$$\text{QE}(\exists x A) \equiv \text{QE}(\exists x \text{QE}(A)) \quad \text{if } A \text{ not quantifier-free}$$

It is possible, although not necessary and not even necessarily helpful, to simplify the form of A as well. The following transformations transform the kernel of a quantifier into negation normal form using deMorgan's equivalences.

$$\begin{aligned} \text{QE}(\exists x (A \vee B)) &\equiv \text{QE}(\exists x A) \vee \text{QE}(\exists x B) \\ \text{QE}(\exists x \neg(A \wedge B)) &\equiv \text{QE}(\exists x (\neg A \vee \neg B)) \\ \text{QE}(\exists x \neg(A \vee B)) &\equiv \text{QE}(\exists x (\neg A \wedge \neg B)) \\ \text{QE}(\exists x \neg\neg A) &\equiv \text{QE}(\exists x A) \end{aligned}$$

Distributivity can be used to simplify the form of the quantifier-free *kernel* A to disjunctive normal form and split existential quantifiers over disjuncts:

$$\begin{aligned} \text{QE}(\exists x (A \wedge (B \vee C))) &\equiv \text{QE}(\exists x ((A \wedge B) \vee (A \wedge C))) \\ \text{QE}(\exists x ((A \vee B) \wedge C)) &\equiv \text{QE}(\exists x ((A \wedge C) \vee (B \wedge C))) \\ \text{QE}(\exists x (A \vee B)) &\equiv \text{QE}((\exists x A) \vee (\exists x B)) \end{aligned}$$

The only remaining case to address is the case $\text{QE}(\exists x (A \wedge B))$ where $A \wedge B$ is a purely conjunctive formula (yet it can actually have any number of conjuncts, not just two). Using the following normalizing equivalences,

$$\begin{aligned} p = q &\equiv p - q = 0 \\ p \geq q &\equiv p - q \geq 0 \\ p > q &\equiv p - q > 0 \\ p \neq q &\equiv p - q \neq 0 \\ p \leq q &\equiv q - p \geq 0 \\ p < q &\equiv q - p > 0 \\ \neg(p \geq q) &\equiv p < q \\ \neg(p > q) &\equiv p \leq q \\ \neg(p = q) &\equiv p \neq q \\ \neg(p \neq q) &\equiv p = q \end{aligned}$$

it is further possible to normalize all atomic formulas equivalently to one of the forms $p = 0, p > 0, p \geq 0, p \neq 0$. Since $p \neq 0 \equiv p > 0 \vee p < 0$, disequations \neq are unnecessary *in theory* as well (although they are quite useful in practice).

5. Substitution Base

Virtual substitution is a quantifier elimination technique that is based on substituting extended terms into formulas virtually, i.e. without the extended terms² actually occurring in the resulting constraints.

²Being an *extended real term* really means it is not a real term, but somehow closely related. We will see more concrete extended real terms and how to get rid of them again later.

Note 4. Virtual substitution essentially leads to an equivalence of the form

$$\exists x F \leftrightarrow \bigvee_{t \in T} A_t \wedge F_x^t \quad (1)$$

for a suitable finite set T of extended terms that depends on the formula F and that gets substituted into F virtually, i.e. in a way that results in standard real arithmetic terms, not extended terms.

Such an equivalence is how quantifier elimination can work. Certainly if the right-hand side of (1) is true, then t is a witness for $\exists x F$. The key to establishing an equivalence of the form (1) is to ensure that if F has a solution at all (in the sense of $\exists x F$ being true), then F must already hold for one of the cases in T . That is, T must cover all representative cases. There might be many more solutions, but if there is one at all, one of the possibilities in T must be a solution as well. If we were to choose all real numbers $T \stackrel{\text{def}}{=} \mathbb{R}$, then (1) would be trivially valid, but then the right-hand side is not a formula because it is uncountably infinitely long, which is even worse than the quantified form on the left-hand side. But if a finite set T is sufficient for the equivalence (1) and the extra formulas A_t are quantifier-free, then the right-hand side of (1) is structurally simpler than the left-hand side, even if it may be (sometimes significantly) less compact.

The various ways of virtually substituting various forms of extended reals e into logical formulas equivalently without having to mention the actual extended reals is the secret of virtual substitution. The first step is to see that it is enough to define substitutions only on atomic formulas of the form $p = 0, p < 0, p \leq 0$ (or, just as well, on $p = 0, p > 0, p \geq 0$). If σ denotes such an extended substitution of θ for x , then σ lifts to arbitrary first-order formulas homomorphically³ as follows

$$\begin{aligned} \sigma(A \wedge B) &\equiv \sigma A \wedge \sigma B \\ \sigma(A \vee B) &\equiv \sigma A \vee \sigma B \\ \sigma(\neg A) &\equiv \neg \sigma A \\ \sigma(\forall y A) &\equiv \forall y \sigma A && \text{if } x \neq y \text{ and } y \notin \theta \\ \sigma(\exists y A) &\equiv \exists y \sigma A && \text{if } x \neq y \text{ and } y \notin \theta \\ \sigma(p = q) &\equiv \sigma(p - q = 0) \\ \sigma(p < q) &\equiv \sigma(p - q < 0) \\ \sigma(p \leq q) &\equiv \sigma(p - q \leq 0) \\ \sigma(p > q) &\equiv \sigma(q - p < 0) \\ \sigma(p \geq q) &\equiv \sigma(q - p \leq 0) \\ \sigma(p \neq q) &\equiv \sigma(\neg(p - q = 0)) \end{aligned}$$

This lifting applies the substitution σ to all subformulas, with minor twists on quantifiers for admissibility and normalization of atomic formulas into the canonical forms

³With a caveat on admissibility for quantifiers to avoid capture of variables.

$p = 0, p < 0, p \leq 0$ for which σ has been assumed to already have been defined.

From now on, all that remains to be done for defining a substitution or virtual substitution is to define it on atomic formulas of the remaining forms $p = 0, p < 0, p \leq 0$ and the above construction will take care of substituting in any first-order formulas. Of course, the above construction is only helpful for normalizing atomic formulas that are not already of one of those forms, so the term q above can be assumed not to be the term 0.

6. Term Substitutions

Consider a formula of the form

$$\exists x (bx + c = 0 \wedge F) \quad (x \notin b, c) \quad (2)$$

where x does not occur in the terms b, c . Let's consider how a first mathematical solution to this formula might look like. The only solution that the conjunct $bx + c = 0$ has is $x = -c/b$. Hence, the left conjunct in (2) only holds for $x = -c/b$, so formula (2) can only be true if F also holds for that single solution $-c/b$ in place of x . That is, formula (2) holds only if $F_x^{-c/b}$ does. Hence, (2) is equivalent to the formula $F_x^{-c/b}$, which is quantifier-free.

So, how can we eliminate the quantifier in (2) equivalently?

Before you read on, see if you can find the answer for yourself.

Most certainly, $F_x^{-c/b}$ is quantifier-free. But it is not exactly always equivalent to (2) and, thus, does not necessarily qualify as its quantifier eliminate form. Oh no! What we wrote down is a good intuitive start, but does not make any sense at all if $b = 0$, for then $-c/b$ would have been a rather ill-devised division by zero. Performing such divisions by zero sounds like a fairly shaky start for an equivalence transformation such as quantifier elimination. And certainly sounds like a shaky start for anything that is supposed to ultimately turn into a proof.

Let's start over. The first conjunct in (2) has the only solution $x = -c/b$ if $b \neq 0$. In that case, indeed, (2) is equivalent to $F_x^{-c/b}$, because the only way for (2) to be true then is exactly when the second conjunct F holds for the solution of the first conjunct, i.e. when $F_x^{-c/b}$ holds. But there is, in general, no way of knowing whether evaluation could yield $b \neq 0$ or not, because b might be a complicated polynomial term that is only zero under some interpretations, not under all. Certainly if b is the zero polynomial, we know for sure. Or if b is a polynomial that is never zero, such as a sum of squares plus a positive constant. In general, if $b = 0$, then, the first conjunct in (2) has all numbers for x as solutions if $c = 0$ and, otherwise, has no solution at all if $c \neq 0$. In the latter case, $b = 0, c \neq 0$, (2) is false, because its first conjunct is already false. In the former case, $b = c = 0$, however, the first conjunct $bx + c = 0$ is trivial and does not impose any constraints on x , nor does it help for finding out a quantifier-free equivalent of (2). In that case $b = c = 0$, the trivial constraint will be dropped and the remaining formula will be considered recursively instead.

Note 5. In the non-degenerate case $b \neq 0$ with $x \notin b, c$, (2) can be rephrased into a quantifier-free equivalent over \mathbb{R} as follows:

$$b \neq 0 \rightarrow (\exists x (bx + c = 0 \wedge F) \leftrightarrow b \neq 0 \wedge F_x^{-c/b}) \quad (3)$$

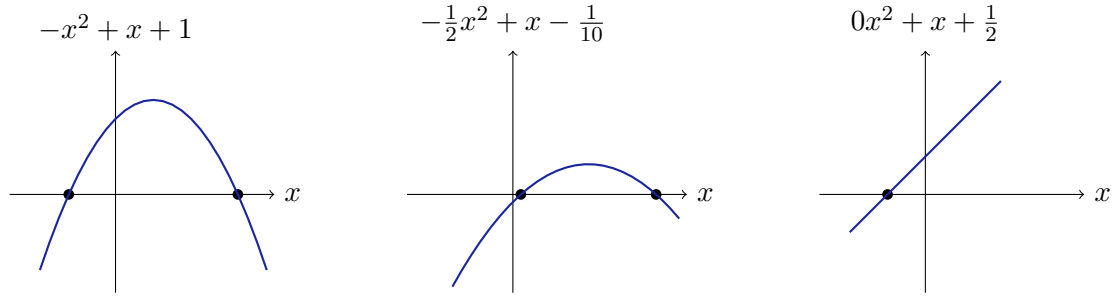
All it takes is, thus, the ability to substitute the term $-c/b$ for x in the formula F . The division $-c/b$ that will occur in $F_x^{-c/b}$ for ordinary term substitutions can cause technical annoyances but at least it is well-defined, because $b \neq 0$ holds in that context. Instead of pursuing the looming question how exactly this substitution in $F_x^{-c/b}$ works, we make the question more general by moving the quadratic case already.

7. Square Root $\sqrt{\cdot}$ Substitutions for Quadratics

Consider a formula of the form

$$\exists x (ax^2 + bx + c = 0 \wedge F) \quad (x \notin a, b, c) \quad (4)$$

where x does not occur in the terms a, b, c . The generic solution of its first conjunct is $x = (-b \pm \sqrt{b^2 - 4ac})/(2a)$, but that, of course, again depends on whether a could evaluate to zero, in which case linear solutions may be possible and the division by $2a$ is most certainly not well-defined; see Fig. 2. Whether a could be zero may again

Figure 2: Roots of quadratic functions p

sometimes be hard to say when a is a polynomial term that has roots, but does not always evaluate to 0 either (which only the zero polynomial would). So let's be more careful this time to find an equivalent formulation right away for all possible cases of a, b, c . The cases to consider are where the first conjunct is either a constant equation (in which case the equation imposes no interesting constraint on x) or a linear equation (in which case $x = -c/b$ is the solution Sect. 6) or a proper quadratic equation with $a \neq 0$ (in which case $x = (-b \pm \sqrt{b^2 - 4ac})/(2a)$ is the solution). The trivial equation $0 = 0$ when $a = b = c = 0$ is again useless, so another part of F would have to be considered in that case, and the equation $c = 0$ for $a = b = 0, c \neq 0$ is again *false*.

When $ax^2 + bx = 0$ is either a proper linear or a proper quadratic equation, its respective solutions single out the only points that can solve (4), so the only points in which it remains to be checked whether the second conjunct F also holds.

Theorem 5 (Virtual substitution of quadratic equations). *For a quantifier-free formula F with $x \notin a, b, c$, the following equivalence is valid over \mathbb{R} :*

$$\begin{aligned}
 a \neq 0 \vee b \neq 0 \vee c \neq 0 \rightarrow \\
 & \left(\exists x (ax^2 + bx + c = 0 \wedge F) \leftrightarrow \right. \\
 & \quad a = 0 \wedge b \neq 0 \wedge F_x^{-c/b} \\
 & \quad \left. \vee a \neq 0 \wedge b^2 - 4ac \geq 0 \wedge \left(F_x^{(-b + \sqrt{b^2 - 4ac})/(2a)} \vee F_x^{(-b - \sqrt{b^2 - 4ac})/(2a)} \right) \right) \\
 & \hspace{15em} (5)
 \end{aligned}$$

Hold on, we fortunately noticed just in time for writing down the formula (5) that $(-b + \sqrt{b^2 - 4ac})/(2a)$ only ever makes actual sense in the reals if $b^2 - 4ac \geq 0$, because the square root is otherwise imaginary, which is hard to find in $\text{FOL}_{\mathbb{R}}$.

The resulting formula on the right-hand side of the biimplication is quantifier-free and, thus, sounds like it could be chosen for $\text{QE}(\exists x (ax^2 + bx + c = 0 \wedge F))$ as long as it is not the case that $a = b = c = 0$.

Note 7. The important thing to notice, though, is that $(-b \pm \sqrt{b^2 - 4ac})/(2a)$ is not a polynomial term, nor even a rational term, because it involves a square root $\sqrt{\cdot}$. Hence, (5) is not generally a formula of first-order real arithmetic! Unless we do something about its square roots and divisions.

Recall from [Lecture 2 on Differential Equations & Domains](#) that the terms of $\text{FOL}_{\mathbb{R}}$ are polynomials with rational coefficients in \mathbb{Q} . So $4x^2 + \frac{1}{7}x - 1.41$ is a polynomial term of $\text{FOL}_{\mathbb{R}}$. But $4x^2 + \frac{1}{y}x - 1.41$ is not, because of the division by variable y , which should make us panic about y possibly being zero in any case. And $4x^2 + \frac{1}{7}x - \sqrt{2}$ is not either, because of the square root $\sqrt{2}$.

Note 8 (Semantic domains versus syntactic expressions). While the domains that the quantifiers \forall and \exists of first-order logic $\text{FOL}_{\mathbb{R}}$ of real arithmetic quantify over includes reals like $\sqrt{2}$, the terms and logical formulas themselves are syntactically restricted to be built from polynomials with rational coefficients. Square roots (and all higher roots) are already part of the semantic domain \mathbb{R} , but not allowed in the syntax of $\text{FOL}_{\mathbb{R}}$.

Of course, it is still easy to write down a formula such as $\exists x x^2 = 2$ which indirectly makes sure that x will have to assume the value $\sqrt{2}$, but that formula mentions a quantifier again.

Square roots are really not part of real arithmetic. But they can be defined, still, by appropriate quadratures. For example, the positive root $x = \sqrt{y}$ can be defined as $x^2 = y \wedge y \geq 0$. Let's find out how square roots such as $(-b \pm \sqrt{b^2 - 4ac})/(2a)$ can be substituted into first-order formulas systematically without the need for involving any square roots in the resulting formula.

A square root expression is an expression of the form

$$(a + b\sqrt{c})/d$$

with polynomials $a, b, c, d \in \mathbb{Q}[x_1, \dots, x_n]$ of rational coefficients in the variables x_1, \dots, x_n and, for well-definedness, $d \neq 0 \wedge c \geq 0$. Square root expressions with the same \sqrt{c} can be added and multiplied symbolically by considering them as algebraic objects.⁴

$$\begin{aligned} (a + b\sqrt{c})/d + (a' + b'\sqrt{c})/d' &= ((ad' + da') + (bd' + db')\sqrt{c})/(dd') \\ ((a + b\sqrt{c})/d) \cdot ((a' + b'\sqrt{c})/d') &= ((aa' + bb'c) + (ab' + ba')\sqrt{c})/(dd') \end{aligned} \quad (6)$$

Another way of saying that is that square root expressions with the same \sqrt{c} provide an addition and a multiplication operation that leads to square root expressions. Substituting $(a + b\sqrt{c})/d$ for a variable x in a polynomial term p , thus, leads to a square root

⁴Despite the poor notation, please don't mistake the primes for derivatives here. The name a' is not the derivative of a here but just meant as a name for a polynomial term that happens to go by the misleading name a' .

expression $p_x^{(a+b\sqrt{c})/d} = (\tilde{a} + \tilde{b}\sqrt{c})/\tilde{d}$ with the same \sqrt{c} , because the arithmetic resulting from evaluating the polynomial only requires addition and multiplication using (6).⁵

Note 9. Subsequent symbolic addition and multiplication makes it possible to substitute a square root expression in for a variable in a polynomial to form. Yet, the result $p_x^{(a+b\sqrt{c})/d}$ is still a square root expression, which still cannot be written down directly in first-order real arithmetic. Yet, as soon as a square root expression appears in an atomic formula of first-order real arithmetic, that square root can be rephrased equivalently to disappear.

The substitution of a square root expression $(a' + b'\sqrt{c})/d'$ into a polynomial p for x to form $p_x^{(a+b\sqrt{c})/d}$ by polynomial evaluation leads to a square root expression, say the square root expression $p_x^{(a'+b'\sqrt{c})/d'} = (a + b\sqrt{c})/d$.

The next step is to handle the comparison of the resulting square root expression to 0 in atomic formulas $p \sim 0$ for some $\sim \in \{=, \leq, <\}$. That works by characterizing it using the square root expression $p_x^{(a'+b'\sqrt{c})/d'}$:

$$(p \sim 0)_{\tilde{x}}^{(a'+b'\sqrt{c})/d'} \equiv (p_x^{(a'+b'\sqrt{c})/d'} \sim 0)$$

Suppose the square root expression $p_x^{(a'+b'\sqrt{c})/d'}$ from the polynomial evaluation is $(a + b\sqrt{c})/d$. All that remains to be done is to rewrite $(a + b\sqrt{c})/d \sim 0$ equivalently in $\text{FOL}_{\mathbb{R}}$.

Assume $d \neq 0 \wedge c \geq 0$ for well-definedness. For square-root-free expressions ($b = 0$) with just divisions, i.e. those of the form $(a + 0\sqrt{c})/d$, the following equivalences hold:

$$\begin{aligned} (a + 0\sqrt{c})/d = 0 &\equiv a = 0 \\ (a + 0\sqrt{c})/d \leq 0 &\equiv ad \leq 0 \\ (a + 0\sqrt{c})/d < 0 &\equiv ad < 0 \end{aligned}$$

Assume $d \neq 0 \wedge c \geq 0$ for well-definedness. For square root expressions $(a + b\sqrt{c})/d$ with arbitrary b , the following equivalences hold:

$$\begin{aligned} (a + b\sqrt{c})/d = 0 &\equiv ab \leq 0 \wedge a^2 - b^2c = 0 \\ (a + b\sqrt{c})/d \leq 0 &\equiv ad \leq 0 \wedge a^2 - b^2c \geq 0 \vee bd \leq 0 \wedge a^2 - b^2c \leq 0 \\ (a + b\sqrt{c})/d < 0 &\equiv ad < 0 \wedge a^2 - b^2c > 0 \vee bd \leq 0 \wedge (ad < 0 \vee a^2 - b^2c < 0) \end{aligned}$$

The first line characterizes that $= 0$ holds iff a, b have different signs (possibly 0) and their squares cancel, because $a^2 = b^2c$. The second line characterizes that ≤ 0 holds iff $a^2 \geq b^2c$ so that a will dominate, which has a different sign than d , or if $a^2 \leq b^2c$ so that $b\sqrt{c}$ will dominate, which has a different sign than d (possibly 0). The squares $a^2 - b^2c = a^2 - b^2\sqrt{c}^2$ is the square of the absolute value of the involved terms, which uniquely identifies the truth-values along with the accompanying sign conditions. The

⁵In practice, the polynomial addition and multiplication operations for a polynomial η are performed by Horner's scheme for dense polynomials η and by repeated squaring for sparse polynomials η .

third line characterizes that < 0 holds iff a strictly dominates, because $a^2 > b^2c$ and the dominant a, d have different signs or if b, d have different signs and either a, d have different signs as well (so a, b have the same sign or 0 and different than d) or b strictly dominates because $a^2 < b^2c$. The last case involves a little extra care for the required sign conditions to avoid the $= 0$ case.

This defines the substitution of a square root $(a + b\sqrt{c})/d$ for x into atomic formulas and can be lifted to all first-order logic formulas as explained in Sect. 5. The important thing to observe is that the result of this substitution does not introduce square root expressions nor divisions even though the square root expression $(a + b\sqrt{c})/d$ had the square root \sqrt{c} and the division $/d$. Substitution of a square root $(a + b\sqrt{c})/d$ for x into a (quantifier-free) first-order formula F then works as usual by substitution in all atomic formulas (as defined in Sect. 5). The result of such a *virtual* substitution is denoted by $F_{\hat{x}}^{(a+b\sqrt{c})/d}$.

It is crucial to note that the *virtual substitution* of square root expression $(a + b\sqrt{c})/d$ for x in F giving $F_{\hat{x}}^{(a+b\sqrt{c})/d}$ is semantically equivalent to the result $F_x^{(a+b\sqrt{c})/d}$ of the literal substitution replacing x with $(a + b\sqrt{c})/d$, but operationally quite different, because the virtual substitution never introduces square roots or divisions. Because of their semantical equivalence, we use the same notation by abuse of notation.

Lemma 6 (Virtual substitution lemma for square roots). *The result $F_{\hat{x}}^{(a+b\sqrt{c})/d}$ of the virtual substitution is semantically equivalent to the the result $F_x^{(a+b\sqrt{c})/d}$ of the literal substitution, but better behaved, because it stays within $\text{FOL}_{\mathbb{R}}$ proper. Essentially, the following equivalence of virtual substitution and literal substitution for square root expressions is valid:*

$$F_x^{(a+b\sqrt{c})/d} \leftrightarrow F_{\hat{x}}^{(a+b\sqrt{c})/d}$$

Keep in mind, though, that the result $F_{\hat{x}}^{(a+b\sqrt{c})/d}$ of virtual substitution is a proper formula of $\text{FOL}_{\mathbb{R}}$, while the literal substitution $F_x^{(a+b\sqrt{c})/d}$ could actually only even be considered a formula in an extended logic that allows for a syntactic representation of divisions and square root expressions within a context in which they are meaningful (no divisions by zero, no imaginary roots).

Using Lemma 6, Theorem 5 continues to hold when using the so-defined square root virtual substitutions $F_{\hat{x}}^{(-b \pm \sqrt{b^2 - 4ac})/(2a)}$ that turn (5) into a valid formula of first-order real arithmetic, without scary square root expressions. In particular, since the fraction $-c/b$ also is a (somewhat impoverished) square root expression $(-c + 0\sqrt{0})/b$, the $\text{FOL}_{\mathbb{R}}$ formula $F_{\hat{x}}^{-c/b}$ in (5) can be formed and rephrased equivalently using the square root virtual substitution as well. Hence, the quantifier-free right-hand side of (5) neither introduces square roots nor divisions but happily remains a proper formula in $\text{FOL}_{\mathbb{R}}$.

With this virtual substitution, the right-hand side of the biimplication (5) can be chosen as $\text{QE}(\exists x (ax^2 + bx + c = 0 \wedge F))$ if it is not the case that $a = b = c = 0$.

When using square root substitutions, divisions could, thus, also have been avoided

in the quantifier elimination (3) for the linear case. Thus, the right-hand side of (3) can be chosen as $\text{QE}(\exists x (bx + c = 0 \wedge F))$ if it is not the case that $b = c = 0$.

8. Optimizations

Before going any further, it is helpful to notice that virtual substitutions admit a number of useful optimizations that make it more practical. When substituting a square root expression $(a + b\sqrt{c})/d$ for a variable x in a polynomial p , the resulting square root expression $p_{\tilde{x}}^{(a+b\sqrt{c})/d} = (\tilde{a} + \tilde{b}\sqrt{c})/\tilde{d}$ will end up occurring with a higher power of the form $\tilde{d} = d^k$ where k is the degree of p in variable x . This is easy to see just by inspecting the definitions of addition and multiplication from (6). Such larger powers of d can be avoided using the equivalences $(pq^3 \sim 0) \equiv (pq \sim 0)$ and, if $q \neq 0$, using also $(pq^2 \sim 0) \equiv (p \sim 0)$ for arithmetic relations $\sim \in \{=, >, \geq, \neq, <, \leq\}$. Since $d \neq 0$ needs to be assumed for well-definedness of a square root expression $(a + b\sqrt{c})/d$, the degree of d in the result $F_{\tilde{x}}^{(a+b\sqrt{c})/d}$ of the virtual substitution can, thus, be lowered to either 0 or 1 depending on whether it ultimately occurs as an even or as an odd power (Exercise 7). If d occurs as an odd power, its occurrence can be lowered to degree 1. If d occurs as an even power, its occurrence can be reduced to degree 0, which makes it disappear entirely.

The significance of lowering degrees does not just come from the conceptual and computational impact that large degrees have on the problem of quantifier elimination, but, for the case of virtual substitution, also from the fact that virtual substitution only works for certain bounded but common degrees.

Example 7. Using this principle to check under which circumstance the quadratic equality from (4) evaluates to *true* requires a nontrivial number of algebraic and logical computations to handle the virtual substitution of the respective roots of $ax^2 + bx + c = 0$ into F .

Just out of curiosity: What would happen if we tried to apply the same virtual substitution coming from this equation to $ax^2 + bx + c = 0$ itself instead of to F ? Imagine, for example, that $ax^2 + bx + c = 0$ shows up a second time in F . Let's only consider the case of quadratic solutions, i.e. where $a \neq 0$. And let's only consider the root $(-b + \sqrt{b^2 - 4ac})/(2a)$. The other cases are left as an exercise. First virtually substitute $(-b + \sqrt{b^2 - 4ac})/(2a)$ into the polynomial $ax^2 + bx + c$ leading to symbolic square root

expression arithmetic:

$$\begin{aligned}
& (ax^2 + bx + c)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)} \\
&= a((-b + \sqrt{b^2 - 4ac})/(2a))^2 + b((-b + \sqrt{b^2 - 4ac})/(2a)) + c \\
&= a((b^2 + b^2 - 4ac + (-b - b)\sqrt{b^2 - 4ac})/(4a^2)) + (-b^2 + b\sqrt{b^2 - 4ac})/(2a) + c \\
&= (ab^2 + ab^2 - 4a^2c + (-ab - ab)\sqrt{b^2 - 4ac})/(4a^2) + (-b^2 + 2ac + b\sqrt{b^2 - 4ac})/(2a) \\
&= ((ab^2 + ab^2 - 4a^2c)2a + (-b^2 + 2ac)4a^2 + ((-ab - ab)2a + b4a^2)\sqrt{b^2 - 4ac})/(4a^2) \\
&= (\cancel{2a^2b^2} + \cancel{2a^2b^2} - \cancel{8a^3c} + \cancel{-4a^2b^2} + \cancel{8a^3c} + (-\cancel{2a^2b} - \cancel{2a^2b} + \cancel{4a^2b})\sqrt{b^2 - 4ac})/(4a^2) \\
&= (0 + 0\sqrt{0})/1 = 0
\end{aligned}$$

So $(ax^2 + bx + c)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)}$ is the zero square root expression? That is actually exactly as expected by construction, because $(-b \pm \sqrt{b^2 - 4ac})/(2a)$ is supposed to be the root of $ax^2 + bx + c$ in the case where $a \neq 0 \wedge b^2 - 4ac \geq 0$. In particular, if $ax^2 + bx + c$ occurs again in F as either an equation or inequality, its virtual substitute in the various cases just ends up being:

$$\begin{aligned}
(ax^2 + bx + c = 0)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)} &\equiv ((0 + 0\sqrt{0})/1 = 0) \equiv (0 \cdot 1 = 0) \equiv \text{true} \\
(ax^2 + bx + c \leq 0)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)} &\equiv ((0 + 0\sqrt{0})/1 \leq 0) \equiv (0 \cdot 1 \leq 0) \equiv \text{true} \\
(ax^2 + bx + c < 0)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)} &\equiv ((0 + 0\sqrt{0})/1 < 0) \equiv (0 \cdot 1 < 0) \equiv \text{false} \\
(ax^2 + bx + c \neq 0)_{\hat{x}}^{(-b + \sqrt{b^2 - 4ac})/(2a)} &\equiv ((0 + 0\sqrt{0})/1 \neq 0) \equiv (0 \cdot 1 \neq 0) \equiv \text{false}
\end{aligned}$$

And that makes sense as well. After all, the roots of $ax^2 + bx + c = 0$ satisfy the weak inequality $ax^2 + bx + c \leq 0$ but not the strict inequality $ax^2 + bx + c < 0$. In particular, Theorem 5 could substitute the roots of $ax^2 + bx + c = 0$ also into the full formula $ax^2 + bx + c = 0 \wedge F$ under the quantifier, but the formula resulting from the left conjunct $ax^2 + bx + c = 0$ will always simplify to *true* so that only the virtual substitution into F will remain, where actual logic with real arithmetic happens.

The above computations are all that is needed for Theorem 5 to show the following quantifier elimination equivalences:

$$\begin{aligned}
a \neq 0 \rightarrow (\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c = 0) &\leftrightarrow b^2 - 4ac \geq 0 \wedge \text{true}) \\
a \neq 0 \rightarrow (\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c \leq 0) &\leftrightarrow b^2 - 4ac \geq 0 \wedge \text{true})
\end{aligned}$$

With analog computations for the case $(-b - \sqrt{b^2 - 4ac})/(2a)$, this also justifies:

$$\begin{aligned}
a \neq 0 \rightarrow (\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c < 0) &\leftrightarrow b^2 - 4ac \geq 0 \wedge \text{false}) \\
a \neq 0 \rightarrow (\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c \neq 0) &\leftrightarrow b^2 - 4ac \geq 0 \wedge \text{false})
\end{aligned}$$

Consequently, in a context where $a \neq 0$ is known, for example because it is a term such as 5 or $y^2 + 1$, Theorem 5 and simplification yields the following quantifier elimination results:

$$\begin{aligned} \text{QE}(\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c = 0)) &\equiv b^2 - 4ac \geq 0 \\ \text{QE}(\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c \leq 0)) &\equiv b^2 - 4ac \geq 0 \\ \text{QE}(\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c < 0)) &\equiv \text{false} \\ \text{QE}(\exists x (ax^2 + bx + c = 0 \wedge ax^2 + bx + c \neq 0)) &\equiv \text{false} \end{aligned}$$

In a context where $a \neq 0$ is not known, more cases become possible and the disjunctive structure in Theorem 5 remains, leading to a case distinction on whether $a = 0$ or $a \neq 0$.

Example 8 (Nonnegative roots of quadratic polynomials). Consider the formula

$$\exists x (ax^2 + bx + c = 0 \wedge x \geq 0) \quad (7)$$

for the purpose of eliminating quantifiers using Theorem 5. For simplicity, again assume $a \neq 0$ is known, e.g., because $a = 5$. Since $a \neq 0$, Theorem 5 will only consider the two square root expressions $(-b + \sqrt{b^2 - 4ac})/(2a)$ and $(-b - \sqrt{b^2 - 4ac})/(2a)$ and no linear roots. The first thing that happens during the virtual substitution of those roots into the remaining formula $F \equiv (x \geq 0)$ is that the construction in Sect. 5 will flip $x \geq 0$ around to a base case $-x \leq 0$. On that base case, the substitution of the square root expression $(-b + \sqrt{b^2 - 4ac})/(2a)$ into the polynomial $-x$ leads to the following square root computations following (6):

$$-(-b + \sqrt{b^2 - 4ac})/(2a) = ((-1 + 0\sqrt{b^2 - 4ac})/1) \cdot ((-b + \sqrt{b^2 - 4ac})/(2a)) = (b - \sqrt{b^2 - 4ac})/(2a)$$

Observe how the unary minus operator expands to multiplication by -1, whose representation as a square root expression with square root $\sqrt{b^2 - 4ac}$ is $(-1 + 0\sqrt{b^2 - 4ac})/1$. The virtual square root substitution of this square root expression, thus, yields

$$\begin{aligned} &(-x \leq 0)_{\hat{x}}^{(b - \sqrt{b^2 - 4ac})/(2a)} \\ &\equiv b2a \leq 0 \wedge \cancel{b^2} - (-1)^2(\cancel{b^2} - 4ac) \geq 0 \vee -1 \cdot 2a \leq 0 \wedge \cancel{b^2} - (-1)^2(\cancel{b^2} - 4ac) \leq 0 \\ &\equiv 2ba \leq 0 \wedge 4ac \geq 0 \vee -2a \leq 0 \wedge 4ac \leq 0 \end{aligned}$$

For the second square root expression $(-b - \sqrt{b^2 - 4ac})/(2a)$, the corresponding polynomial evaluation leads to

$$-(-b - \sqrt{b^2 - 4ac})/(2a) = ((-1 + 0\sqrt{b^2 - 4ac})/1) \cdot ((-b - \sqrt{b^2 - 4ac})/(2a)) = (b + \sqrt{b^2 - 4ac})/(2a)$$

The virtual square root substitution of this square root expression, thus, yields

$$\begin{aligned} &(-x \leq 0)_{\hat{x}}^{(b + \sqrt{b^2 - 4ac})/(2a)} \\ &\equiv b2a \leq 0 \wedge \cancel{b^2} - 1^2(\cancel{b^2} - 4ac) \geq 0 \vee 1 \cdot 2a \leq 0 \wedge \cancel{b^2} - 1^2(\cancel{b^2} - 4ac) \leq 0 \\ &\equiv 2ba \leq 0 \wedge 4ac \geq 0 \vee 2a \leq 0 \wedge 4ac \leq 0 \end{aligned}$$

Consequently, since $a \neq 0$, Theorem 5 implies the quantifier elimination equivalence:

$$a \neq 0 \rightarrow (\exists x (ax^2 + bx + c = 0 \wedge x \geq 0)) \\ \leftrightarrow b^2 - 4ac \geq 0 \wedge (2ba \leq 0 \wedge 4ac \geq 0 \vee -2a \leq 0 \wedge 4ac \leq 0 \vee 2ba \leq 0 \wedge 4ac \geq 0 \vee 2a \leq 0 \wedge 4ac \leq 0))$$

Consequently, in a context where $a \neq 0$ is known, 5 yields the following quantifier elimination results:

$$\text{QE}(\exists x (ax^2 + bx + c = 0 \wedge x \geq 0)) \\ \equiv b^2 - 4ac \geq 0 \wedge (2ba \leq 0 \wedge 4ac \geq 0 \vee -2a \leq 0 \wedge 4ac \leq 0 \vee \cancel{2ba \leq 0 \wedge 4ac \geq 0} \vee 2a \leq 0 \wedge 4ac \leq 0) \\ \equiv b^2 - 4ac \geq 0 \wedge (ba \leq 0 \wedge ac \geq 0 \vee a \geq 0 \wedge ac \leq 0 \vee a \leq 0 \wedge ac \leq 0)$$

The sign conditions that this formula expresses make sense when you consider that the original quantified formula (7) expresses that the quadratic equation has a nonnegative root.

9. Summary

This lecture showed part of the miracle of quantifier elimination and quantifier elimination is possible in first-order real arithmetic. Today's technique works for formulas that normalize into an appropriate form as long as the technique can latch on to a linear or quadratic equation for all quantified variables. Note that there can be higher-degree or inequality occurrences of the variables as well within the formula F of Theorem 5, but there has to be at least one linear or quadratic equation. Commuting the formula so that it has the required form is easily done if such an equation is anywhere at all. What is to be done if there is no quadratic equation but only other quadratic inequalities is the topic of the next lecture.

It is also foreseeable that the virtual substitution approach will ultimately run into difficulties for pure high-degree polynomials, because those generally have no radicals to solve the equations. That is where other more algebraic quantifier elimination techniques come into play that are beyond the scope of this lecture.

Virtual substitution of square root expressions uses simple symbolic computations:

$$(\alpha + \beta\sqrt{\gamma})/\delta + (\alpha' + \beta'\sqrt{\gamma})/\delta' = ((\alpha\delta' + \delta\alpha') + (\beta\delta' + \delta\beta')\sqrt{\gamma})/(\delta\delta') \\ ((\alpha + \beta\sqrt{\gamma})/\delta) \cdot ((\alpha' + \beta'\sqrt{\gamma})/\delta') = ((\alpha\alpha' + \beta\beta'\gamma) + (\alpha\beta' + \beta\alpha')\sqrt{\gamma})/(\delta\delta')$$

The following expansions were the core of eliminating square root expressions by virtual substitutions. For square root expressions $(\alpha + \beta\sqrt{\gamma})/\delta$ with $\delta \neq 0 \wedge \gamma \geq 0$ for well-definedness, the following equivalences hold:

$$(\alpha + \beta\sqrt{\gamma})/\delta = 0 \equiv \alpha\beta \leq 0 \wedge \alpha^2 - \beta^2\gamma = 0 \\ (\alpha + \beta\sqrt{\gamma})/\delta \leq 0 \equiv \alpha\delta \leq 0 \wedge \alpha^2 - \beta^2\gamma \geq 0 \vee \beta\delta \leq 0 \wedge \alpha^2 - \beta^2\gamma \leq 0 \\ (\alpha + \beta\sqrt{\gamma})/\delta < 0 \equiv \alpha\delta < 0 \wedge \alpha^2 - \beta^2\gamma > 0 \vee \beta\delta \leq 0 \wedge (\alpha\delta < 0 \vee \alpha^2 - \beta^2\gamma < 0)$$

A. Real Algebraic Geometry

This course follows a logical view on cyber-physical systems. It is helpful to develop an intuition to what geometric objects the various logical concepts correspond. The part that is most interesting in this context is real algebraic geometry [BCR98] as it relates to real arithmetic [BPR06]. General algebraic geometry is also very elegant and beautiful, especially over algebraically closed fields [Har95, CLO92].

The geometric counterpart of polynomial equations are real affine algebraic varieties. Every set F of polynomials defines a geometric object, its variety, i.e. the set of points on which all those polynomials are zero.

Definition 9 (Real Affine Algebraic Variety). $V \subseteq \mathbb{R}^n$ is an *affine variety* iff, for some set $F \subseteq \mathbb{R}[X_1, \dots, X_n]$ of polynomials over \mathbb{R} :

$$V = V(F) := \{x \in \mathbb{R}^n : f(x) = 0 \text{ for all } f \in F\}$$

i.e., affine varieties are subsets of \mathbb{R}^n that are definable by a set of polynomial equations.

The converse construction is that of the vanishing ideal, which describes the set of all polynomials that are zero on a given set V .

Definition 10 (Vanishing Ideal). $I \subseteq \mathbb{R}[X_1, \dots, X_n]$ is the *vanishing ideal* of $V \subseteq \mathbb{R}^n$:

$$I(V) := \{f \in \mathbb{R}[X_1, \dots, X_n] : f(x) = 0 \text{ for all } x \in V\}$$

i.e., all polynomials that are zero on all of V .

Affine varieties and vanishing ideals are related by

$$\begin{aligned} S &\subseteq V(I(S)) && \text{for any set } S \subseteq \mathbb{R}^n \\ V &= V(I(V)) && \text{if } V \text{ an affine variety} \\ F \subseteq G &\Rightarrow V(F) \supseteq V(G) \end{aligned}$$

Affine varieties and vanishing ideals are intimately related by Hilbert's Nullstellensatz over algebraically closed fields such as \mathbb{C} and by Stengle's Nullstellensatz over real-closed fields such as \mathbb{R} .

The affine varieties corresponding to a number of interesting polynomials are illustrated in Fig. 3.

Exercises

Exercise 1. Example 7 showed that $ax^2 + bx + c = 0$ simplifies to *true* for the virtual substitution of the root $(-b + \sqrt{b^2 - 4ac})/(2a)$. Show that the same thing happens for the root $(-b - \sqrt{b^2 - 4ac})/(2a)$ and the root $(-c + 0\sqrt{0})/b$.

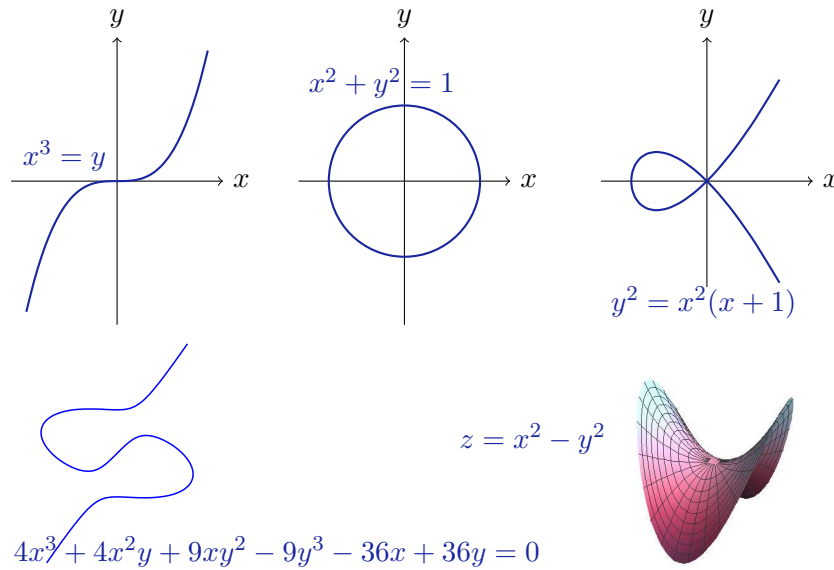


Figure 3: Polynomial equations describe (real) affine (algebraic) varieties

Exercise 2. Example 7 argued that the simplification of $ax^2 + bx + c = 0$ to *true* for the virtual substitution of the root $(-b + \sqrt{b^2 - 4ac})/(2a)$ is to be expected, because the real number to which $(-b + \sqrt{b^2 - 4ac})/(2a)$ evaluates is a root of $ax^2 + bx + c = 0$ in the case where $a \neq 0 \wedge b^2 - 4ac \geq 0$. Yet, what happens in the case where the extra assumption $a \neq 0 \wedge b^2 - 4ac \geq 0$ does not hold? What is the value of the virtual substitution in that case? Is that a problem? Discuss carefully!

Exercise 3. Use Theorem 5 to eliminate quantifiers in the following formula, assuming $a \neq 0$ is known:

$$\exists x (ax^2 + bx + c = 0 \wedge x < 1)$$

Exercise 4. Use Theorem 5 to eliminate quantifiers in the following formula, assuming $a \neq 0$ is known:

$$\exists x (ax^2 + bx + c = 0 \wedge x^3 + x \leq 0)$$

Exercise 5. How does Example 8 change when removing the assumption that $a \neq 0$?

Exercise 6. Would first-order logic of real arithmetic miss the presence of π ? That is, if we delete π from the domain and make all quantifiers range only over $\mathbb{R} \setminus \{\pi\}$, would there be any formula that notices by having a different truth-value? If we delete $\sqrt[3]{5}$ from the domain, would $\text{FOL}_{\mathbb{R}}$ notice?

Exercise 7. Consider the process of substituting a square root expression $(a + b\sqrt{c})/d$ for a variable x in a polynomial p . Let k be the degree of p in variable x , so that d occurs as d^k with power k in the result $p_{\frac{(a+b\sqrt{c})}{d}} = (\tilde{a} + \tilde{b}\sqrt{c})/\tilde{d}$. Let $\delta = 1$ when k is odd and $\delta = 0$ when k is even. Show that the following optimization can be used for the virtual substitution. Assume $d \neq 0 \wedge c \geq 0$ for well-definedness. For square-root-free

expressions ($b = 0$) with just divisions, i.e. those of the form $(a + 0\sqrt{c})/d$, the following equivalences hold:

$$\begin{aligned}(a + 0\sqrt{c})/d = 0 &\equiv a = 0 \\ (a + 0\sqrt{c})/d \leq 0 &\equiv ad^\delta \leq 0 \\ (a + 0\sqrt{c})/d < 0 &\equiv ad^\delta < 0 \\ (a + 0\sqrt{c})/d \neq 0 &\equiv a \neq 0\end{aligned}$$

Assume $d \neq 0 \wedge c \geq 0$ for well-definedness. For square root expressions $(a + b\sqrt{c})/d$ with arbitrary b , the following equivalences hold:

$$\begin{aligned}(a + b\sqrt{c})/d = 0 &\equiv ab \leq 0 \wedge a^2 - b^2c = 0 \\ (a + b\sqrt{c})/d \leq 0 &\equiv ad^\delta \leq 0 \wedge a^2 - b^2c \geq 0 \vee bd^\delta \leq 0 \wedge a^2 - b^2c \leq 0 \\ (a + b\sqrt{c})/d < 0 &\equiv ad^\delta < 0 \wedge a^2 - b^2c > 0 \vee bd^\delta \leq 0 \wedge (ad^\delta < 0 \vee a^2 - b^2c < 0) \\ (a + b\sqrt{c})/d \neq 0 &\equiv ab > 0 \vee a^2 - b^2c \neq 0\end{aligned}$$

References

- [BCR98] Jacek Bochnak, Michel Coste, and Marie-Francoise Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer, 1998.
- [BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2nd edition, 2006. [doi:10.1007/3-540-33099-2](https://doi.org/10.1007/3-540-33099-2).
- [CC56] Claude Chevalley and Henri Cartan. Schémas normaux; morphismes; ensembles constructibles. In *Séminaire Henri Cartan*, volume 8, pages 1–10. Numdam, 2955-1956.
- [CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, 1991.
- [Chu36] Alonzo Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.
- [CLO92] David A. Cox, John Little, and Donal O’Shea. *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, New York, 1992.
- [Coh69] Paul J. Cohen. Decision procedures for real and p -adic fields. *Communications in Pure and Applied Mathematics*, 22:131–151, 1969.
- [Col75] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

- [dMP13] Leonardo Mendonça de Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2013.
- [DS97] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bull.*, 31:2–9, 1997.
- [Eng93] E. Engeler. *Foundations of Mathematics: Questions of Analysis, Geometry and Algorithmics*. Springer, 1993.
- [Göd30] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Mon. hefte Math. Phys.*, 37:349–360, 1930.
- [Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys.*, 38:173–198, 1931.
- [Har95] Joe Harris. *Algebraic Geometry: A First Course*. Graduate Texts in Mathematics. Springer, 1995.
- [Her30] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie, Class III, Sciences Mathématiques et Physiques*, 33:33–160, 1930.
- [Hör83] L. Hörmander. *The Analysis of Linear Partial Differential Operators II*, volume 257 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1983.
- [JdM12] Dejan Jovanovic and Leonardo Mendonça de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *LNCS*, pages 339–354. Springer, 2012.
- [KK71] Georg Kreisel and Jean-Louis Krivine. *Elements of mathematical logic: Model Theory*. North-Holland, 2 edition, 1971.
- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Pas11] Grant Olney Passmore. *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*. PhD thesis, School of Informatics, University of Edinburgh, 2011.
- [Pla07] André Platzer. Combining deduction and algebraic constraints for hybrid system analysis. In Bernhard Beckert, editor, *VERIFY’07 at CADE, Bremen, Germany*, volume 259 of *CEUR Workshop Proceedings*, pages 164–178. CEUR-WS.org, 2007.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:10.1109/LICS.2012.64.

- [Pla12b] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:10.1109/LICS.2012.13.
- [PQ08] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. doi:10.1007/978-3-540-71070-7_15.
- [PQR09] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 485–501. Springer, 2009. doi:10.1007/978-3-642-02959-2_35.
- [Ric53] H. Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Trans. AMS*, 89:25–59, 1953.
- [Rob49] Julia Robinson. Definability and decision problems in arithmetic. *J. Symb. Log.*, 14(2):98–114, 1949.
- [Rob77] Abraham Robinson. *Complete Theories*. Studies in logic and the foundations of mathematics. North-Holland, 2 edition, 1977.
- [Sei54] Abraham Seidenberg. A new decision method for elementary algebra. *Annals of Mathematics*, 60:365–374, 1954.
- [Tar31] Alfred Tarski. Sur les ensembles définissables de nombres réels I. *Fundam. Math.*, 17:210–239, 1931.
- [Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.
- [Tur37] Alan M. Turing. Computability and lambda-definability. *J. Symb. Log.*, 2(4):153–163, 1937.
- [Wei97] Volker Weispfenning. Quantifier elimination for real algebra — the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997.

Lecture Notes on Virtual Substitution & Real Arithmetic

[André Platzer](#)

Carnegie Mellon University
Lecture 19

1. Introduction

Reasoning about cyber-physical systems and hybrid systems requires understanding and handling their real arithmetic, which can be challenging, because cyber-physical systems can have complex behavior. Differential dynamic logic and its proof calculus [[Pla08](#), [Pla10](#), [Pla12](#)] reduce the verification of hybrid systems to real arithmetic. How arithmetic interfaces with proofs has already been discussed in [Lecture 9 on Proofs & Arithmetic](#). How real arithmetic with linear and quadratic equations can be handled by virtual substitution has been shown in [Lecture 18 on Virtual Substitution & Real Equations](#). Today's lecture shows how virtual substitution for quantifier elimination in real arithmetic extends to the case of linear and quadratic inequalities.

These lecture notes are loosely based on [[Wei97](#), [Pla10](#), Appendix D]. They add substantial intuition and motivation that is helpful for following the technical development. More information about virtual substitution can be found in the literature [[Wei97](#)]. See, e.g., [[PQR09](#), [Pas11](#)] for an overview of other techniques for real arithmetic.

The most important learning goals of this lecture are:

Modeling and Control: This lecture refines the indirect impact that the previous lecture had on CPS models and controls by informing the reader about the consequences of the analytic complexity resulting from different arithmetical modeling tradeoffs. There are subtle analytic consequences from different arithmetic formulations of similar questions that can have an impact on finding the right tradeoffs for expressing a CPS model. A safe distance of car x to a stopping light m could equally well be captured as $x \leq m$ or as $x < m$, for example.

Computational Thinking: The primary purpose of today's lecture is to understand how arithmetical reasoning, which is crucial for CPS, can be done rigorously and

automatically not just for the equations considered in [Lecture 18 on Virtual Substitution & Real Equations](#) but also for inequalities. While formulas involving sufficiently many quadratic equations among other inequalities could already be handled using the techniques from [Lecture 18](#), such extensions are crucial for proving arithmetic formulas that involve only inequalities, which happens rather frequently in a world of CPS where many questions concern inequality bounds on distances. Developing an intuition for the working principles of real arithmetic decision procedures can be very helpful for developing strategies to verify CPS models at scale. We will again see the conceptually very important device in the logical trinity: the flexibility of moving back and forth between syntax and semantics at will. Virtual substitutions will again allow us to move back and forth at will between syntax and semantics. This time, square roots will not be all there is to it, but the logical trinity will lead us to ideas from nonstandard analysis to bridge the gap between semantic operations that are inexpressible otherwise in first-order logic of real arithmetic.

CPS Skills: The author is not aware of any impact that this lecture has on CPS skills, because this lecture focuses on the handling arithmetic in CPS analysis.

2. Recall: Square Root $\sqrt{\cdot}$ Substitutions for Quadratics

Recall the way to handle quantifier elimination for linear or quadratic equations from [Lecture 18 on Virtual Substitution & Real Equations](#) by virtually substituting in its symbolic solutions $x = -c/b$ or $x = (-b \pm \sqrt{b^2 - 4ac})/(2a)$, respectively:

Theorem 1 (Virtual substitution of quadratic equations). *For a quantifier-free formula F with $x \notin a, b, c$, the following equivalence is valid over \mathbb{R} :*

$$\begin{aligned}
 a \neq 0 \vee b \neq 0 \vee c \neq 0 \rightarrow \\
 & \left(\exists x (ax^2 + bx + c = 0 \wedge F) \leftrightarrow \right. \\
 & \quad a = 0 \wedge b \neq 0 \wedge F_{\hat{x}}^{-c/b} \\
 & \quad \left. \vee a \neq 0 \wedge b^2 - 4ac \geq 0 \wedge \left(F_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} \vee F_{\hat{x}}^{(-b-\sqrt{b^2-4ac})/(2a)} \right) \right) \\
 & \hspace{15em} (1)
 \end{aligned}$$

When using virtual substitutions of square roots from [Lecture 18](#), the resulting formula on the right-hand side of the biimplication is quantifier-free and can be chosen for QE($\exists x (ax^2 + bx + c = 0 \wedge F)$) as long as it is not the case that $a = b = c = 0$. In case $a = b = c = 0$, another formula in F needs to be considered for directing quantifier elimination by commuting and reassociating \wedge , because the equation $ax^2 + bx + c = 0$ is noninformative if $a = b = c = 0$, e.g. when a, b, c are the zero polynomials or even if they just have a common root.

The equivalent formula on the right-hand side of the biimplication in (1) is a formula in the first-order logic of real arithmetic when using the virtual substitution of square root expressions defined in [Lecture 18](#).

3. Infinity ∞ Substitution

Theorem 1 addresses the case where the quantified variable occurs in a linear or quadratic equation, in which case it is efficient to use Theorem 1, because there are at most 3 symbolic points to consider corresponding to the respective solutions of the equation. The quantified variable might only occur in inequalities, however. Consider a formula of the form

$$\exists x (ax^2 + bx + c \leq 0 \wedge F) \quad (x \notin a, b, c) \quad (2)$$

where x does not occur in a, b, c . Under the respective conditions from Theorem 1, the possible solutions $-c/b$, $(-b + \sqrt{d})/(2a)$, $(-b - \sqrt{d})/(2a)$ from (1) continue to be options for solutions of (2), because one way of satisfying the weak inequality $ax^2 + bx + c \leq 0$ is by satisfying the equation $ax^2 + bx + c = 0$. So if F is *true* for any of those solutions of the quadratic equation (under the auspices of the additional constraints on a, b, c), then (2) holds as well.

Yet, if those points do not work out, the weak inequality in (2) allows for more possible solutions than the equation does. For example, if $a = 0, b > 0$, then sufficiently small values of x would satisfy $0x^2 + bx + c \leq 0$. Also, if $a < 0$, then sufficiently small values of x would satisfy $ax^2 + bx + c \leq 0$, because x^2 grows faster than x and, thus the negative ax^2 ultimately overcomes any contribution of bx and c to the value of $ax^2 + bx + c$. But if we literally substituted each such smaller value of x into F , that would quickly diverge into the full substitution $\bigvee_{t \in T} F_x^t$ for the un insightful case of $T \stackrel{\text{def}}{=} \mathbb{R}$ from [Lecture 18](#).

Now, one possibility of pursuing this line of thought may be to substitute smaller and smaller values for x into (2) and see if one of those happens to work. There is a much better way though. The only really small value that would have to be substituted into (2) for x to see if it happens to work is one that is so negative that it is smaller than all others: $-\infty$, which is the lower limit of all negative real numbers. Alternatively, $-\infty$ can be understood as being “always as negative as needed, i.e. more negative than anything else”. Think of $-\infty$ as being built out of elastic rubber so that it always ends up being smaller when being compared to any actual real number, because the elastic number $-\infty$ simply shrinks every time it is being compared to any other number. Analogously, ∞ is the upper limit of all real numbers or “always as positive as needed, i.e. more positive than anything else”. The elastic rubber version of understanding ∞ is such that ∞ always grows as needed every time it is being compared to any other number.

Let $\infty, -\infty$ be *positive and negative infinities*, respectively, i.e. choose extra elements $\infty, -\infty \notin \mathbb{R}$ with $-\infty < r < \infty$ for all $r \in \mathbb{R}$. Formulas of real arithmetic can be substituted with $\pm\infty$ for a variable x in the compactified reals $\mathbb{R} \cup \{\infty, -\infty\}$. Yet, just like

with square root expressions, $\pm\infty$ do not actually need to ever occur in the resulting formula, because substitution of infinities into formulas can be defined differently. For example, $(x + 5 > 0)_x^\infty$ simplifies to *false*, while $(x + 5 < 0)_x^\infty$ simplifies to *true*.

Note 2. Substitution of the infinity $-\infty$ for x into an atomic formula for a polynomial $p \stackrel{\text{def}}{=} \sum_{i=0}^n a_i x^i$ with polynomials a_i that do not contain x is defined by the following equivalences (accordingly for substituting ∞ for x).

$$(p = 0)_{\hat{x}}^{-\infty} \equiv \bigwedge_{i=0}^n a_i = 0 \quad (3)$$

$$(p \leq 0)_{\hat{x}}^{-\infty} \equiv (p < 0)_{\hat{x}}^{-\infty} \vee (p = 0)_{\hat{x}}^{-\infty} \quad (4)$$

$$(p < 0)_{\hat{x}}^{-\infty} \equiv p(-\infty) < 0 \quad (5)$$

$$(p \neq 0)_{\hat{x}}^{-\infty} \equiv \bigvee_{i=0}^n a_i \neq 0 \quad (6)$$

Lines (3) and its dual (6) use that the only equation of real arithmetic that infinities $\pm\infty$ satisfy is the trivial equation $0 = 0$. Line (4) uses the equivalence $p \leq 0 \equiv p < 0 \vee p = 0$ and is equal to $(p < 0 \vee p = 0)_{\hat{x}}^{-\infty}$ by the substitution base from [Lecture 18](#). Line (5) uses a simple inductive definition based on the degree, $\deg(p)$, in the variable x of the polynomial p to characterize whether p is ultimately negative at $-\infty$ (or for sufficiently negative numbers):

Note 3. Let $p \stackrel{\text{def}}{=} \sum_{i=0}^n a_i x^i$ with polynomials a_i that do not contain x . Whether p is ultimately negative at $-\infty$, suggestively written $p(-\infty) < 0$, is easy to characterize by induction on the degree of the polynomial:

$$p(-\infty) < 0 \stackrel{\text{def}}{\equiv} \begin{cases} p < 0 & \text{if } \deg(p) = 0 \\ (-1)^n a_n < 0 \vee (a_n = 0 \wedge (\sum_{i=0}^{n-1} a_i x^i)(-\infty) < 0) & \text{if } \deg(p) > 0 \end{cases}$$

$p(-\infty) < 0$ is true in a state in which $\lim_{x \rightarrow -\infty} p(x) < 0$.

The first line captures that the sign of polynomials of degree 0 in the variable x does not depend on x , so $p(-\infty) < 0$ iff the polynomial that has degree 0 in x and, thus, only consists of a term $p = a_0$ that is constant in x , is negative (which may still depend on the value of other variables in a_0 but not on x). The second line captures that the sign at $-\infty$ of a polynomial of degree $n = \deg(p) > 0$ is determined by the degree-modulated sign of its leading coefficient a_n , because for x of sufficiently big value, the value of $a_n x^n$ will dominate all lower-degree values, whatever their coefficients are. For even $n > 0$, $x^n < 0$ while $x^n > 0$ for odd n at $-\infty$. In case the leading coefficient a_n evaluates to zero, the value of p at $-\infty$ depends on the value at $-\infty$ of the remaining polynomial $\sum_{i=0}^{n-1} a_i x^i$ of lower degree, which can be determined recursively as $(\sum_{i=0}^{n-1} a_i x^i)(-\infty) < 0$.

Substitution of ∞ for x into an atomic formula is defined similarly, except that the sign factor $(-1)^n$ disappears, because $x^n > 0$ at ∞ whatever value $n > 0$ has. Substitution of ∞ or of $-\infty$ for x into other first-order formulas is then defined on this basis as in [Lecture 18](#).

Example 2 (Sign of quadratic polynomials at $-\infty$). Using this principle to check under which circumstance the quadratic inequality from (2) evaluates to *true* yields the answer from our earlier ad-hoc analysis of what happens for sufficiently small values of x :

$$\begin{aligned} (ax^2 + bx + c \leq 0)_{\hat{x}}^{-\infty} &\equiv (-1)^2 a < 0 \vee a = 0 \wedge ((-1)b < 0 \vee b = 0 \wedge c < 0) \\ &\equiv a < 0 \vee a = 0 \wedge (b > 0 \vee b = 0 \wedge c < 0) \end{aligned}$$

One representative example for each of those cases is illustrated in Fig. 1. In the same way, the virtual substitution can be used to see under which circumstance the remainder formula F from (2) also evaluates to *true* for sufficiently small values of x , exactly when $F_{\hat{x}}^{-\infty}$ holds. In contrast, note that (at least for $a \neq 0$), the virtual substitution of ∞ for x would not make sense to check (2) at, because in that case, the inequality $ax^2 + bx + c \leq 0$ is violated, as can be confirmed by checking $(ax^2 + bx + c \leq 0)_{\hat{x}}^{\infty}$. That would be different if the inequality were $ax^2 + bx + c \geq 0$.

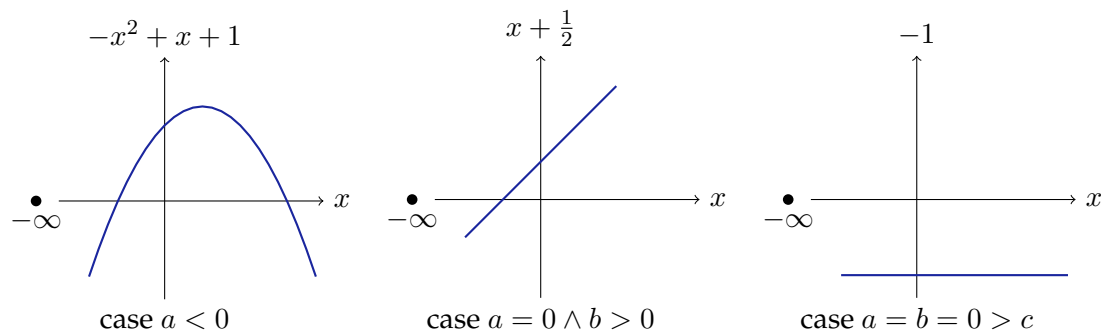


Figure 1: Illustration of the value of different quadratic functions p where $p_{\hat{x}}^{-\infty} \equiv \text{true}$

The crucial thing to note is again that the *virtual substitution* of infinities $\pm\infty$ for x in F giving $F_{\hat{x}}^{\pm\infty}$ is semantically equivalent to the result $F_x^{\pm\infty}$ of the literal substitution replacing x with $\pm\infty$, but operationally different, because the virtual substitution never introduces actual infinities. Because of their semantical equivalence, we use the same notation by abuse of notation.

Lemma 3 (Virtual substitution lemma for infinities). *The result $F_{\hat{x}}^{-\infty}$ of the virtual substitution is semantically equivalent to the the result $F_x^{-\infty}$ of the literal substitution, but better behaved, because it stays within $\text{FOL}_{\mathbb{R}}$ proper. Essentially, the following equivalence of virtual substitution and literal substitution for infinities is valid:*

$$F_x^{-\infty} \leftrightarrow F_{\hat{x}}^{-\infty}$$

Keep in mind that the result $F_{\hat{x}}^{-\infty}$ of virtual substitution is a proper formula of $\text{FOL}_{\mathbb{R}}$, while the literal substitution $F_x^{-\infty}$ could only be considered a formula in an extended logic such as $\text{FOL}_{\mathbb{R} \cup \{-\infty, \infty\}}$ that allows for infinite quantities. The same property holds for $F_{\hat{x}}^{\infty}$.

Note that the situation is, in a sense, the converse of [Lecture 18](#), where the square root expressions were already in the semantic domain \mathbb{R} , and just had to be made accessible in the syntactic formulas via virtual substitutions. In [Lemma 3](#), instead, virtual substitutions already know more about infinities $\pm\infty$ than the semantic domain \mathbb{R} does, which is why the semantic domain needs an extension to $\mathbb{R} \cup \{-\infty, \infty\}$ for the alignment in [Lemma 3](#).

Expedition 1 (Infinite challenges with infinities in extended reals $\mathbb{R} \cup \{-\infty, \infty\}$). The set $\mathbb{R} \cup \{-\infty, \infty\}$ is seemingly easily written down as a semantic domain of extended reals. What exactly do we mean by it? The set of reals to which we adjoin two new elements, denoted $-\infty$ and ∞ which are the minimum and maximum element of the ordering \leq :

$$\forall x (-\infty \leq x \leq \infty) \quad (7)$$

This turns $\mathbb{R} \cup \{-\infty, \infty\}$ into a complete lattice, because every subset has a supremum and an infimum. The extended reals are a compactification of \mathbb{R} . But where does that leave the other arithmetic properties of \mathbb{R} ? What is $\infty + 1$ or $\infty + x$ when ∞ is already infinitely big? The compatibility of \leq with $+$ expects $\infty \leq \infty + x$ at least for all $x \geq 0$. By (7) also $\infty + x \leq \infty$. Because ∞ is so infinitely big, the same $\infty + x = \infty$ is expected even for all x , except $-\infty$. The compatibility of \leq with \cdot expects $\infty \leq \infty \cdot x$ at least for all $x \geq 1$. By (7) also $\infty \cdot x \leq \infty$. Since ∞ is infinitely big, the same $\infty \cdot x = \infty$ is expected even for all $x > 0$.

$\infty + x = \infty$	for all $x \neq -\infty$
$-\infty + x = -\infty$	for all $x \neq \infty$
$\infty \cdot x = \infty$	for all $x > 0$
$\infty \cdot x = -\infty$	for all $x < 0$
$-\infty \cdot x = -\infty$	for all $x > 0$
$-\infty \cdot x = \infty$	for all $x < 0$

This extension sounds reasonable. But the resulting set $\mathbb{R} \cup \{-\infty, \infty\}$ is not a field. Otherwise ∞ would have an additive inverse. But what x would satisfy $\infty + x = 0$? One might guess $x = -\infty$, but then one would also expect $0 = \infty + (-\infty) = \infty + (-\infty + 1) = ((\infty + (-\infty)) + 1 = 0 + 1 = 1$, which is not a good idea to adopt for proving anything in a sound way. Instead, problematic terms remain explicitly undefined.

$$\begin{aligned}\infty - \infty &= \text{undefined} \\ 0 \cdot \infty &= \text{undefined} \\ \pm\infty / \pm\infty &= \text{undefined} \\ 1/0 &= \text{undefined}\end{aligned}$$

Since these conventions make infinities somewhat subtle, we happily remember that the only thing we need them for is to make sense of inserting sufficiently negative (or sufficiently positive) numbers into inequalities to satisfy them. That is still mostly harmless.

4. Infinitesimal ε Substitutions

Theorem 1 addresses the case where the quantified variable occurs in a linear or quadratic equation and the virtual substitution in Sect. 3 adds the case of sufficiently small values for x to handle $ax^2 + bx + c \leq 0$. Consider a formula of the form

$$\exists x (ax^2 + bx + c < 0 \wedge F) \quad (x \notin a, b, c) \quad (8)$$

In this case, the roots from Theorem 1 will not help, because they satisfy the equation $ax^2 + bx + c = 0$ but not the strict inequality $ax^2 + bx + c < 0$. The virtual substitution of $-\infty$ for x from Sect. 3 still makes sense to consider, because the arbitrarily small negative numbers that it corresponds to might satisfy F and $ax^2 + bx + c < 0$. If $-\infty$ does not work, however, the solution of (8) could be near one of the roots of $ax^2 + bx + c = 0$, just slightly off so that $ax^2 + bx + c < 0$ is actually satisfied rather than $ax^2 + bx + c = 0$. How far off? Well, saying that exactly by any real number is again difficult, because any particular real number might already have been too large in absolute value, depending on the constraints in the remainder of F . Again, this calls for quantities that are always as small as we need them to be.

Sect. 3 used a negative quantity that is so small that it is smaller than all negative numbers and hence infinitely small (but infinitely large in absolute value). The negative infinity $-\infty$ that is smaller no matter what other number we compare it with. Analyzing (8) needs *positive* quantities that are infinitely small and hence also infinitely small in absolute value. Infinitesimals are positive quantities that are always smaller than all positive real numbers, i.e. “always as small as needed”. Think of them as built out of elastic rubber so that they always shrink as needed when compared with any actual positive real number so that the infinitesimals end up being smaller than posi-

tive reals. Another way of looking at infinitesimals is that they are the multiplicative inverses of $\pm\infty$.

A positive infinitesimal $\infty > \varepsilon > 0$ is positive and an extended real that is infinitesimal, i.e. positive but smaller than all positive real numbers ($\varepsilon < r$ for all $r \in \mathbb{R}$ with $r > 0$). For all non-zero polynomials $p \in \mathbb{R}[x] \setminus \{0\}$, $\zeta \in \mathbb{R}$, the Taylor series

$$p(\zeta + \varepsilon) = \sum_{n=0}^{\infty} \frac{p^{(n)}(\zeta)}{n!} (\zeta + \varepsilon - \zeta)^n = \sum_{n=0}^{\infty} \frac{p^{(n)}(\zeta)}{n!} \varepsilon^n = \sum_{n=0}^{\deg(p)} \frac{p^{(n)}(\zeta)}{n!} \varepsilon^n$$

of p around ζ evaluated at $\zeta + \varepsilon$ (note that ε is small enough to be in the domain of convergence of the Taylor series) can be used to show:

1. $p(\zeta + \varepsilon) \neq 0$
that is, infinitesimals ε are always so small that they never yield roots of any equation, except the trivial zero polynomial. Whenever it looks like there might be a root, the infinitesimal just became a bit smaller to avoid satisfying the equation. And nonzero univariate polynomials $p(x)$ only have finitely many roots, so the infinitesimals will take care to avoid all of them by becoming just a little smaller.
2. $p(\zeta) \neq 0 \Rightarrow p(\zeta)p(\zeta + \varepsilon) > 0$,
that is, p has constant sign on infinitesimal neighborhoods of nonroots ζ . If the neighborhood around ζ is small enough (and for an infinitesimal it will be), then the polynomial will not yet have changed sign on that interval, because the sign will only change after passing one of the roots.
3. $0 = p(\zeta) = p'(\zeta) = p''(\zeta) = \dots = p^{(k-1)}(\zeta) \neq p^{(k)}(\zeta) \Rightarrow p^{(k)}(\zeta)p(\zeta + \varepsilon) > 0$,
that is the first nonzero derivative of p at ζ determines the sign of p in a small enough neighborhoods of ζ (infinitesimal neighborhoods will be small enough), because the sign will only change after passing one of the roots.

Note 5. Substitution of an infinitesimal expression $e + \varepsilon$ with a square root expression $e = (a + b\sqrt{c})/d$ and a positive infinitesimal ε for x into a polynomial $p = \sum_{i=0}^n a_i x^i$ with polynomials a_i that do not contain x is defined by the following equivalences.

$$(p = 0)_{\hat{x}}^{e+\varepsilon} \equiv \bigwedge_{i=0}^n a_i = 0 \quad (9)$$

$$(p \leq 0)_{\hat{x}}^{e+\varepsilon} \equiv (p < 0)_{\hat{x}}^{e+\varepsilon} \vee (p = 0)_{\hat{x}}^{e+\varepsilon} \quad (10)$$

$$(p < 0)_{\hat{x}}^{e+\varepsilon} \equiv (p^+ < 0)_{\hat{x}}^e \quad (11)$$

$$(p \neq 0)_{\hat{x}}^{e+\varepsilon} \equiv \bigvee_{i=0}^n a_i \neq 0 \quad (12)$$

Lines (9) and its dual (12) use that infinitesimals offsets satisfy no equation except the trivial equation $0=0$ (case 1), which makes infinitesimals and infinities behave the same

as far as equations go. Line (10) again uses the equivalence $p \leq 0 \equiv p < 0 \vee p = 0$. Line (11) checks whether the sign of p at the square root expression e is already negative (which will make p inherit the same negative sign after an infinitesimal offset at $e + \varepsilon$ by case 2) or will immediately become negative right away using a recursive formulation of immediately becoming negative that uses higher derivatives (which determine the sign by case 3). The lifting to arbitrary quantifier-free formulas of real arithmetic is again by substitution into all atomic subformulas and equivalences such as $(p > q) \equiv (p - q > 0)$ as defined in Lecture 18. Note that, for the case $(p < 0)_{\hat{x}}^{e+\varepsilon}$, the (non-infinitesimal) square root expression e gets virtually substituted in for x into a formula $p^+ < 0$, which characterizes whether p becomes negative immediately at or after x (which will be virtually substituted by the intended square root expression e momentarily).

Note 6. Whether p is immediately negative at x , i.e. negative itself or of a derivative p' that makes it negative on an infinitesimal interval $[x, x + \varepsilon]$, suggestively written $p^+ < 0$, can be characterized recursively:

$$p^+ < 0 \stackrel{\text{def}}{\equiv} \begin{cases} p < 0 & \text{if } \deg(p) = 0 \\ p < 0 \vee (p = 0 \wedge (p')^+ < 0) & \text{if } \deg(p) > 0 \end{cases}$$

$p^+ < 0$ is true in a state in which $\lim_{y \rightarrow x^+} p(x) = \lim_{y \searrow x} p(x) = \lim_{y \rightarrow x} p(x) < 0$ holds for the limit of p at x from the right.

The first line captures that the sign of polynomials of degree 0 in the variable x does not depend on x , so they are negative at x iff the polynomial $p = a_0$ that has degree 0 in x is negative (which may still depend on the value of other variables in a_0). The second line captures that the sign at $x + \varepsilon$ of a non-constant polynomial is still negative if it is negative at x (because $x + \varepsilon$ is not far enough away from x for any sign changes by case 2) or if x is a root of p but its derivative p' at x is immediately negative, since the first nonzero derivative at x determines the sign near x by case 3.

Example 4 (Sign of quadratic polynomials after second root). Using this principle to check under which circumstance the quadratic strict inequality from (8) evaluates to true at $(-b + \sqrt{b^2 - 4ac})/(2a) + \varepsilon$, i.e. right after its root $(-b + \sqrt{b^2 - 4ac})/(2a)$, leads to the following computation.

$$(ax^2 + bx + c)^+ < 0 \equiv ax^2 + bx + c < 0 \vee ax^2 + bx + c = 0 \wedge (2ax + b < 0 \vee 2ax + b = 0 \wedge 2a < 0)$$

with successive derivatives to break ties (i.e. zero signs in previous derivatives). Hence,

$$\begin{aligned}
 (ax^2 + bx + c < 0)_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)+\varepsilon} &\equiv ((ax^2 + bx + c)^+ < 0)_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} \equiv \\
 (ax^2 + bx + c < 0 \vee ax^2 + bc + c = 0 \wedge (2ax + b < 0 \vee 2ax + b = 0 \wedge 2a < 0))_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} &\equiv \\
 \equiv 0 \cdot 1 < 0 \vee 0 = 0 \wedge \underbrace{(0 < 0 \vee 4a^2 \leq 0 \wedge (0 < 0 \vee -4a^2(b^2 - 4ac) < 0))}_{(2ax+b<0)_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)}} \vee \underbrace{0 = 0}_{(2ax+b=0)_{\hat{x}}} \wedge \underbrace{2a < 0}_{2a < 0} &\equiv \\
 \equiv 4a^2 \leq 0 \wedge -4a^2(b^2 - 4ac) < 0 \vee 2a < 0 &
 \end{aligned}$$

because the square root virtual substitution of its own root $(-b + \sqrt{b^2 - 4ac})/(2a)$ into $ax^2 + bx + c$ gives $(ax^2 + bx + c)_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} = 0$ by construction (compare example from [Lecture 18](#)). The virtual substitution into the other polynomial $2ax + b$ above computes as follows:

$$\begin{aligned}
 (2ax + b)_{\hat{x}}^{(-b \pm \sqrt{b^2 - 4ac})/(2a)} &\equiv 2a \cdot (-b \pm \sqrt{b^2 - 4ac})/(2a) + b \\
 &= (-2ab \pm 2a\sqrt{b^2 - 4ac})/(2a) + b \\
 &= (\cancel{-2ab} + \cancel{2ab} \pm 2a\sqrt{b^2 - 4ac})/(2a) \\
 &= (0 \pm 2a\sqrt{b^2 - 4ac})/(2a)
 \end{aligned}$$

The resulting formula can be further simplified internally to

$$(ax^2 + bx + c < 0)_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)+\varepsilon} \equiv 4a^2 \leq 0 \wedge -4a^2(b^2 - 4ac) < 0 \vee 2a < 0 \equiv 2a < 0$$

because the first conjunct $4a^2 \leq 0 \equiv a = 0$ and, with $a = 0$, the second conjunct simplifies to $-4a^2(b^2 - 4ac)_a^0 = -0(b^2) < 0$, which is impossible in the reals. This answer makes sense. Because, indeed, exactly if $2a < 0$ will a quadratic polynomial still evaluate to $ax^2 + bx + c < 0$ right after its second root $(-b + \sqrt{b^2 - 4ac})/(2a)$. Fig. 2 illustrates to how this relates to the parabola point downwards, because of $2a < 0$.

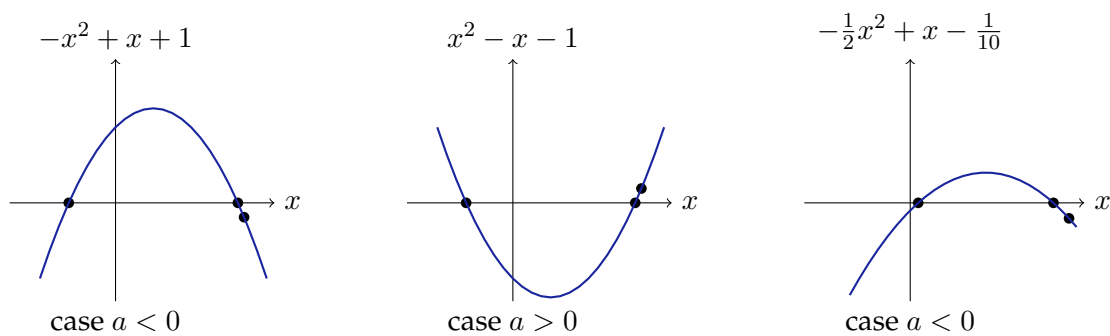


Figure 2: Illustration of the sign after the second root for quadratic functions p

Formulas such as this one ($2a > 0$) are the result of a quantifier elimination procedure. If the formula after quantifier elimination is either *true* or *false*, then you know for sure that the formula is valid (*true*) or unsatisfiable (*false*), respectively. If the result of quantifier elimination is *true*, for example, KeYmaera can close proof branches (marked by proof rule \mathbb{R} in our sequent proofs). Yet, quantifier elimination can also return other formulas, such as $2a > 0$, which are equivalent to the formula where quantifier elimination has been applied. In particular, they identify exactly under which circumstance that respective quantified formula is true. This can be very useful for identifying the missing assumptions to make a proof work and the corresponding statement true.

Note 7 (Quantifier elimination identifies requirements). *If the result of quantifier elimination is true, the corresponding formula is valid. If it is false, the corresponding formula is not valid (and even unsatisfiable). In between, i.e. when quantifier elimination results in a logical formula that is sometimes false and sometimes true, then this formula identifies exactly the missing requirements that are needed to make the desired formula true. This can be useful to synthesize missing requirements. Take care, however, not to work with universal closures, in which case true and false are the only possible outcomes.*

The crucial thing to note about the process is again that the *virtual substitution* of infinitesimal expressions $e + \varepsilon$ for x in F giving $F_{\hat{x}}^{e+\varepsilon}$ is semantically equivalent to the result $F_x^{e+\varepsilon}$ of the literal substitution replacing x with $e + \varepsilon$, but operationally different, because the virtual substitution never introduces actual infinitesimals. Because of their semantical equivalence, we use the same notation by abuse of notation.

Lemma 5 (Virtual substitution lemma for infinitesimals). *The result $F_{\hat{x}}^{e+\varepsilon}$ of the virtual substitution is semantically equivalent to the the result $F_x^{e+\varepsilon}$ of the literal substitution, but better behaved, because it stays within $\text{FOL}_{\mathbb{R}}$ proper. Essentially, the following equivalence of virtual substitution and literal substitution for infinitesimals is valid:*

$$F_x^{e+\varepsilon} \leftrightarrow F_{\hat{x}}^{e+\varepsilon}$$

Keep in mind that the result $F_{\hat{x}}^{e+\varepsilon}$ of virtual substitution is a proper formula of $\text{FOL}_{\mathbb{R}}$, while the literal substitution $F_x^{e+\varepsilon}$ could only be considered a formula in an extended logic such as $\text{FOL}_{\mathbb{R}[\varepsilon]}$ that allows for infinitesimal quantities from nonstandard analysis.

Computationally more efficient substitutions of infinitesimals have been reported elsewhere [BD07].

Expedition 2 (Nonstandard analysis: infinite challenges with infinitesimal ε). Infinite quantities in the extended reals $\mathbb{R} \cup \{-\infty, \infty\}$ already needed some attention to stay away from undefined expressions. Infinitesimals are infinitely more subtle than infinities. Real numbers are Archimedean, i.e. for every non-zero $x \in \mathbb{R}$, there

is an $n \in \mathbb{N}$ such that

$$\underbrace{|x + x + \dots + x|}_{n \text{ times}} > 1$$

Infinitesimals are non-Archimedean, because it does not matter how often you add ε , it still won't sum to one. There is a myriad of ways of making sense of infinitesimal quantities in nonstandard analysis, including surreal numbers, superreal numbers, and hyperreals. In a sense, infinitesimal quantities can be considered as multiplicative inverses of infinities but bring up many subtleties. For example, if an infinitesimal ε is added to \mathbb{R} , then the following terms need to denote values and satisfy ordering relations:

$$\varepsilon^2 \quad \varepsilon \quad x^2 + \varepsilon \quad (x + \varepsilon)^2 \quad x^2 + 2\varepsilon x + 5\varepsilon + \varepsilon^2$$

Fortunately, a rather tame version of infinitesimals is enough for the context of virtual substitution. The crucial properties of infinitesimals we need are [dMP13]:

$$\begin{aligned} \varepsilon &> 0 \\ \forall x \in \mathbb{R} (x > 0 \rightarrow \varepsilon < x) \end{aligned}$$

That is, the infinitesimal ε is positive and smaller than all positive reals.

5. Quantifier Elimination by Virtual Substitution

The following quantifier elimination technique due to Weispfenning [Wei97] works for formulas with a quantified variable that occurs at most quadratically.

Theorem 6 (Virtual substitution of quadratic constraints [Wei97]). *Let F be a quantifier-free formula in which all atomic formulas are of the form $ax^2 + bx + c \sim 0$ for x -free polynomials a, b, c (i.e. $x \notin a, b, c$) and $\sim \in \{=, \leq, <, \neq\}$, with corresponding discriminant $d \stackrel{\text{def}}{=} b^2 - 4ac$. Then $\exists x F$ is equivalent over \mathbb{R} to the following quantifier-free formula:*

$$\begin{aligned} &F_x^{-\infty} \\ \vee \bigvee_{(ax^2+bx+c \left\{ \begin{smallmatrix} = \\ \leq \end{smallmatrix} \right\} 0) \in F} & (a = 0 \wedge b \neq 0 \wedge F_x^{-c/b} \vee a \neq 0 \wedge d \geq 0 \wedge (F_x^{(-b+\sqrt{d})/(2a)} \vee F_x^{(-b-\sqrt{d})/(2a)})) \\ \vee \bigvee_{(ax^2+bx+c \left\{ \begin{smallmatrix} < \\ < \end{smallmatrix} \right\} 0) \in F} & (a = 0 \wedge b \neq 0 \wedge F_x^{-c/b+\varepsilon} \vee a \neq 0 \wedge d \geq 0 \wedge (F_x^{(-b+\sqrt{d})/(2a)+\varepsilon} \vee F_x^{(-b-\sqrt{d})/(2a)+\varepsilon})) \end{aligned}$$

Proof. The proof is an extended form of the proof reported in the literature [Wei97]. The proof first considers the literal substitution of square root expressions, infinities, and infinitesimals and then, as a second step, uses that the virtual substitutions that avoid square root expressions, infinities, and infinitesimals are equivalent (Lecture 18, Lemma 3 and 5). Let G denote the quantifier-free right-hand side so that the validity of

the following formula needs to be shown:

$$\exists x F \leftrightarrow G \quad (13)$$

The implication from the quantifier-free formula G to $\exists x F$ in (13) is obvious, because each disjunct of the quantifier-free formula has a conjunct of the form F_x^t for some (extended) term t even if it may be a square root expression or infinity or term involving infinitesimals. Whenever a formula of the form F_x^t is true, $\exists x F$ holds with that t as a witness, even when t is a square root expression, infinity, or infinitesimal.

The converse implication from $\exists x F$ to the quantifier-free formula G in (13) depends on showing that the quantifier-free formula G covers all possible representative cases and that the accompanying constraints on a, b, c, d are necessary so that they do not constrain solutions in unjustified ways.

One key insight is that it is enough to prove (13) for the case where all variables in F except x have concrete numeric real values, because the equivalence (13) is valid iff it true in all states. So considering one concrete state at a time is enough. By a fundamental property of real arithmetic called *o-minimality*, the set

$$\mathcal{S}(F) = \{\nu(x) \in \mathbb{R} : \nu \models F\}$$

of all real values for x that satisfy F forms a finite union of (pairwise disjoint) intervals, because the polynomials in F only change signs at their roots. There are only finitely many roots now that the polynomials have become univariate, i.e. with the only variable x , since all free variables are evaluated to concrete real numbers in ν . Without loss of generality (by merging overlapping or adjacent intervals), all those intervals are assume to be maximal, i.e. no bigger interval would satisfy F . So F actually changes its truth-value at the lower and upper endpoints of these intervals (unless the interval is unbounded).

The endpoints of these intervals are of the form $-c/b$, $(-b + \sqrt{d})/(2a)$, $(-b - \sqrt{d})/(2a)$ or ∞ , $-\infty$ for any of the polynomials $ax^2 + bx + c$ in F , because all polynomials in F are at most quadratic and all roots of those polynomials are of one of the above forms. In particular, if $-c/b$ is an endpoint of the intervals of $\mathcal{S}(F)$ for a polynomial $ax^2 + bx + c$ in F , then $a = 0$, $b \neq 0$, because that is the only case where $-c/b$ satisfies F , which has only at most quadratic polynomials. Likewise, if $(-b + \sqrt{d})/(2a)$ or $(-b - \sqrt{d})/(2a)$ are endpoints of intervals of $\mathcal{S}(F)$ for a polynomial $ax^2 + bx + c$ in F , then both imply that $a \neq 0$ and discriminant $d \geq 0$, otherwise there is no such solution in the reals. Consequently, all the side conditions for the roots in the quantifier-free formula G are necessary.

Now consider one interval $I \subseteq \mathcal{S}(F)$ (if there is none, $\exists x F$ is *false* and so will G be). If I has no lower bound in \mathbb{R} , then $F_{\hat{x}}^{-\infty}$ is true by construction (by Lemma 3, the virtual substitution $F_{\hat{x}}^{-\infty}$ is equivalent to the literal substitution $F_x^{-\infty}$ in $\pm\infty$ -extended real arithmetic). Otherwise, let $\alpha \in \mathbb{R}$ be the lower bound of I . If $\alpha \in I$ (i.e. I is closed at the lower bound), then α is of the form $-c/b$, $(-b + \sqrt{d})/(2a)$, $(-b - \sqrt{d})/(2a)$ for some equation $(ax^2 + bx + c = 0) \in F$ or some weak inequality $(ax^2 + bx + c \leq 0) \in F$

from F . Since the respective extra conditions on a, b, c, d hold, the quantifier-free formula G evaluates to true. If, otherwise, $\alpha \notin I$ (i.e. I is open at the lower bound α), then α is of the form $-c/b, (-b + \sqrt{d})/(2a), (-b - \sqrt{d})/(2a)$ for some disequation $(ax^2 + bx + c \neq 0) \in F$ or some strict inequality $(ax^2 + bx + c < 0) \in F$. Hence, the interval I cannot be a single point. Thus, one of the infinitesimal increments $-c/b + \varepsilon, (-b + \sqrt{d})/(2a) + \varepsilon, (-b - \sqrt{d})/(2a) + \varepsilon$ is in $I \subseteq S(F)$, because infinitesimals are smaller than all positive real numbers. Since the respective conditions a, b, c, d hold, the quantifier-free formula G is again true. Hence, in either case, the quantifier-free formula is equivalent to $\exists x F$ in state ν . Since the state ν giving concrete real numbers to all free variables of $\exists x F$ was arbitrary, the same equivalence holds for all ν , which means that the quantifier-free formula G is equivalent to $\exists x F$. That is $G \leftrightarrow \exists x F$ is valid, i.e. $\models G \leftrightarrow \exists x F$. \square

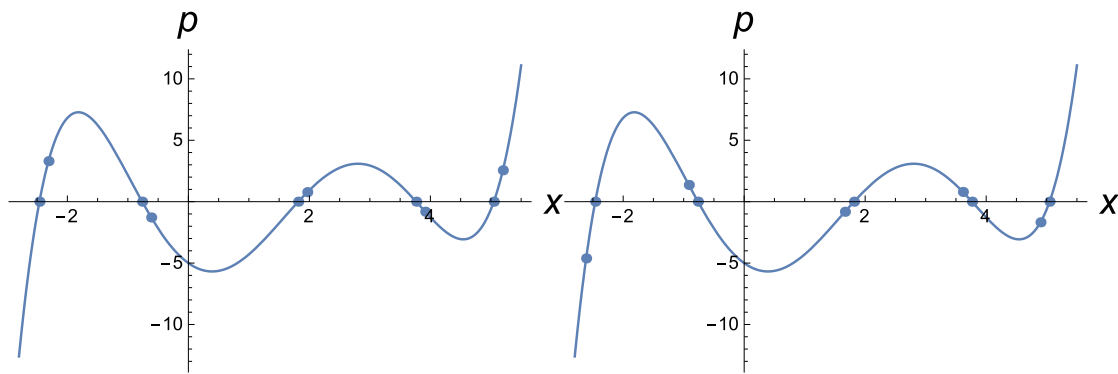


Figure 3: **(left)** Illustration of roots e and infinitesimal offsets $e + \varepsilon$ checked by virtual substitution along with $-\infty$ **(right)** Illustration of roots e and infinitesimal offsets $e - \varepsilon$ that could be checked along with $+\infty$ instead

The order of the interval endpoints that the proof of Theorem 6 uses in addition to $-\infty$ are illustrated in Fig. 3(left). Observe that exactly one representative point is placed in each of the regions of interest, $-\infty$, each of the roots r , and just infinitesimally after the roots at $e + \varepsilon$. Alternatively, Theorem 6 could be rephrased to work with ∞ , at each root e , and always before the roots at $e - \varepsilon$; see Fig. 3(right) and Exercise 3. The illustrations in Fig. 3 show the ordering situation for a higher-degree polynomial p even if Theorem 6 only makes use of the argument for $p = ax^2 + bx + c$ up to degree 2. Quantifier elimination procedures for higher degrees are still based on this fundamental principle, but require more algebraic computations.

Finally note that it is quite possible that the considered polynomial p does not single out the appropriate root e or off-root $e + \varepsilon$ that satisfies F to witness $\exists x F$. So none of the points illustrated in Fig. 3 will satisfy F , because only a point other than $e + \varepsilon$ in the open interval between two roots will work.

Note 10 (No rejection without mention). *The key argument underlying all quantifier elimination procedures in some way or another is that another part of F that is not satisfied for any of the points in Fig. 3 that p brings about would have to mention another polynomial q with different roots d or different off-roots $d + \varepsilon$ that will then enter the big disjunction in Theorem 6.*

Example 7. The example of nonnegative roots of quadratic polynomials from [Lecture 18 on Virtual Substitutions & Real Equations](#) used Theorem 1 to construct and justify the quantifier elimination equivalence:

$$\begin{aligned} \text{QE}(\exists x (ax^2 + bx + c = 0 \wedge x \geq 0)) \\ \equiv b^2 - 4ac \geq 0 \wedge (ba \leq 0 \wedge ac \geq 0 \vee a \geq 0 \wedge ac \leq 0 \vee a \leq 0 \wedge ac \leq 0) \end{aligned}$$

under the assumption $a \neq 0$. Specializing to a case shown in Fig. 2 gives:

$$\text{QE}(\exists x (x^2 - x + c = 0 \wedge x \geq 0)) \equiv (-1)^2 - 4c \geq 0 \wedge (c \geq 0 \vee c \leq 0) \equiv 1 - 4c \geq 0 \equiv c \leq \frac{1}{4}$$

By Theorem 6, the same square root expression substitution as in [Lecture 18 on Virtual Substitutions & Real Equations](#) will happen for the atomic formula $x^2 - x + c \leq 0$ except that the case of $-\infty$ will be added as well as the root 0 that results from considering the linear atomic formula $-x \geq 0$:

$$\begin{aligned} \text{QE}(\exists x (x^2 - x + c \leq 0 \wedge x \geq 0)) \equiv \\ \underbrace{(x^2 - x + c \leq 0 \wedge \dots)_{\hat{x}}^{-\infty}}_{\text{false}} \vee 1 - 4c \geq 0 \vee \underbrace{(x^2 - x + c \leq 0 \wedge x \geq 0)_{\hat{x}}^0}_{c \leq 0 \wedge 0 \geq 0} \equiv 1 - 4c \geq 0 \end{aligned}$$

Note that the additional disjunction $x \leq 0$ coming from the root 0 of $-x$ is in this case subsumed by the previous disjunct $1 - 4c \geq 0$. Hence, adding the roots of $-x$ did not modify the answer in this case. When adding a third conjunct $-x + 2 = 0$, this becomes critical:

$$\text{QE}(\exists x (x^2 - x + c \leq 0 \wedge x \geq 0 \wedge -x + 2 = 0))$$

Since the first two polynomials $x^2 - x + c$ and $-x$ are still the same, the same virtual substitutions will happen as before. Except that they fail on the new conjunct $-x + 2 = 0$, because the root 0 of the polynomial $-x$ from the second conjunct does not satisfy $-x + 2 = 0$ and because the virtual substitution of the roots $(-1 \pm \sqrt{1 - 4c})/2$ of the first polynomial $x^2 - x + c$ fail:

$$\begin{aligned} (-x + 2 = 0)_{\hat{x}}^{(-1 \pm \sqrt{1 - 4c})/2} \equiv ((1 + \mp 1\sqrt{1 - 4c})/2 + 2 = 0) \equiv ((3 + \mp 1\sqrt{1 - 4c})/2 = 0) \\ \equiv \mp 3 \leq 0 \wedge 3^2 - (\mp 1)^2(1 - 4c) = 0 \equiv -3 \leq 0 \wedge 3^2 - (-1)^2(1 - 4c) = 0 \equiv 8 - 4c = 0 \end{aligned}$$

The latter is only possible for $c = 2$, which is ruled out by the discriminant condition $1 - 4c \geq 0$ that precedes it. And, indeed, neither the roots of the quadratic polynomial

illustrated in Fig. 2 nor the roots of $-x$ nor $-\infty$ are the right points to consider to satisfy the last conjunct. Of course, the last conjunct expresses that by saying $-x + 2 = 0$ quite explicitly. Never mind that this is an equation for now. Either way, the atomic formula clearly exposes that $-x + 2$ is the polynomial that it cares about. So its roots might be of interest and will, indeed, be considered in the big disjunction of Theorem 6 as well. Since $-x + 2$ is a visibly linear polynomial, its solution is $x = -2 / -1 = 2$ which is even kind enough to be a standard real number so that literal substitution is sufficient and no virtual substitution is needed. Consequently, the substitution of this root $x = 2$ of the last conjunct into the full formula quickly yields:

$$(x^2 - x + c \leq 0 \wedge x \geq 0 \wedge -x + 2 = 0)_x^2 \equiv 2^2 - 2 + c \leq 0 \wedge 2 \geq 0 \wedge 0 = 0 \equiv 2 + c \leq 0$$

This provides an answer that the quadratic polynomial $x^2 - x + c$ itself could not foresee because it depends on the polynomial $-x + 2$ to even take this root into consideration. By Theorem 6, the overall result of quantifier elimination, thus, is the combination of the cases considered separately above:

$$\begin{aligned} & \text{QE}(\exists x (x^2 - x + c \leq 0 \wedge x \geq 0 \wedge -x + 2 = 0)) \\ & \equiv \underbrace{(x^2 - x + c \leq 0 \wedge \dots)_{\hat{x}}^{-\infty}}_{\text{false}} \\ & \vee 1 - 4c \geq 0 \wedge \underbrace{(\dots \wedge -x + 2 = 0)_{\hat{x}}^{(-1 \pm \sqrt{1-4c})/2}}_{8-4c=0} \\ & \vee -1 \neq 0 \wedge \underbrace{(x^2 - x + c \leq 0 \wedge x \geq 0)_x^0}_{c \leq 0 \wedge 0 \geq 0} \wedge \underbrace{(-x + 2 = 0)_x^0}_{2=0} \\ & \vee -1 \neq 0 \wedge \underbrace{(x^2 - x + c \leq 0 \wedge x \geq 0 \wedge -x + 2 = 0)_x^2}_{2+c \leq 0} \\ & \equiv 2 + c \leq 0 \equiv c \leq -2 \end{aligned}$$

In this particular case, observe that Theorem 1 using $-x + 2 = 0$ as the key formula would have been most efficient, because that would have gotten the answer right away without fruitless disjunctions. This illustrates that it pays off to pay attention with real arithmetic and always choose the computationally most parsimonious approach. But the example also illustrates that the same computation would happen if the third conjunct would have been $-x + 2 \leq 0$, in which case Theorem 1 would not have helped.

Optimizations are possible for virtual substitution [Wei97] if there is only one quadratic occurrence of x , and that occurrence is not in an equation. If that occurrence is an equation, Theorem 1 already showed what to do. If there is only one occurrence of a quadratic inequality, the following variation of Theorem 6 works, which uses exclusively linear fractions.

Note 11 ([Wei97]). Let $\left(Ax^2 + Bx + C \left\{ \begin{smallmatrix} \leq \\ < \\ \neq \end{smallmatrix} \right\} 0\right) \in F$ be the only quadratic occurrence of x . In that case, $\exists x F$ is equivalent over \mathbb{R} to the following quantifier-free formula:

$$\begin{aligned} & A = 0 \wedge B \neq 0 \wedge F_{\hat{x}}^{-C/B} \vee A \neq 0 \wedge F_{\hat{x}}^{-B/(2A)} \\ & \vee F_{\hat{x}}^{-\infty} \vee F_{\hat{x}}^{\infty} \\ & \vee \bigvee_{(0x^2+bx+c \left\{ \begin{smallmatrix} = \\ \leq \end{smallmatrix} \right\} 0) \in F} (b \neq 0 \wedge F_{\hat{x}}^{-c/b}) \\ & \vee \bigvee_{(0x^2+bx+c \left\{ \begin{smallmatrix} \neq \\ < \end{smallmatrix} \right\} 0) \in F} (b \neq 0 \wedge (F_{\hat{x}}^{-c/b+\varepsilon} \vee F_{\hat{x}}^{-c/b-\varepsilon})) \end{aligned}$$

The clou in this case is that the extremal values of $Ax^2 + Bx + C$ are at the roots of the derivative

$$(Ax^2 + Bx + C)' = 2AX + B \stackrel{!}{=} 0 \text{ i.e. } x = -\frac{B}{2A}$$

Since the only quadratic occurrence in Note 11 is not an equation, this extremal value is the only point of the quadratic polynomial that matters. In this case, $F_{\hat{x}}^{-B/(2A)}$ will substitute $-B/(2A)$ for x in the only quadratic polynomial as follows:

$$\left(Ax^2 + Bx + C \left\{ \begin{smallmatrix} \leq \\ < \\ \neq \end{smallmatrix} \right\} 0\right)_{\hat{x}}^{-B/(2A)} \equiv \left(A \frac{(-B)^2}{4A^2} + \frac{-B^2}{2A} + C \left\{ \begin{smallmatrix} \leq \\ < \\ \neq \end{smallmatrix} \right\} 0\right) \equiv \left(\frac{-B^2}{4A} + C \left\{ \begin{smallmatrix} \leq \\ < \\ \neq \end{smallmatrix} \right\} 0\right)$$

The formula resulting from Note 11 might be bigger than that of Theorem 6 but it does not increase the polynomial degrees.

Further optimizations are possible if some signs of a, b are known, because several cases in the quantifier-free expansion then become impossible and can be simplified to *true* or *false* immediately. This helps simplify the formula in Theorem 6, because one of the cases $a = 0$ versus $a \neq 0$ might drop. But it also reduces the number of disjuncts in $F_{\hat{x}}^{-\infty}$, see Example 2, and in the virtual substitutions of square roots (Lecture 18) and of infinitesimals (Sect. 4), which can lead to significant simplifications.

Theorem 6 also applies for polynomials of higher degrees in x if all those factor to polynomials of at most quadratic degree in x [Wei97]. Degree reduction is also possible by renaming based on the greatest common divisor of all powers of x that occur in F . If a quantified variable x occurs only with degrees that are multiples of an odd number d then virtual substitution can use $\exists x F(x^d) \equiv \exists y F(y)$. If x only occurs with degrees that are multiples of an even number d then $\exists x F(x^d) \equiv \exists y (y \geq 0 \wedge F(y))$. Finally, the cases in Theorem 6 with infinitesimals $+\varepsilon$ are only needed if x occurs in strict inequalities in F . The cases $F_{\hat{x}}^{(-b \pm \sqrt{d})/(2a)}$ are only needed if x occurs in equations or weak inequalities.

6. Summary

Virtual substitution is one technique for eliminating quantifiers in real arithmetic. It works for linear and quadratic constraints and can be extended to some cubic cases [Wei94]. Virtual substitution can be applied repeatedly from inside out to eliminate quantifiers. In each case, however, virtual substitution requires the eliminated variable to occur with small enough degrees only. Even if that was the case initially, it may stop to be the case after eliminating the innermost quantifier, because the degrees of the formulas resulting from virtual substitution may increase. In that case, degree optimizations and simplifications may sometimes work. If not, then other quantifier elimination techniques need to be used, which are based on semialgebraic geometry or model theory. Virtual substitution alone always works for mixed quadratic-linear formulas, i.e. those in which all quantified variables occur linearly except for one variable that occurs quadratically. In practice, however, many other cases turn out to work well with virtual substitution.

By inspecting Theorem 6 and its optimizations, we also observe that it is interesting to look at only closed sets or only open sets, corresponding to formulas with only \leq and $=$ or sets with only $<$ and \neq conditions, because half of the cases then drop out of the expansion in Theorem 6. Furthermore, if the formula $\exists x F$ only mentions strict inequalities $<$ and disequations \neq , then all virtual substitutions will involve infinitesimals or infinities. While both are conceptually more demanding than virtual substitutions with mere square root expressions, the advantage is that both infinitesimals and infinities rarely satisfy any equations (except when they are trivial because all coefficients are zero). In that case, most formulas simplify tremendously. That is an indication in the virtual substitution method for a more general phenomenon: existential arithmetic with strict inequalities or, dually, validity of universal arithmetic with weak inequalities, is computationally easier.

A. Semialgebraic Geometry

The geometric counterpart of polynomial equations or quantifier-free first-order formulas with polynomial equations are affine varieties. The geometric counterpart of first-order formulas of real arithmetic that may mention inequalities are called semialgebraic sets in real algebraic geometry [BCR98, BPR06]. By quantifier elimination, the sets definable with quantifiers is the same as the sets definable without quantifiers. Hence, the formulas of first-order real arithmetic exactly define semialgebraic sets:

Definition 8 (Semialgebraic Set). $S \subseteq \mathbb{R}^n$ is an *semialgebraic set* iff it is defined by a finite intersection of polynomial equations and inequalities or any finite union of such sets.

$$S = \bigcup_{i=1}^t \bigcap_{j=1}^s \{x \in \mathbb{R}^n : p(x) \sim 0\} \quad \text{where } \sim \in \{=, \geq, >\}$$

The geometric counterpart of the quantifier elimination result is that semialgebraic sets are closed under projection (the other closure properties are obvious in logic).

Theorem 9 (Tarski Seidenberg [Tar51, Sei54]). *Semialgebraic sets are closed under finite unions, finite intersections, complements and projection to linear subspaces.*

The semialgebraic sets corresponding to a number of interesting systems of polynomial inequalities are illustrated in Fig. 4.

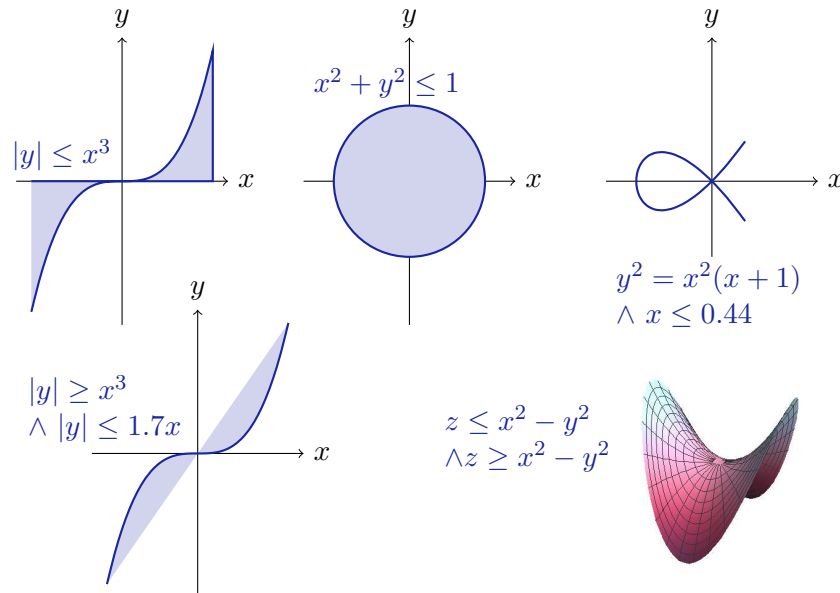


Figure 4: Systems of polynomial inequalities describe semialgebraic sets

Exercises

Exercise 1. Consider

$$\exists x (ax^2 + bx + c \leq 0 \wedge F) \quad (14)$$

The virtual substitution of the roots of $ax^2 + bx + c = 0$ according to Sect. 2 as well as of $-\infty$ according to Sect. 3 will lead to

$$F_{\hat{x}}^{-\infty} \vee a = 0 \wedge b \neq 0 \wedge F_{\hat{x}}^{-c/b} \vee a \neq 0 \wedge b^2 - 4ac \geq 0 \wedge (F_{\hat{x}}^{(-b+\sqrt{b^2-4ac})/(2a)} \vee F_{\hat{x}}^{(-b-\sqrt{b^2-4ac})/(2a)})$$

But when F is $-ax^2 + bx + e < 0$, then none of those cases necessarily works. Does that mean the result of virtual substitution is not equivalent to (14)? Where is the catch in this argument?

Exercise 2. Perform quantifier elimination by virtual substitution to compute

$$\text{QE}(\exists x (x^2 - x + c \leq 0 \wedge x \geq 0 \wedge -x + 2 \leq 0))$$

Exercise 3. Develop and prove a virtual substitution formula for quadratic polynomials analog to Theorem 6 that uses the points illustrated in Fig. 3(right) instead of Fig. 3(left).

References

- [BCR98] Jacek Bochnak, Michel Coste, and Marie-Francoise Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer, 1998.
- [BD07] Christopher W. Brown and James H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In Dongming Wang, editor, *ISSAC*, pages 54–60. ACM, 2007.
- [BPR06] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2nd edition, 2006. doi:10.1007/3-540-33099-2.
- [dMP13] Leonardo Mendonça de Moura and Grant Olney Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *LNCS*, pages 178–192. Springer, 2013.
- [Pas11] Grant Olney Passmore. *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*. PhD thesis, School of Informatics, University of Edinburgh, 2011.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.
- [Pla12] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. doi:10.1109/LICS.2012.13.
- [PQR09] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 485–501. Springer, 2009. doi:10.1007/978-3-642-02959-2_35.
- [Sei54] Abraham Seidenberg. A new decision method for elementary algebra. *Annals of Mathematics*, 60:365–374, 1954.
- [Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.
- [Wei94] Volker Weispfenning. Quantifier elimination for real algebra — the cubic case. In *ISSAC*, pages 258–263, 1994.
- [Wei97] Volker Weispfenning. Quantifier elimination for real algebra — the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997.

Lecture Notes on Hybrid Systems & Games

André Platzer

Carnegie Mellon University
Lecture 20

1 Introduction

Hybrid systems have so far served us well throughout this course as a model for cyber-physical systems [Pla08, Pla10b]. Most definitely, hybrid systems can also serve as models for other systems that are not cyber-physical per se, i.e. they are not built as a combination of cyber and computing capabilities with physical capabilities. Some biological systems can be understood as hybrid systems, because they combine discrete and continuous dynamics. Or physical processes in which things happen at very different speeds, so where there is a slow process about which a continuous understanding is critical as well as a very fast process in which a discrete abstraction might be sufficient. Neither of those examples are particularly cyber-physical. Yet, nevertheless, they can have natural models as hybrid systems, because their fundamental characteristic is the interaction of discrete and continuous dynamics, which is exactly what hybrid systems are good for. Hence, despite their good match, not *all* hybrid systems are cyber-physical systems.

One important point of today's lecture is that the converse is not true either. Not all cyber-physical systems are hybrid systems. The reason for that is *not* that cyber-physical systems lack discrete and continuous dynamics, but, rather, that they involve also additional dynamical aspects. It is a common phenomenon in cyber-physical systems that they involve several dynamical aspects, which is why they are best understood as *multi-dynamical systems*, i.e. systems with multiple dynamical features [Pla12c, Pla12b, Pla11, Pla13].

In a certain sense, applications often have a +1 effect on dynamical aspects. Your analysis might start out focusing on some number of dynamical aspects just to observe during the elaboration of the analysis that there is a part of the system for which one more dynamical aspect is relevant than was originally anticipated. The bouncing ball

is an example where a preliminary analysis might first ascribe an entirely continuous dynamics to it, just to find out after a while that the singularity of bouncing back from the ground would be better understood by discrete dynamics. So whenever you are analyzing a system, be prepared to find one more dynamical aspect around the corner. That is yet another reason why it is useful to have flexible and general analysis techniques grounded in logic that still work even after a new dynamical aspect has been found.

Of course, it is not going to be feasible to understand all multi-dynamical system aspects at once in today's lecture. But today's lecture is going to introduce one very fundamental dynamical aspect: *adversarial dynamics* [Pla13, Pla14]. Adversarial dynamics comes from multiple players that, in the context of CPS, interact on a hybrid system and are allowed to make their respective choices arbitrarily, just in pursuit of their goals. The combination of discrete, continuous, and adversarial dynamics leads to *hybrid games*. Unlike hybrid systems, hybrid games allow choices in the system dynamics to be resolved adversarially by different players with different objectives.

Hybrid games are necessary in situations where multiple agents actively compete. The canonical situation of a hybrid game would, thus, be RoboCup, where two teams of robots play robot soccer, moving around physically in space, controlled according to discrete computer decisions, and in active competition for scoring goals in opposite directions on the field. The robots in a RoboCup match just can't agree on the direction into which they try to get the ball rolling. It turns out, however, that hybrid games also come up for reasons of analytic competition, that is, where possible competition is assumed only for the sake of a worst-case analysis.

Consider lab 4, the static and dynamic obstacles lab, for example, where your robot is interacting with a roguebot. You are in control of the robot, but somebody else is controlling the roguebot. Your objective is to control your robot so that it will not run into the roguebot no matter what. That means you need to find *some* way of playing your control choices for your robot so that it makes progress but will be safe *for all* possible control choices that the roguebot might follow. After all you do not exactly know how the other roguebot is implemented and how it will react to your control decisions. That makes your robot play a hybrid game with the roguebot in which your robot is trying to safely avoid collisions. The roguebot might behave sanely and tries to stay safe as well. But the roguebot's objectives could differ from yours, because its objective is not to get you to your goal. The roguebot rather wants to get to its own goal instead, which might cause unsafe interferences whenever the roguebot takes an action in pursuit of its goal that is not in your robot's interest. If your robot causes a collision, because it chose an action that was incompatible with the roguebot's action, your robot would certainly be faulty and sent back to the design table.

Alas, when you try to understand how you need to control your robot to stay safe, it can be instructive to think about what the worst-case action of a roguebot might be to make life difficult for you. And when your friendly course instructors try to demonstrate for you under which circumstance a simulation of your robot controller exhibits a faulty behavior, so that you can learn from the cases where your control does not work, they might actually be playing a hybrid game with you. If your robot wins

and stays safe, that is an indication of a strong robot design. But if your course TAs win and show an unsafe trace, you still win even if you lose this particular simulation, because you learn more about the corner cases in your robot control design than when staring at simulation movies where everything is just fair-weather control.

If you think carefully again about lab 2, where your robot was put on a highway and had to find some way of being controlled to stay safe for all possible choices of the robot in front of it, then you will find that a hybrid game interpretation might be in order for that lab as well.

These lecture notes are based on [Pla13, Pla14], where more information can be found on logic and hybrid games. The most important learning goals of this lecture are:

Modeling and Control: We identify an important additional dynamical aspect, the aspect of *adversarial dynamics*, which adds an adversarial way of resolving the choices in the system dynamics. This dynamical aspect is important for understanding the core principles behind CPS, because multiple agents with possibly conflicting actions are featured frequently in CPS applications. It is helpful to learn under which circumstance adversarial dynamics is important for understanding a CPS and when it can be neglected without loss. CPS in which all choices are resolved against you or all choices are resolved for you can already be described and analyzed in differential dynamic logic. Adversarial dynamics is interesting in mixed cases, where some choices fall in your favor and others turn out against you. Another important goal of this lecture is how to develop models and controls of CPS with adversarial dynamics corresponding to multiple agents.

Computational Thinking: This lecture follows fundamental principles from computational thinking to capture the new phenomenon of adversarial dynamics in CPS models. We leverage core ideas from programming languages by extending syntax and semantics of program models and specification and verification logics with the complementary operator of duality to incorporate adversariality in a modular way into the realm of hybrid systems models. This leads to a compositional model of hybrid games with compositional operators. Modularity makes it possible to generalize our rigorous reasoning principles for CPS to hybrid games while simultaneously taming their complexity. This lecture introduces *differential game logic* dGL [Pla13, Pla14] extending by adversarial dynamics the familiar differential dynamic logic, which has been used as the specification and verification language for CPS in the other parts of this course. Computer science ultimately is about analysis like worst-case analysis or expected-case analysis or correctness analysis. Hybrid games enable analysis of CPS at a more fine-grained level in between worst-case analysis and best-case analysis. In the dL formula $[\alpha]\phi$ all choices are resolved against us in the sense that $[\alpha]\phi$ is only true if ϕ holds after all runs of α . In the dL formula $\langle\alpha\rangle\phi$ all choices are resolved in favor in the sense that $\langle\alpha\rangle\phi$ is true if ϕ holds after at least one run of α . Hybrid games can be used to attribute some but not all of the choices in a system to an opponent while leaving the others to be resolved favorably. Finally, this lecture provides a perspective on advanced models of computation with alternating choices.

CPS Skills: We add a new dimension into our understanding of the semantics of a CPS model: the adversarial dimension corresponding to how a system changes state over time as multiple agents react to each other. This understanding is crucial for developing an intuition for the operational effects of multi-agent CPS. The presence of adversarial dynamics will cause us to reconsider the semantics of CPS models to incorporate the effects of multiple agents and their mutual reactions. This generalization, while crucial for understanding adversarial dynamics in CPS, also shines a helpful complementary light on the semantics of hybrid systems without adversariality by causing us to reflect on choices.

2 Choices & Nondeterminism

Note 1 (Choices in hybrid systems). *Hybrid systems involve choices. They manifest evidently in hybrid programs as nondeterministic choices $\alpha \cup \beta$ whether to run HP α or HP β , in nondeterministic repetitions α^* where the choice is how often to repeat α , and in differential equations $x' = \theta \ \& \ H$ where the choice is how long to follow that differential equation. All those choices, however, have still been resolved in one way, i.e. by the same entity or player.*

In which way the various choices are resolved depends on the context. In the box modality $[\alpha]$ of differential dynamic logic [Pla08, Pla10b, Pla12c], the choices are resolved in *all possible ways* so that the modal formula $[\alpha]\phi$ expresses that formula ϕ holds for all ways how the choices in HP α could resolve. In the diamond modality $\langle\alpha\rangle$, instead, the choices are resolved in *some way* so that formula $\langle\alpha\rangle\phi$ expresses that formula ϕ holds for one way of resolving the choices in HP α . That is how $[\alpha]\phi$ expresses that ϕ holds necessarily after α while $\langle\alpha\rangle\phi$ expresses that ϕ is possible after α .

In particular, choices in α help $\langle\alpha\rangle\phi$, because what this formula calls for is *some* way of making ϕ happen after α . If α has many possible behaviors, this is easier to satisfy. Choices in α hurt $[\alpha]\phi$, however, because this formula requires ϕ to hold for all those choices. The more choices there are, the more difficult it is to make sure that ϕ holds after every single combination of those choices.

Note 2. *In differential dynamic logic, choices in α either help uniformly (when they occur in $\langle\alpha\rangle\phi$) or make things more difficult uniformly (when they occur in $[\alpha]\phi$).*

That is why these various forms of choices in hybrid programs have been called *nondeterministic*. They are “unbiased”. All possible resolutions of the choices in α could happen nondeterministically when running α . Which possibilities we care about (all or some) just depends on the modal formula around it.

3 Control & Dual Control

Another way of looking at the choices that are to be resolved during the runs of a hybrid program α is that they can be resolved by one player. Let's call her *Angel*, because she helps us so much in making $\langle\alpha\rangle\phi$ formulas true. Whenever a choice is about to happen (by running the program statements $\alpha \cup \beta$, α^* , or $x' = \theta \ \& \ H$), Angel is called upon to see how the choice is supposed to be resolved this time.

From that perspective, it sounds easy enough to add a second player. Let's call him *Demon* as Angel's perpetual opponent.¹ Only so far, Demon will probably be rather bored after a while, when he realizes that he never actually gets to decide anything, because Angel has all the fun in choosing how the hybrid program world unfolds and Demon just sits around idly. So to keep Demon entertained, we need to introduce some choices that fall under Demon's control.

One thing, we could do to keep Demon interested in playing along is to add a pair of shiny new controls especially for him. They might be called $\alpha \cap \beta$ for Demon's choice between α or β as well as α^\times for repetition of α under Demon's control as well as an operation, say $x' = \theta \ \& \ H^d$, for continuous evolution under Demon's reign. But that would cause a lot of attention to Demon's control, which might make him feel overly majestic. Let's not do that, because we don't want Demon to get any ideas.

Instead, we will find it sufficient to add just a single operator to hybrid programs: the dual operator d . What α^d does is to give all control that Angel had in α to Demon, and, vice versa, all control that Demon had in α to Angel. The dual operator, thus, is a little bit like what happens when you turn a chessboard around by 180° in the middle of the game. Whoever played the choices of player White before suddenly controls Black, and whoever played Black now controls White. With just this single duality operator it turns out that Demon still gets his own set of controls ($\alpha \cap \beta$, α^\times , $x' = \theta \ \& \ H^d$) by suitably nesting the operators, but we did not have to give him those controls specifically. Yet, now those extra controls are not special but simply an aspect of a more fundamental principle: duality.

4 Hybrid Games

Differential game logic (dGL) is a logic for studying properties of hybrid games. The idea is to describe the game form, i.e. rules, dynamics, and choices of the particular hybrid game of interest, using a program notation and to then study its properties by proving the validity of logical formulas that refer to the existence of winning strategies for objectives of those hybrid games. Even though hybrid game forms only describe the game *form* with its dynamics and rules and choices, not the actual objective, they are still simply called hybrid games. The objective for a hybrid game is defined in the modal logical formula that refers to that hybrid game form.

¹The responsibilities of such ontologically loaded names are easier to remember than those of neutral player names I and II.

Definition 1 (Hybrid games). The *hybrid games of differential game logic* dGL are defined by the following grammar (α, β are hybrid games, x a vector of variables, θ a vector of (polynomial) terms of the same dimension, H is a dGL formula or just a formula of first-order real arithmetic):

$$\alpha, \beta ::= x := \theta \mid x' = \theta \ \& \ H \mid ?H \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid \alpha^d$$

The only syntactical difference of hybrid games compared to hybrid programs for hybrid systems from [Lecture 3 on Choice & Control](#) is that, unlike hybrid programs, hybrid games allow the dual operator α^d . This minor syntactic change also requires us to reinterpret the meaning of the other operators in a much more flexible way to make sense of the presence of subgames within the games in which the players already interact. The basic principle is that whenever there used to be nondeterminism in the hybrid program semantics, there will now be a choice of Angel in the hybrid game semantics. But don't be fooled. The parts of such a hybrid game may still be hybrid games, in which players interact, rather than just a single system running. So *all* operators of hybrid games still need a careful understanding as games, not just \cdot^d , because all operators can be applied to subgames that mention \cdot^d or be part of a context that mentions \cdot^d .

The *atomic games* of dGL are assignments, continuous evolutions, and tests. In the *deterministic assignment game* (or discrete assignment game) $x := \theta$, the value of variable x changes instantly and deterministically to that of θ by a discrete jump without any choices to resolve. In the *continuous evolution game* (or continuous game) $x' = \theta \ \& \ H$, the system follows the differential equation $x' = \theta$ where the duration is Angel's choice, but Angel is not allowed to choose a duration that would, at any time, take the state outside the region where formula H holds. In particular, Angel is deadlocked and loses immediately if H does not hold in the current state, because she cannot even evolve for duration 0 then without being outside H .² The *test game* or *challenge* $?H$ has no effect on the state, except that Angel loses the game immediately if dGL formula H does not hold in the current state, because she failed the test she was supposed to pass. The test game $?H$ challenges Angel and she loses immediately if she fails. Angel does not win just because she passed the challenge $?H$, but at least the game continues. So passing challenges is a necessary condition to win games. Failing challenges, instead, immediately makes Angel lose.

The *compound games* of dGL are sequential, choice, repetition, and duals. The *sequential game* $\alpha; \beta$ is the hybrid game that first plays hybrid game α and, when hybrid game α terminates without a player having won already (so no challenge in α failed), continues by playing game β . When playing the *choice game* $\alpha \cup \beta$, Angel chooses whether to play hybrid game α or play hybrid game β . Like all the other choices, this choice is

² Note that the most common case for H is a formula of first-order real arithmetic, but any dGL formula will work. Evolution domain constraints H turn out to be unnecessary, because they can be defined using hybrid games [Pla13, Pla14]. In the ordinary differential equation $x' = \theta$, the term x' denotes the time-derivative of x and θ is a polynomial term that is allowed to mention x and other variables. More general forms of differential equations are possible [Pla10a, Pla10b], but will not be considered explicitly.

dynamic, i.e. every time $\alpha \cup \beta$ is played, Angel gets to choose again whether she wants to play α or β this time. The *repeated game* α^* plays hybrid game α repeatedly and Angel chooses, after each play of α that terminates without a player having won already, whether to play the game again or not, albeit she cannot choose to play indefinitely but has to stop repeating ultimately. Angel is also allowed to stop α^* right away after zero iterations of α . Most importantly, the *dual game* α^d is the same as playing the hybrid game α with the roles of the players swapped. That is Demon decides all choices in α^d that Angel has in α , and Angel decides all choices in α^d that Demon has in α . Players who are supposed to move but deadlock lose. Thus, while the test game $?H$ causes Angel to lose if formula H does not hold, the *dual test game* (or *dual challenge*) $(?H)^d$ instead causes Demon to lose if H does not hold.

For example, if α describes the game of chess, then α^d is chess where the players switch sides. If α , instead, describes the hybrid game corresponding to your lab 5 robot model where you are controlling a robot and your course instructors are controlling the roguebot, then α^d describes the dual game where you take control of the roguebot and the course instructors are stuck with your robot controls.

The dual operator d is the only syntactic difference of dGL for hybrid games compared to dL for hybrid systems [Pla08, Pla12a], but a fundamental one [Pla13, Pla14], because it is the only operator where control passes from Angel to Demon or back. Without d all choices are resolved uniformly by Angel without interaction. The presence of d requires a thorough semantic generalization throughout the logic to cope with such flexibility.

5 Differential Game Logic

Hybrid games describe how the world can unfold when Angel and Demon interact according to their respective control choices. They explain the rules of the game how Angel and Demon interact, but not who wins the game, nor what the respective objectives of the players are.³ The winning conditions are specified by logical formulas of differential game logic. Modal formulas $\langle\alpha\rangle\phi$ and $[\alpha]\phi$ refer to hybrid games and the existence of winning strategies for Angel and Demon, respectively, in a hybrid game α with a winning condition specified by a logical formula ϕ .

Definition 2 (dGL formulas). The *formulas of differential game logic* dGL are defined by the following grammar (ϕ, ψ are dGL formulas, p is a predicate symbol of arity k , θ_i are (polynomial) terms, x a variable, and α is a hybrid game):

$$\phi, \psi ::= p(\theta_1, \dots, \theta_k) \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \exists x \phi \mid \langle\alpha\rangle\phi \mid [\alpha]\phi$$

Other operators $>, =, \leq, <, \vee, \rightarrow, \leftrightarrow, \forall x$ can be defined as usual, e.g., $\forall x \phi \equiv \neg \exists x \neg \phi$. The modal formula $\langle\alpha\rangle\phi$ expresses that Angel has a winning strategy to achieve ϕ in hybrid game α , i.e. Angel has a strategy to reach any of the states satisfying dGL formula

³Except that players lose if they disobey the rules of the game by failing their respective challenges.

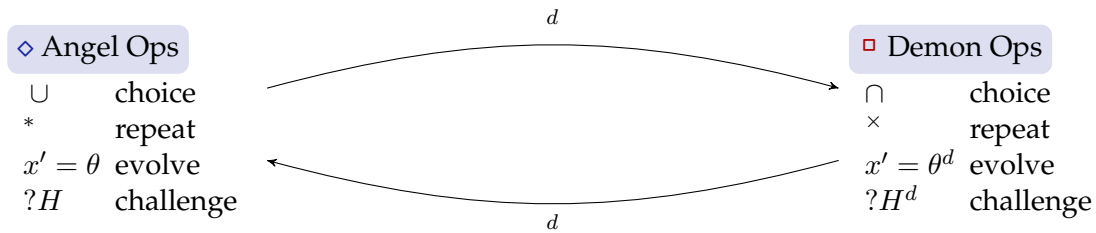
ϕ when playing hybrid game α , no matter what strategy Demon chooses. The modal formula $[\alpha]\phi$ expresses that Demon has a winning strategy to achieve ϕ in hybrid game α , i.e. a strategy to reach any of the states satisfying ϕ , no matter what strategy Angel chooses.⁴ Note that the same game is played in $[\alpha]\phi$ as in $\langle\alpha\rangle\phi$ with the same choices resolved by the same players. The difference between both dG \mathcal{L} formulas is the player whose winning strategy they refer to. Both use the set of states where dG \mathcal{L} formula ϕ is true as the winning states for that player. The winning condition is defined by the modal formula, α only defines the hybrid game form, not when the game is won, which is what ϕ does. Hybrid game α defines the rules of the game, including conditions on state variables that, if violated, cause the present player to lose for violation of the rules of the game. The dG \mathcal{L} formulas $\langle\alpha\rangle\phi$ and $[\alpha]\neg\phi$ consider complementary winning conditions for Angel and Demon.

6 Demon's Controls

Angel has full control over all choices in each of the operators of hybrid games *except* when the operator d comes into play. All choices within the scope of (an odd number of) d belong to Demon, because d makes the players switch sides. Demon's controls, i.e. direct controls for Demon, can be defined using the duality operator d on Angel's controls.

Demonic choice between hybrid game α and β is $\alpha \cap \beta$, defined by $(\alpha^d \cup \beta^d)^d$, in which either the hybrid game α or the hybrid game β is played, by Demon's choice. The choice for the \cup operator belongs to Angel, yet since it is nested within d , that choice goes to Demon, except that the d operators around α and β restore the original ownership of controls. *Demonic repetition* of hybrid game α is α^\times , defined by $((\alpha^d)^*)^d$, in which α is repeated as often as Demon chooses to. Again, the choice in the $*$ operator belongs to Angel, but in a d context goes to Demon, while the choices in the α, β subgames underneath stay as they were originally thanks to the additional d operators. In α^\times , Demon chooses after each play of α whether to repeat the game, but cannot play indefinitely so he has to stop repeating ultimately. The *dual differential equation* $(x' = \theta \ \& \ H)^d$ follows the same dynamics as $x' = \theta \ \& \ H$ except that Demon chooses the duration, so he cannot choose a duration during which H stops to hold at any time. Hence he loses when H does not hold in the current state. Dual assignment $(x := \theta)^d$ is equivalent to $x := \theta$, because it never involved any choices to begin with. Angel's control operators and Demon's control operators correspond to each other by duality:

⁴It is easy to remember which modal operator is which. The formula $\langle\alpha\rangle\phi$ clearly refers to Angel's winning strategies because the diamond operator $\langle\cdot\rangle$ has wings.



7 Operational Game Semantics (informally)

Treatment of a proper semantics for differential game logic will be deferred to the next lecture. A graphical illustration of the choices when playing hybrid games is depicted in Fig. 1. The nodes where Angel gets to decide are shown as diamonds \diamond , the nodes where Demon decides are shown as boxes \square . Circle nodes are shown when it depends on the remaining hybrid game which player it is that gets to decide. Dashed edges indicate Angel's actions, solid edges would indicate Demon's actions, while zigzag edges indicate that a hybrid game is played and the respective players move as specified by that game. The actions are the choice of time for $x' = \theta \ \& \ H$, the choice of playing the left or the right game for a choice game $\alpha \cup \beta$, and the choice of whether to stop or repeat in a repeated game α^* . This principle can be made rigorous in an operational game semantics [Pla13], which conveys the intuition of interactive game play for hybrid games, relates to game theory and descriptive set theory, but is also beyond the scope of these lecture notes. Observe how all choices involve at most two possibilities except differential equations, which have an uncountably infinite branching factor, one option for each duration $r \in \mathbb{R}$.

As an example, consider the *filibuster formula*:

$$\langle (x := 0 \cap x := 1)^* \rangle x = 0 \quad (1)$$

It is Angel's choice whether to repeat (*), but every time Angel repeats, it is Demon's choice (\cap) whether to play $x := 0$ or $x := 1$. What is the truth-value of the dGL formula (1)?

The game in this formula never deadlocks, because every player always has a remaining move (here even two). But it may appear as if the game had perpetual checks, because no strategy helps either player win the game; see Fig. 2. How could that happen and what can be done about it?

Before you read on, see if you can find the answer for yourself.

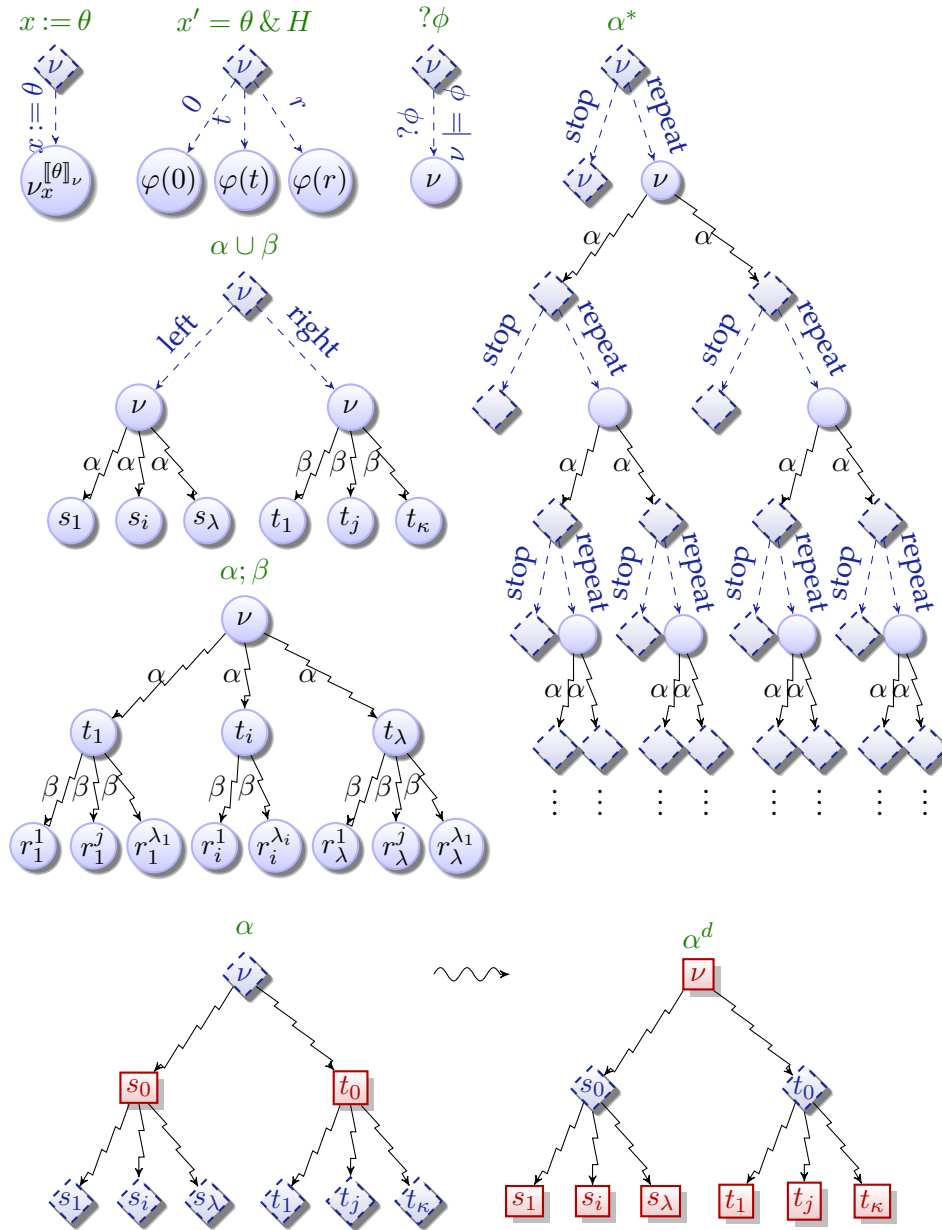


Figure 1: Operational game semantics for hybrid games of dGL

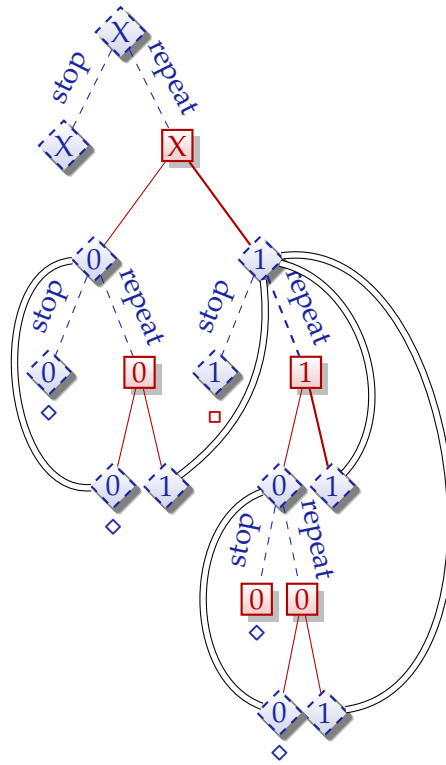


Figure 2: The filibuster game formula $\langle (x := 0 \cap x := 1)^* \rangle x = 0$ looks like it might be non-determined and not have a truth-value (unless $x = 0$ initially) when the strategies follow the thick actions. Angel's action choices are illustrated by dashed edges from dashed diamonds, Demon's action choices by solid edges from solid squares, and double lines indicate identical states with the same continuous state and a subgame of the same structure of subsequent choices. States where Angel wins are marked \diamond and states where Demon wins by \square .

The mystery of the filibuster game can be solved when we remember that the game still ultimately ought to stop in order to be able to inspect who won the game. Angel is in charge of the $*$ repetition and she can decide whether to stop or repeat. The filibuster game has no tests, so the winner only depends on the final state of the game, because both players are allowed to play arbitrarily without having to pass tests in between. Angel wins a game play if $x = 0$ holds in the final state and Demon wins if $x \neq 0$ holds in the final state. What do the strategies indicated in Fig. 2 do? They postpone the end of the game forever, hence there would never be a final state in which it could be evaluated who won. That is, indeed, not a way for anybody to win anything. Yet, Angel was in charge of the repetition $*$, so it is really her fault if the game never comes to a stop to evaluate who won, because she has to call it quits at some point. Consequently, the semantics of hybrid games requires players to repeat as often as they want but they cannot repeat indefinitely. This will be apparent in the actual semantics of hybrid games, which is defined as a denotational semantics corresponding to winning regions. Thus, (1) is false unless $x = 0$ already holds initially.

The same phenomenon happens in

$$\langle (x := 0; x' = 1^d)^* \rangle x = 0 \quad (2)$$

in which both players can let the other one win. Demon can let Angel win by choosing to evolve for duration 0. And Angel can let Demon win by choosing to stop even if $x \neq 0$. Only because Angel will ultimately have to stop repeating does the formula in (2) have a truth-value and the formula is false unless $x = 0$ already holds initially.

It is of similar importance that the players cannot decide to follow a differential equation forever (duration ∞), because that would make

$$\langle (x' = 1^d; x := 0)^* \rangle x = 0 \quad (3)$$

non-determined. If players were allowed to evolve along a differential equation forever (duration ∞), then Demon would have an incentive to evolve along $x' = 1^d$ forever in the continuous filibuster (3), because as soon as he stops, Angel would win because of the subsequent $X := 0$. But Angel cannot win without Demon stopping. Since Demon can evolve along $x' = 1^d$ for any finite amount of time he wants, he will ultimately have to stop so that Angel wins and (3) is valid.

8 Summary

This lecture saw the introduction of differential game logic, which extends the familiar differential dynamic logic with capabilities of modeling and understanding hybrid games. Hybrid games combine discrete dynamics, continuous dynamics, and adversarial dynamics. Compared to hybrid systems, the new dynamical aspect of adversarial dynamics is captured entirely by the duality operator d . Without it, hybrid games are single-player hybrid games, which are equivalent to hybrid systems.

After this lecture showed an informal and intuitive discussion of the actions that hybrid games allow, the next lecture gives a proper semantics to differential game logic and their hybrid games.

Exercises

Exercise 1. Single player hybrid games, i.e. d -free hybrid games, are just hybrid programs. For each of the following formulas, convince yourself that it has the same meaning, whether you understand it as a differential dynamic logic formula with a hybrid systems or as a differential game logic formula with a hybrid game (that happens to have only a single player):

$$\begin{aligned}
 &\langle x := 0 \cup x := 1 \rangle x = 0 \\
 &[x := 0 \cup x := 1] x = 0 \\
 &\langle (x := 0 \cup x := 1); ?x = 1 \rangle x = 0 \\
 &[(x := 0 \cup x := 1); ?x = 1] x = 0 \\
 &\langle (x := 0 \cup x := 1); ?x = 0 \rangle x = 0 \\
 &[(x := 0 \cup x := 1); ?x = 0] x = 0 \\
 &\langle (x := 0 \cup x := 1)^* \rangle x = 0 \\
 &[(x := 0 \cup x := 1)^*] x = 0 \\
 &\langle (x := 0 \cup x := x + 1)^* \rangle x = 0 \\
 &[(x := 0 \cup x := x + 1)^*] x = 0
 \end{aligned}$$

Exercise 2. Consider the following **dGL** formulas and identify under which circumstance they are true?

$$\langle (x := x + 1; (x' = x^2)^d \cup x := x - 1)^* \rangle (0 \leq x < 1)$$

$$\langle (x := x + 1; (x' = x^2)^d \cup (x := x - 1 \cap x := x - 2))^* \rangle (0 \leq x < 1)$$

Exercise 3. The following **dGL** formula characterizes a one-dimensional game of chase of a robot at position x and a robot at position y , each with instant control of the velocity v among $a, -a, 0$ for x (Angel's choice) and velocity w among $b, -b, 0$ for y (Demon's subsequent choice). The game repeats any number of control rounds following Angel's choice (*). Angel is trying for her robot x to be close to Demon's robot y . Under which circumstance is the formula true?

$$\begin{aligned}
 &\langle ((v := a \cup v := -a \cup v := 0); \\
 &\quad (w := b \cap w := -b \cap w := 0); \\
 &\quad x' = v, y' = w)^* \rangle (x - y)^2 \leq 1
 \end{aligned}$$

Exercise 4 ()*. The following **dGL** formula characterizes a two-dimensional game of chase of a robot at position (x_1, x_2) facing in direction (d_1, d_2) and a robot at position (y_1, y_2) facing in direction (e_1, e_2) . Angel has direct control over the angular velocity ω among $1, -1, 0$ for robot (x_1, x_2) and, subsequently, Demon has direct control over the angular velocity ϱ among $1, -1, 0$ for robot (y_1, y_2) . The game repeats any number of

control rounds following Angel's choice (*). Angel is trying for her robot to be close to Demon's robot. Is the following dGL formula valid? Can you identify some circumstances under which it is true? Or some circumstances under which it is false? How does this formula relate to lab 4?

$$\left\langle \left((\omega := 1 \cup \omega := -1 \cup \omega := 0); \right. \right. \\
(\varrho := 1 \cap \varrho := -1 \cap \varrho := 0); \\
\left. \left. (x'_1 = d_1, x'_2 = d_2, d'_1 = -\omega d_2, d'_2 = \omega d_1, y'_1 = e_1, y'_2 = e_2, e'_1 = -\varrho e_2, e'_2 = \varrho e_1)^d \right)^* \right\rangle \\
(x_1 - y_1)^2 + (x_2 - y_2)^2 \leq 1$$

References

- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. doi:10.1093/logcom/exn070.
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.
- [Pla11] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 431–445. Springer, 2011. doi:10.1007/978-3-642-22438-6_34.
- [Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:10.1109/LICS.2012.64.
- [Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:1205.4788.
- [Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:10.1109/LICS.2012.13.
- [Pla13] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.
- [Pla14] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014. arXiv:1408.1980.

Lecture Notes on Winning Strategies & Regions

[André Platzer](#)

Carnegie Mellon University
Lecture 21

1 Introduction

This lecture continues the study of hybrid games and their logic, differential game logic [[Pla13](#), [Pla14](#)], that [Lecture 20 on Hybrid Systems & Games](#) started. [Lecture 20](#) saw the introduction of differential game logic with a focus on identifying and highlighting the new dynamical aspect of adversarial dynamics. The meaning of hybrid games in differential game logic had been left informal, based on the intuition one relates to interactive gameplay and decisions in trees. While it is possible to turn such a tree-type semantics into an operational semantics for hybrid games [[Pla14](#)], the resulting development is technically rather involved. Even if such an operational semantics is interesting and touches on interesting concepts from descriptive set theory, it is unnecessarily complicated compared.

This lecture will, thus, be devoted to developing a much simpler yet rigorous semantics, a denotational semantics of hybrid games. [Lecture 20](#) already highlighted subtleties how never-ending game play ruins determinacy, simply because there never is a state in which the winner would be declared. Especially the aspect of repetition and its interplay with differential equations will need careful attention. The denotational semantics will make this subtle aspect crystal clear.

These lecture notes are based on [[Pla13](#), [Pla14](#)], where more information can be found on logic and hybrid games. The most important learning goals of this lecture are:

Modeling and Control: We further our understanding of the core principles behind CPS for the adversarial dynamics resulting from multiple agents with possibly conflicting actions that occur in many CPS applications. This time, we devote attention to the nuances of their semantics.

Computational Thinking: This lecture follows fundamental principles from computational thinking to capture the semantics of the new phenomenon of adversarial

dynamics in CPS models. We leverage core ideas from programming languages by extending syntax and semantics of program models and specification and verification logics with the complementary operator of duality to incorporate adversariality in a modular way into the realm of hybrid systems models. This leads to a compositional model of hybrid games with compositional operators. Modularity makes it possible to generalize our rigorous reasoning principles for CPS to hybrid games while simultaneously taming their complexity. This lecture introduces the semantics of *differential game logic* dGL [Pla13, Pla14], which adds adversarial dynamics to differential dynamic logic, which has been used as the specification and verification language for CPS in the other parts of this course. This lecture provides a perspective on advanced models of computation with alternating choices. The lecture will also encourage us to reflect on the relationship of denotational and operational semantics.

CPS Skills: This lecture focuses on developing and understanding the semantics of CPS models with adversarial dynamics corresponding to how a system changes state over time as multiple agents react to each other. This understanding is crucial for developing an intuition for the operational effects of multi-agent CPS. The presence of adversarial dynamics will cause us to reconsider the semantics of CPS models to incorporate the effects of multiple agents and their mutual reactions. This generalization, while crucial for understanding adversarial dynamics in CPS, also shines a helpful complementary light on the semantics of hybrid systems without adversariality by causing us to reflect on choices. The semantics of hybrid games properly generalizes the semantics of hybrid systems from earlier lectures.

2 Semantics

What is the most elegant way of defining a semantics for differential game logic? How could a semantics be defined at all? First of all, the dGL formulas ϕ that are used in the postconditions of dGL modal formulas $\langle \alpha \rangle \phi$ and $[\alpha] \phi$ define the winning conditions for the hybrid game α . Thus, when playing the hybrid game α , we need to know the set of states in which the winning condition ϕ is satisfied. That set of states in which ϕ is true is denoted $\llbracket \phi \rrbracket^I$, which defines the semantics of ϕ . The I in that notation is a reminder that the semantics depends on the interpretation of predicate symbols as defined in interpretation I . Thus, when we used to write $\nu \models \phi$ to indicate that dL formula ϕ is true in state ν , we will now write $\nu \in \llbracket \phi \rrbracket^I$, instead, to say that state ν is among the set of states in which ϕ is true. Working with the set of states $\llbracket \phi \rrbracket^I$ in which a formula ϕ is true will come in handy for defining a semantics of hybrid games.

The logic dGL has a denotational semantics. The dGL semantics defines, for each formula ϕ , the set $\llbracket \phi \rrbracket^I$ of states in which ϕ is true. For each hybrid game α and each set of winning states X , the dGL semantics defines the set $\varsigma_\alpha(X)$ of states from which Angel has a winning strategy to achieve X in hybrid game α , as well as the set $\delta_\alpha(X)$

of states from which Demon has a winning strategy to achieve X in α .

A *state* ν is a mapping from variables to \mathbb{R} . An *interpretation* I assigns a relation $I(p) \subseteq \mathbb{R}^k$ to each predicate symbol p of arity k . The interpretation further determines the set of states \mathcal{S} , which is isomorphic to a Euclidean space \mathbb{R}^n when n is the number of relevant variables. For a subset $X \subseteq \mathcal{S}$ the complement $\mathcal{S} \setminus X$ is denoted X^c . Let ν_x^d denote the state that agrees with state ν except for the interpretation of variable x , which is changed to $d \in \mathbb{R}$. The value of term θ in state ν is denoted by $\llbracket \theta \rrbracket_\nu$. The denotational semantics of dGL formulas will be defined in Def. 1 by simultaneous induction along with the denotational semantics, $\varsigma_\alpha(\cdot)$ and $\delta_\alpha(\cdot)$, of hybrid games, defined later, because dGL formulas are defined by simultaneous induction with hybrid games. The (denotational) semantics of a hybrid game α defines for each interpretation I and each set of Angel's winning states $X \subseteq \mathcal{S}$ the *winning region*, i.e. the set of states $\varsigma_\alpha(X)$ from which Angel has a winning strategy to achieve X (whatever strategy Demon chooses). The *winning region* of Demon, i.e. the set of states $\delta_\alpha(X)$ from which Demon has a winning strategy to achieve X (whatever strategy Angel chooses) is defined later as well.

Definition 1 (dGL semantics). The semantics of a dGL formula ϕ for each interpretation I with a corresponding set of states \mathcal{S} is the subset $\llbracket \phi \rrbracket^I \subseteq \mathcal{S}$ of states in which ϕ is true. It is defined inductively as follows

1. $\llbracket p(\theta_1, \dots, \theta_k) \rrbracket^I = \{\nu \in \mathcal{S} : (\llbracket \theta_1 \rrbracket_\nu, \dots, \llbracket \theta_k \rrbracket_\nu) \in I(p)\}$
That is, the set of states in which a predicate $p(\theta_1, \dots, \theta_k)$ is true is the set of states ν in which the tuple $(\llbracket \theta_1 \rrbracket_\nu, \dots, \llbracket \theta_k \rrbracket_\nu)$ of values of the terms θ_i in ν is in the relation $I(p)$ associated to predicate symbol p .
2. $\llbracket \theta_1 \geq \theta_2 \rrbracket^I = \{\nu \in \mathcal{S} : \llbracket \theta_1 \rrbracket_\nu \geq \llbracket \theta_2 \rrbracket_\nu\}$
That is, the set of states in which $\theta_1 \geq \theta_2$ is true is the set in which the value of θ_1 is greater than or equal to the value θ_2 .
3. $\llbracket \neg \phi \rrbracket^I = (\llbracket \phi \rrbracket^I)^c$
That is, the set of states in which $\neg \phi$ is true is the complement of the set of states in which ϕ is true.
4. $\llbracket \phi \wedge \psi \rrbracket^I = \llbracket \phi \rrbracket^I \cap \llbracket \psi \rrbracket^I$
That is, the set of states in which $\phi \wedge \psi$ is true is the intersection of the states in which ϕ is true with the set of states in which ψ is true.
5. $\llbracket \exists x \phi \rrbracket^I = \{\nu \in \mathcal{S} : \nu_x^r \in \llbracket \phi \rrbracket^I \text{ for some } r \in \mathbb{R}\}$
That is, the states in which $\exists x \phi$ is true are those which only differ in the value of x from a state in which ϕ is true.
6. $\llbracket \langle \alpha \rangle \phi \rrbracket^I = \varsigma_\alpha(\llbracket \phi \rrbracket^I)$
That is, the set of states in which $\langle \alpha \rangle \phi$ is true is Angel's winning region to achieve $\llbracket \phi \rrbracket^I$ in hybrid game α , i.e. the set of states from which Angel has a winning strategy in hybrid game α to reach a state where ϕ holds.
7. $\llbracket [\alpha] \phi \rrbracket^I = \delta_\alpha(\llbracket \phi \rrbracket^I)$
That is, the set of states in which $[\alpha] \phi$ is true is Demon's winning region to achieve $\llbracket \phi \rrbracket^I$ in hybrid game α , i.e. the set of states from which Demon has a winning strategy in hybrid game α to reach a state where ϕ holds.

A dGL formula ϕ is *valid in I* , written $I \models \phi$, iff $\llbracket \phi \rrbracket^I = \mathcal{S}$. Formula ϕ is *valid*, $\models \phi$, iff $I \models \phi$ for all interpretations I .

The semantics $\varsigma_\alpha(X)$ and $\delta_\alpha(X)$ of Angel's and Demon's winning regions still needs to be defined, which is the next goal.

Note that the semantics of $\langle \alpha \rangle \phi$ cannot be defined as it would in dL via

$$\llbracket \langle \alpha \rangle \phi \rrbracket^I = \left\{ \nu \in \mathcal{S} : \omega \in \llbracket \phi \rrbracket^I \text{ for some } \omega \text{ with } (\nu, \omega) \in \rho(\alpha) \right\}$$

First of all, the reachability relation $(\nu, \omega) \in \rho(\alpha)$ is only defined when α is a hybrid program, not when it is a hybrid game. But the deeper reason is that the above shape is

too harsh. Criteria of this shape would require Angel to single out a single state ν that satisfies the winning condition $\omega \in \llbracket \phi \rrbracket^I$ and then get to that state ω by playing α from ν . Yet all that Demon then has to do to spoil that plan is lead the play into a different state (e.g., one in which Angel would also have won) but which is different from the projected ω . More generally, winning into a single state is really difficult. Winning by leading the play into one of several states that satisfy the winning condition is more feasible. This is what the winning region $\varsigma_\alpha(\llbracket \phi \rrbracket^I)$ is supposed to capture. It captures the set of states from which Angel has a winning strategy in hybrid game α to achieve one of the states in which ϕ holds true. What a beneficial coincidence that the semantics of dGL formulas was already defined in terms of the set of states in which they are true.

3 Winning Regions

Def. 1 needs a definition of the winning regions $\varsigma_\alpha(\cdot)$ and $\delta_\alpha(\cdot)$ for Angel and Demon, respectively, in the hybrid game α . Rather than taking a detour for understanding those by operational game semantics (as in [Lecture 20](#)), the winning regions of hybrid games can be defined directly, giving a denotational semantics to hybrid games.¹

¹The semantics of a hybrid game is not merely a reachability relation between states as for hybrid systems [\[Pla12\]](#), because the adversarial dynamic interactions and nested choices of the players have to be taken into account. For brevity, the following informal explanations sometimes say “win the game” when really they mean “have a winning strategy to win the game”.

Definition 2 (Semantics of hybrid games). The *semantics of a hybrid game* α is a function $\varsigma_\alpha(\cdot)$ that, for each interpretation I and each set of Angel's winning states $X \subseteq \mathcal{S}$, gives the *winning region*, i.e. the set of states $\varsigma_\alpha(X)$ from which Angel has a winning strategy to achieve X (whatever strategy Demon chooses). It is defined inductively as follows

1. $\varsigma_{x:=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
That is, an assignment $x := \theta$ wins a game into X from any state whose modification $\nu_x^{\llbracket \theta \rrbracket}$ after the change $x := \theta$ is in X .
2. $\varsigma_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for some } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
That is, Angel wins the differential equation game $x' = \theta \& H$ into X from any state $\varphi(0)$ from which there is a solution φ of $x' = \theta$ of any duration r that remains within H all the time and leads to a state $\varphi(r) \in X$ in the end.
3. $\varsigma_{?H}(X) = \llbracket H \rrbracket^I \cap X$
That is, Angel wins into X for a challenge $?H$ from the states which satisfy H to pass the challenge and are already in X , because challenges $?H$ do not change the state.
4. $\varsigma_{\alpha \cup \beta}(X) = \varsigma_\alpha(X) \cup \varsigma_\beta(X)$
That is, Angel wins a game of choice $\alpha \cup \beta$ into X whenever she wins α into X or wins β into X (by choosing a subgame she has a winning strategy for).
5. $\varsigma_{\alpha; \beta}(X) = \varsigma_\alpha(\varsigma_\beta(X))$
That is Angel wins a sequential game $\alpha; \beta$ into X whenever she has a winning strategy in game α to achieve $\varsigma_\beta(X)$, i.e. to make it to one of the states from which she has a winning strategy in game β to achieve X .
6. $\varsigma_{\alpha^*}(X)$ will be defined later.
7. $\varsigma_{\alpha^d}(X) = (\varsigma_\alpha(X^c))^c$
That is, Angel wins α^d to achieve X in exactly the states in which she does not have a winning strategy in game α to achieve the opposite X^c .

Demon's winning regions are defined accordingly (Def. 3).

Definition 3 (Semantics of hybrid games, continued). The *winning region* of Demon, i.e. the set of states $\delta_\alpha(X)$ from which Demon has a winning strategy to achieve X (whatever strategy Angel chooses) is defined inductively as follows

1. $\delta_{x:=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
That is, an assignment $x := \theta$ wins a game into X from any state whose modification $\nu_x^{\llbracket \theta \rrbracket}$ after the change $x := \theta$ is in X .
2. $\delta_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for all } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
That is, Demon wins the differential equation game $x' = \theta \& H$ into X from any state $\varphi(0)$ from which all solutions φ of $x' = \theta$ of any duration r that remain within H all the time lead to states $\varphi(r) \in X$ in the end.
3. $\delta_{?H}(X) = (\llbracket H \rrbracket^I)^\complement \cup X$
That is, Demon wins into X for a challenge $?H$ from the states which violate H so that Angel fails her challenge $?H$ or that are already in X , because challenges $?H$ do not change the state.
4. $\delta_{\alpha \cup \beta}(X) = \delta_\alpha(X) \cap \delta_\beta(X)$
That is, Demon wins a game of choice $\alpha \cup \beta$ into X whenever he wins α into X and also wins β into X (because Angel might choose either subgame).
5. $\delta_{\alpha;\beta}(X) = \delta_\alpha(\delta_\beta(X))$
That is Demon wins a sequential game $\alpha;\beta$ into X whenever he has a winning strategy in game α to achieve $\delta_\beta(X)$, i.e. to make it to one of the states from which he has a winning strategy in game β to achieve X .
6. $\delta_{\alpha^*}(X)$ will be defined later.
7. $\delta_{\alpha^d}(X) = (\delta_\alpha(X^\complement))^\complement$
That is, Demon wins α^d to achieve X in exactly the states in which he does not have a winning strategy in game α to achieve the opposite X^\complement .

This notation uses $\varsigma_\alpha(X)$ instead of $\varsigma_\alpha^I(X)$ and $\delta_\alpha(X)$ instead of $\delta_\alpha^I(X)$, because the interpretation I that gives a semantics to predicate symbols in tests and evolution domains is clear from the context. Strategies do not occur explicitly in the **dGL** semantics, because it is based on the existence of winning strategies, not on the strategies themselves. The winning regions for Angel are illustrated in Fig. 1.

Just as the semantics **dL**, the semantics of **dGL** is *compositional*, i.e. the semantics of a compound **dGL** formula is a simple function of the semantics of its pieces, and the semantics of a compound hybrid game is a function of the semantics of its pieces. Furthermore, existence of a strategy in hybrid game α to achieve X is independent of any game and **dGL** formula surrounding α , but just depends on the remaining game α itself and the goal X . By a simple inductive argument, this shows that one can focus

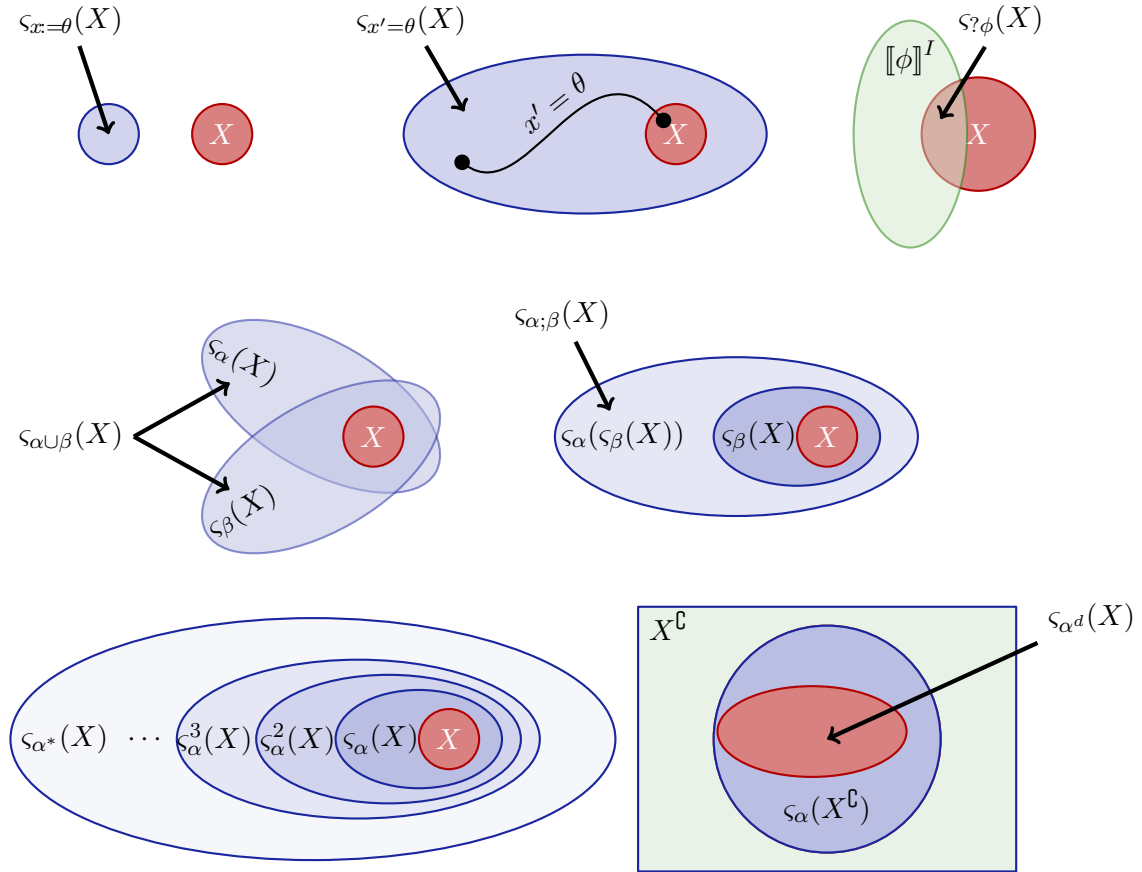


Figure 1: Illustration of denotational semantics of hybrid games as winning regions

on memoryless strategies, because the existence of strategies does not depend on the context, hence, by working bottom up, the strategy itself cannot depend on past states and choices, only the current state, remaining game, and goal. This also follows from a generalization of a classical result by Zermelo. Furthermore, the semantics is monotone, i.e. larger sets of winning states induce larger winning regions, because it is easier to win into larger sets of winning states.

Lemma 4 (Monotonicity [Pla13]). *The semantics is monotone, i.e. $\varsigma_\alpha(X) \subseteq \varsigma_\alpha(Y)$ and $\delta_\alpha(X) \subseteq \delta_\alpha(Y)$ for all $X \subseteq Y$.*

Proof. A simple check based on the observation that X only occurs with an even number of negations in the semantics. For example, $X \subseteq Y$ implies $X^\complement \supseteq Y^\complement$, hence $\varsigma_\alpha(X^\complement) \supseteq \varsigma_\alpha(Y^\complement)$, so $\varsigma_{\alpha^d}(X) = (\varsigma_\alpha(X^\complement))^\complement \subseteq (\varsigma_\alpha(Y^\complement))^\complement = \varsigma_{\alpha^d}(Y)$. \square

Before going any further, however, we need to define a semantics for repetition, which will turn out to be surprisingly difficult.

4 Advance Notice Repetitions

Def. 2 is still missing a definition for the semantics of repetition in hybrid games. The semantics of repetition in hybrid systems was

$$\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$$

with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?true$.

The obvious counterpart for the semantics of repetition in hybrid games would, thus, be

$$\varsigma_{\alpha^*}(X) \stackrel{?}{=} \bigcup_{n < \omega} \varsigma_{\alpha^n}(X) \quad (1)$$

where ω is the first infinite ordinal (if you have never seen ordinals before, just read $n < \omega$ as n in natural numbers, i.e. as $n \in \mathbb{N}$). Would that give the intended meaning to repetition? Is Angel forced to stop in order to win if the game of repetition would be played this way? Yes, she would, because, even though there is no bound on the number of repetitions that she can choose, for each natural number n , the resulting game $\varsigma_{\alpha^n}(X)$ is finite.

Would this definition capture the intended meaning of repeated game play?

Before you read on, see if you can find the answer for yourself.

The issue is that each way of playing a repetition according to (1) would require Angel to choose a natural number $n \in \mathbb{N}$ of repetitions and *expose this number to Demon* when playing α^n so that he would know how often Angel decided to repeat.

That would lead to what is called the *advance notice semantics* for α^* , which requires the players to announce the number of times that game α will be repeated when the loop begins. The advance notice semantics defines $\varsigma_{\alpha^*}(X)$ as $\bigcup_{n < \omega} \varsigma_{\alpha^n}(X)$ where $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?true$ and defines $\delta_{\alpha^*}(X)$ as $\bigcap_{n < \omega} \delta_{\alpha^n}(X)$. When playing α^* , Angel, thus, announces to Demon how many repetitions n are going to be played when the game α^* begins and Demon announces how often to repeat α^\times . This advance notice makes it easier for Demon to win loops α^* and easier for Angel to win loops α^\times , because the opponent announces an important feature of their strategy immediately as opposed to revealing whether or not to repeat the game once more one iteration at a time as in Def. 2. Angel announces the number $n < \omega$ of repetitions when α^* starts.

The following formula, for example, turns out to be valid in dGL (see Fig. 2), but would not be valid in the advance notice semantics:

$$x = 1 \wedge a = 1 \rightarrow \langle ((x := a; a := 0) \cap x := 0)^* \rangle x \neq 1 \quad (2)$$

If, in the advance notice semantics, Angel announces that she has chosen n repetitions of the game, then Demon wins (for $a \neq 0$) by choosing the $x := 0$ option $n - 1$ times followed by one choice of $x := a; a := 0$ in the last repetition. This strategy would not work in the dGL semantics, because Angel is free to decide whether to repeat α^* after each repetition based on the resulting state of the game. The winning strategy for (2) indicated in Fig. 2(left) shows that this dGL formula is valid.

Since the advance notice semantics misses out on the existence of perfectly reasonable winning strategies, dGL does not choose this semantics. Nevertheless, the advance notice semantics can be a useful semantics to consider for other purposes [QP12]. But it is not interactive enough for proper hybrid game play.

5 ω -Strategic Semantics

The trouble with the semantics in Sect. 4 is that Angel's move for the repetition reveals too much to Demon, because Demon can inspect the remaining game α^n to find out once and for all how long the game will be played before he has to do his first move.

Let's try to undo this. Another alternative choice for the semantics would have been to allow only arbitrary finite iterations of the strategy function for computing the winning region by using the ω -strategic semantics, which defines

$$\varsigma_{\alpha^*}(X) \stackrel{?}{=} \varsigma_{\alpha}^{\omega}(X) = \bigcup_{n < \omega} \varsigma_{\alpha^n}(X)$$

along with a corresponding definition for $\delta_{\alpha^*}(X)$. All we need to do for this is define what it means to nest the winning region construction. For any winning condition

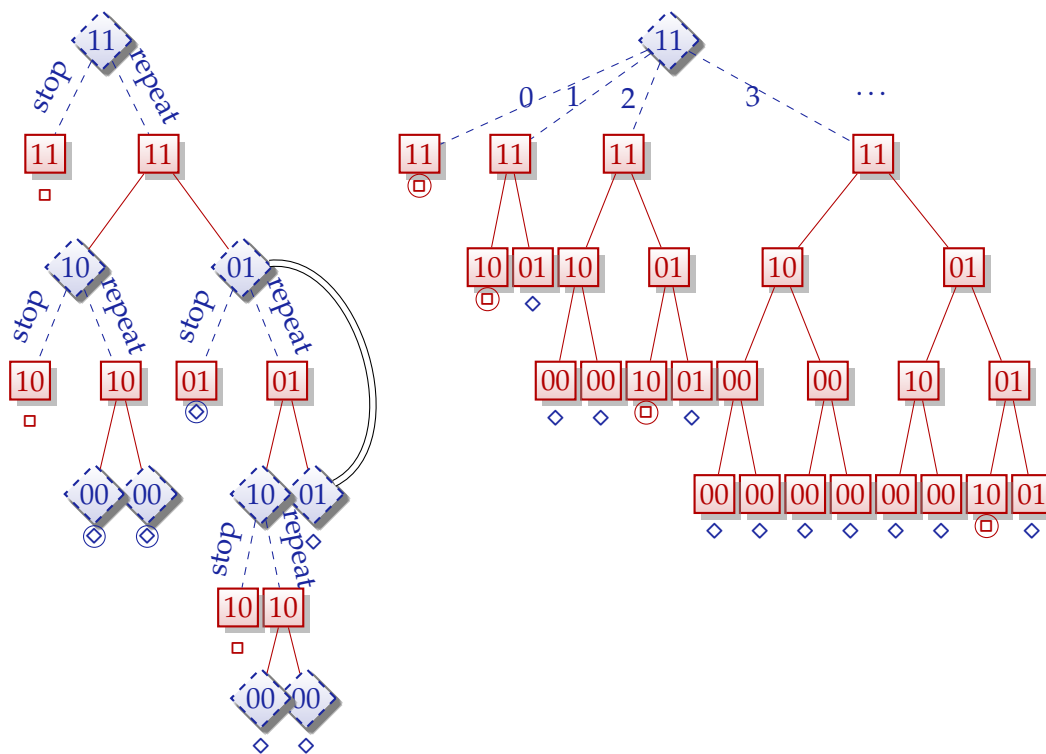


Figure 2: Game trees for $x = 1 \wedge a = 1 \rightarrow \langle \alpha^* \rangle x \neq 1$ with game $\alpha \equiv (x := a; a := 0) \cap x := 0$ (notation: x, a). **(left)** valid in dGL by strategy “repeat once and repeat once more if $x = 1$, then stop” **(right)** false in advance notice semantics by the strategy “ $n - 1$ choices of $x := 0$ followed by $x := a; a := 0$ once”, where n is the number of repetitions Angel announced

$X \subseteq \mathcal{S}$ the iterated winning region of α is defined inductively as:

$$\begin{aligned}\varsigma_{\alpha}^0(X) &\stackrel{\text{def}}{=} X \\ \varsigma_{\alpha}^{\kappa+1}(X) &\stackrel{\text{def}}{=} X \cup \varsigma_{\alpha}(\varsigma_{\alpha}^{\kappa}(X))\end{aligned}$$

The only states from which a repetition can win without actually repeating are the ones that start at the goal X already ($\varsigma_{\alpha}^0(X) = X$). And the states from which a repetition can win into X with $\kappa + 1$ repetitions are those that start in X as well as all the states for which there is a winning strategy in the hybrid game α to achieve a state in $\varsigma_{\alpha}^{\kappa}(X)$.

Does this give the right semantics for repetition of hybrid games? Does it match the existence of winning strategies that we were hoping to define? See Fig. 3 for an illustration.

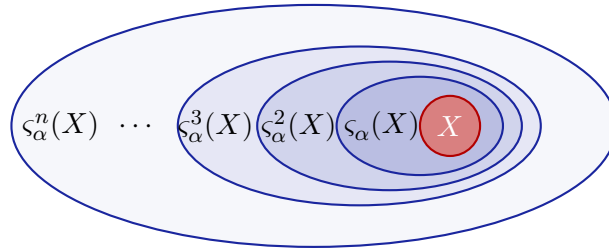


Figure 3: Iteration $\varsigma_{\alpha}^n(X)$ of $\varsigma_{\alpha}(\cdot)$ from winning condition X .

Before you read on, see if you can find the answer for yourself.

The surprising answer is *no* for a very subtle but also very fundamental reason. The existence of winning strategies for α^* does not coincide with the ω th iteration of α .

Would the following **dGL** formula be valid in the ω -strategic semantics?

$$\langle (x := 1; x' = 1^d \cup x := x - 1)^* \rangle (0 \leq x < 1) \quad (3)$$

Before you read on, see if you can find the answer for yourself.

Abbreviate

$$\underbrace{\underbrace{\langle x := 1; x' = 1^d \rangle}_{\beta} \cup \underbrace{\langle x := x - 1 \rangle}_{\gamma}}_{\alpha}^* (0 \leq x < 1)$$

It is easy to see that $\varsigma_{\alpha}^{\omega}([0, 1)) = [0, \infty)$, because $\varsigma_{\alpha}^n([0, 1)) = [0, n + 1)$ for all $n \in \mathbb{N}$ by a simple inductive proof (recall $\alpha \equiv \beta \cup \gamma$):

$$\begin{aligned} \varsigma_{\beta \cup \gamma}^0([0, 1)) &= [0, 1) \\ \varsigma_{\beta \cup \gamma}^{n+1}([0, 1)) &= [0, 1) \cup \varsigma_{\beta \cup \gamma}(\varsigma_{\beta \cup \gamma}^n([0, 1))) \stackrel{\text{IH}}{=} [0, 1) \cup \varsigma_{\beta \cup \gamma}([0, n + 1)) \\ &= [0, 1) \cup \varsigma_{\beta}([0, n + 1)) \cup \varsigma_{\gamma}([0, n)) = [0, 1) \cup \emptyset \cup [1, n + 2) = [0, n + 1 + 1) \end{aligned}$$

Consequently,

$$\varsigma_{\alpha}^{\omega}([0, 1)) = \bigcup_{n < \omega} \varsigma_{\alpha}^n([0, 1)) = \bigcup_{n < \omega} [0, n + 1) = [0, \infty)$$

Hence, the ω -semantics would indicate that the hybrid game (3) can exactly be won from all initial states in $[0, \infty)$, that is, for all initial states that satisfy $0 \leq x$.

Unfortunately, this is quite some nonsense. Indeed, the hybrid game in dG \mathcal{L} formula (3) can be won from all initial states that satisfy $0 \leq x$. But it can also be won from other initial states! So the ω -strategic semantics $\varsigma_{\alpha}^{\omega}([0, 1))$ misses out on winning states. It is way too small for a winning region. There are cases, where the ω -semantics is minuscule compared to the true winning region and arbitrarily far away from the truth [Pla13].

In (3), this ω -level of iteration of the strategy function for winning regions misses out on Angel's perfectly reasonable winning strategy "first choose $x := 1; x' = 1^d$ and then always choose $x := x - 1$ until stopping at $0 \leq x < 1$ ". This winning strategy wins from every initial state in \mathbb{R} , which is a much bigger set than $\varsigma_{\alpha}^{\omega}([0, 1)) = [0, \infty)$.

Now this is the final answer for the winning region of (3). In particular, the dG \mathcal{L} formula (3) is valid. Yet, is there a direct way to see that $\varsigma_{\alpha}^{\omega}([0, 1)) = [0, \infty)$ is not the final answer for (3) without having to put the winning region computations aside and constructing a separate ingenious winning strategy?

Before you read on, see if you can find the answer for yourself.

The crucial observation is the following. The fact $\varsigma_\alpha^\omega([0, 1)) = [0, \infty)$ shows that the hybrid game in (3) can be won from all nonnegative initial values with at most ω (“first countably infinitely many”) steps. Let’s recall how the proof worked, which showed $\varsigma_\alpha^n([0, 1)) = [0, n)$ for all $n \in \mathbb{N}$. Its inductive step basically showed that if, for whatever reason (by inductive hypothesis really), $[0, n)$ is in the winning region, then $[0, n + 1)$ also is in the winning region by simply applying $\varsigma_\alpha(\cdot)$ to $[0, n)$.

How about doing exactly that again? For whatever reason (i.e. by the above argument), $[0, \infty)$ is in the winning region. Doesn’t that mean that $\varsigma_\alpha([0, \infty))$ should again be in the winning region by exactly the same inductive argument above?

Before you read on, see if you can find the answer for yourself.

Note 5 (+1 argument). Whenever a set Y is in the winning region $\varsigma_{\alpha^*}(X)$ of repetition, then $\varsigma_{\alpha}(Y)$ also should be in the winning region $\varsigma_{\alpha^*}(X)$, because it is just one step away from Y and α^* could simply repeat once more. That is

$$Y \subseteq \varsigma_{\alpha^*}(X) \text{ then } \varsigma_{\alpha}(Y) \subseteq \varsigma_{\alpha^*}(X)$$

Applying Note 5 to the situation at hand works as follows. The above inductive proof showed $\varsigma_{\alpha}^{\omega}([0, 1)) = [0, \infty)$, which explains that at least $[0, \infty) \subseteq \varsigma_{(\beta \cup \gamma)^*}([0, 1))$ is in the winning region of repetition. By Note 5, the winning region $\varsigma_{(\beta \cup \gamma)^*}([0, 1))$ should, thus, also contain its one-step winning region $\varsigma_{\beta \cup \gamma}([0, \infty)) \subseteq \varsigma_{(\beta \cup \gamma)^*}([0, 1))$. Computing what that is gives

$$\varsigma_{\beta \cup \gamma}([0, \infty)) = \varsigma_{\beta}([0, \infty)) \cup \varsigma_{\gamma}([0, \infty)) = \mathbb{R} \cup [0, \infty) = \mathbb{R}$$

Beyond that, the winning region cannot contain anything else, because \mathbb{R} is the whole state space already and it is kind of hard to add anything to that. And, indeed, trying to use the winning region construction once more on \mathbb{R} does not change the result:

$$\varsigma_{\beta \cup \gamma}(\mathbb{R}) = \varsigma_{\beta}(\mathbb{R}) \cup \varsigma_{\gamma}(\mathbb{R}) = \mathbb{R} \cup [0, \infty) = \mathbb{R}$$

This result, then coincides with what the ingenious winning strategy above told us as well: formula (3) is valid, because there is a winning strategy for Angel from every initial state. Except that the repeated $\varsigma_{\beta \cup \gamma}(\cdot)$ winning region construction seems more systematic than an ingenious guess of a smart winning strategy. So it gives a more constructive and explicit semantics.

Let's recap. In order to find the winning region of the hybrid game described in (3), it took us not just infinitely many steps, but more than that. After ω many iterations to arrive at $\varsigma_{\alpha}^{\omega}([0, 1)) = [0, \infty)$, it took us one more step to arrive at

$$\varsigma_{(\beta \cup \gamma)^*}([0, 1)) = \varsigma_{\alpha}^{\omega+1}([0, 1)) = \mathbb{R}$$

where we denote the number of steps we took overall by $\omega + 1$, since it was one more step than (first countable) infinitely many (i.e. ω many); see Fig. 4 for an illustration. More than infinitely many steps to get somewhere are plenty. Even worse: there are cases where even $\omega + 1$ has not been enough of iteration to get to the repetition. The number of iterations needed to find $\varsigma_{\alpha^*}(X)$ could in general be much larger [Pla13].

The existence of the above winning strategy is only found at the level $\varsigma_{\alpha}^{\omega+1}([0, 1)) = \varsigma_{\alpha}([0, \infty)) = \mathbb{R}$. Even though any particular use of the winning strategy in any game play uses only some finite number of repetitions of the loop, the argument why it will always work requires $> \omega$ many iterations of $\varsigma_{\alpha}(\cdot)$, because Demon can change x to an arbitrarily big value, so that ω many iterations of $\varsigma_{\alpha}(\cdot)$ are needed to conclude that Angel has a winning strategy for any positive value of x . There is no smaller upper bound on the number of iterations it takes Angel to win, in particular Angel cannot promise ω as a bound on the repetition count, which is what the ω -semantics would effectively require her to do. But strategies do converge after $\omega + 1$ iterations.

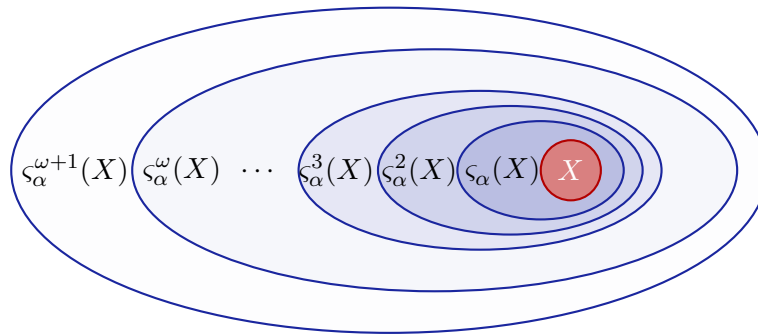


Figure 4: Iteration $\varsigma_\alpha^{\omega+1}(X)$ of $\varsigma_\alpha(\cdot)$ from winning condition $X = [0, 1)$ stops when applying $\varsigma_\alpha(\cdot)$ to the ω th infinite iteration $\varsigma_\alpha^\omega(X)$.

Note 6. The ω -semantics is inappropriate, because it can be arbitrarily far away from characterizing the winning region of hybrid games.

More generally, the semantics of repetition could be defined using

$$\begin{aligned}\varsigma_\alpha^0(X) &\stackrel{\text{def}}{=} X \\ \varsigma_\alpha^{\kappa+1}(X) &\stackrel{\text{def}}{=} X \cup \varsigma_\alpha(\varsigma_\alpha^\kappa(X)) \\ \varsigma_\alpha^\lambda(X) &\stackrel{\text{def}}{=} \bigcup_{\kappa < \lambda} \varsigma_\alpha^\kappa(X) \quad \lambda \neq 0 \text{ a limit ordinal}\end{aligned}$$

where we keep on computing winning regions at limit ordinals λ such as ω as the union of all previous winning regions. The semantics of repetition could then be defined as the union of all winning regions for all ordinals:

$$\varsigma_{\alpha^*}(X) = \varsigma^\infty(\alpha)X = \bigcup_{\kappa \text{ ordinal}} \varsigma_\alpha^\kappa(X)$$

Note 7. Unfortunately, hybrid games might require rather big infinite ordinals until this inflationary style of computing their winning regions stops [Pla14]. That translates into an infinite amount of work and then some more, infinitely often, to compute the winning region starting from \emptyset . Hardly the sort of thing we would like to wait for until we finally know who wins the game.

Finally look back at dGL formula (3) and observe what the above argument about the winning region computation terminating at $\omega + 1$ implies about bounds on how long it takes Angel to win the game in (3). Since the winning region only terminates at $\omega + 1$, she could not win with any finite bound $n \in \mathbb{N}$ on the number of repetitions it takes her to win. Even though she will surely win in the end according to her winning strategy, she has no way of saying how long that would take.

Not that Angels would ever do that. But suppose she were to brag to impress Demon by saying she could win within $n \in \mathbb{N}$ repetitions, then it would be hard for her to keep that promise. No matter how big a bound $n \in \mathbb{N}$ she chose, Demon could always spoil it from any negative initial state by evolving his differential equation $x' = 1^d$ for longer than n time units so that it takes Angel more than n rounds to decrease the resulting value down to $[0, 1)$ again.

This illustrates the dual of the discussion on the advance notice semantics in Sect. 4, which showed that Demon could make Angel win faster than she announced just to make her lose in the final round. In (3), Demon can always make Angel win later than she promised even if she ultimately will still win. This is the sense in which $\omega + 1$ is the only bound on the number of rounds it takes Angel to win the hybrid game in (3). This shows that a variation of the advance notice semantics based on Angel announcing to repeat at most $n \in \mathbb{N}$ times (as opposed to exactly $n \in \mathbb{N}$ times) would not capture the semantics of repetition appropriately.

Expedition 1 (Ordinal arithmetic). Ordinals extend natural numbers. Natural numbers are inductively defined as the (smallest) set \mathbb{N} containing 0 and the successor $n + 1$ of every number $n \in \mathbb{N}$ that is in the set. Natural numbers are totally ordered. Given any two different natural numbers, one number is going to be strictly smaller than the other one. For every finite set of natural numbers there is a smallest natural number that's bigger than all of them. Ordinals extend this beyond infinity. They just refuse to stop after all natural numbers have been written down:

$$0 < 1 < 2 < 3 < \dots$$

Taking all those (countably infinitely many) natural numbers $\{0, 1, 2, 3, \dots\}$, there is a smallest ordinal that's bigger than all of them. This ordinal is ω , the first infinite ordinal.^a

$$0 < 1 < 2 < 3 < \dots < \omega$$

Unlike the ordinals $1, 2, 3, \dots$ from the natural numbers, the ordinal ω is a *limit ordinal*, because it is not the successor of any other ordinal. The ordinals $1, 2, 3, \dots$ are *successor ordinals*, because each of them is the successor $n + 1$ of another ordinal n . The ordinal 0 is special, because it is not a successor ordinal of any ordinal or natural number.

Now, since ordinals are keen on satisfying that every ordinal has a successor, or that every set of ordinals has an ordinal that is bigger, ω must have a successor as well. Its successor is the successor ordinal $\omega + 1$, the successor of which is $\omega + 2$ and so on:

$$0 < 1 < 2 < \dots < \omega < \omega + 1 < \omega + 2 < \dots$$

Of course, in ordinal land, there ought to be an ordinal that's bigger than even all of those ordinals as well. It's the limit ordinal $\omega + \omega = \omega \cdot 2$, at which point we have counted to countable infinity twice already and will keep on finding bigger

ordinals, because even $\omega \cdot 2$ will have a successor, namely $\omega \cdot 2 + 1$:

$$0 < 1 < 2 < \dots < \omega < \omega + 1 < \omega + 2 < \dots \omega \cdot 2 < \omega \cdot 2 + 1 < \omega \cdot 2 + 2 < \dots$$

Now the set of all these will have a bigger ordinal $\omega \cdot 2 + \omega = \omega \cdot 3$, which again has successors and so on. That happens infinitely often so that $\omega \cdot n$ will be an ordinal for any natural number $n \in \mathbb{N}$. All those infinitely many ordinals will also have a limit ordinal that's bigger than all of them, which is $\omega \cdot \omega = \omega^2$. That one again has a successor $\omega^2 + 1$ and so on, also see Fig. 5:

$$0 < 1 < 2 < \dots \omega < \omega + 1 < \omega + 2 < \dots \omega \cdot 2 < \omega \cdot 2 + 1 < \dots \omega \cdot 3 < \omega \cdot 3 + 1 < \dots \\ \omega^2 < \omega^2 + 1 < \dots \omega^2 + \omega < \omega^2 + \omega + 1 < \dots \omega^\omega < \dots \omega^{\omega^\omega} < \dots \omega_1^{\text{CK}} < \dots \omega_1 < \dots$$

The first infinite ordinal is ω , the Church-Kleene ordinal ω_1^{CK} , i.e. the first nonrecursive ordinal, and ω_1 is the first uncountable ordinal. Every ordinal κ is either a successor ordinal, i.e. the smallest ordinal $\kappa = \iota + 1$ greater than some ordinal ι , or a limit ordinal, i.e. the supremum of all smaller ordinals. Depending on the context, 0 is considered a limit ordinal or separate.

Ordinals support (non-commutative) addition, multiplication, and exponentiation, which can be defined by induction on its second argument:

$$\begin{array}{ll} \iota + 0 = \iota & \\ \iota + (\kappa + 1) = (\iota + \kappa) + 1 & \text{for successor ordinals } \kappa + 1 \\ \iota + \lambda = \bigsqcup_{\kappa < \lambda} \iota + \kappa & \text{for limit ordinals } \lambda \\ \iota \cdot 0 = 0 & \\ \iota \cdot (\kappa + 1) = (\iota \cdot \kappa) + \iota & \text{for successor ordinals } \kappa + 1 \\ \iota \cdot \lambda = \bigsqcup_{\kappa < \lambda} \iota \cdot \kappa & \text{for limit ordinals } \lambda \\ \iota^0 = 1 & \\ \iota^{\kappa+1} = \iota^\kappa \cdot \iota & \text{for successor ordinals } \kappa + 1 \\ \iota^\lambda = \bigsqcup_{\kappa < \lambda} \iota^\kappa & \text{for limit ordinals } \lambda \end{array}$$

where \bigsqcup denotes the supremum or least-upper bound. Carefully note ordinal oddities like the noncommutativity coming from $2 \cdot \omega = 4 \cdot \omega$ and $\omega \cdot 2 < \omega \cdot 4$.

^aFor a moment read " $\omega = \infty$ " as infinity, but you will realize in an instant that this view does not go far enough, because there will be reason to distinguish different infinities.

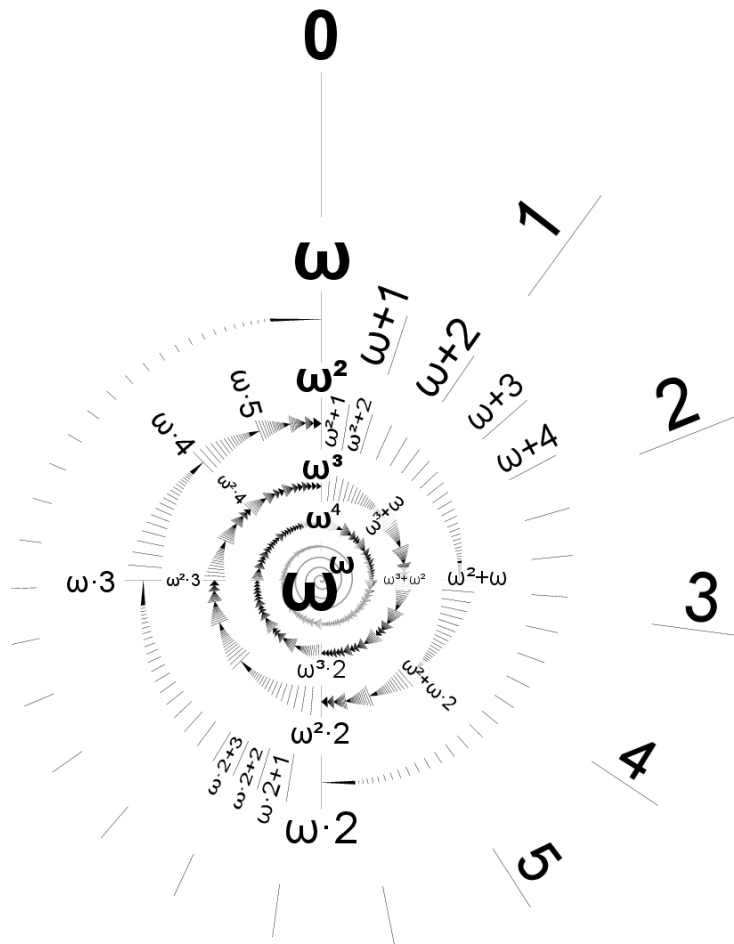


Figure 5: Illustration of infinitely many ordinals up to ω^ω

6 Summary

This lecture saw the introduction of a proper formal semantics for differential game logic and hybrid games. This resulted in a simple denotational semantics, where the meaning of all formulas and hybrid games is a simple function of the meaning of its pieces. The only possible outlier was the semantics of repetition, which turned out to be rather subtle and ultimately defined by higher-ordinal iterations of winning region constructions. This led to an insightful appreciation for the complexities, challenges, and flexibilities of hybrid games.

The next lecture will revisit the semantics of repetition to find a simpler implicit characterization and leverage the semantic basis for the next leg in the logical trinity: axiomatics.

Exercises

Exercise 1. The formula (3) was shown to need $\omega + 1$ iterations of the winning region construction to terminate with the following answer justifying the validity of (3).

$$\varsigma_{\alpha}^*([0, 1)) = \varsigma_{\alpha}^{\omega+1}([0, 1)) = \varsigma_{\alpha}([0, \infty)) = \mathbb{R}$$

What happens if the winning region construction is used again to compute $\varsigma_{\alpha}^{\omega+2}([0, 1))$? How often does the winning region construction need to be iterated to justify validity of

$$\langle (x := x + 1; x' = 1^d \cup x := x - 1)^* \rangle (0 \leq x < 1)$$

Exercise 2. How often does the winning region construction need to be iterated to justify validity of

$$\langle (x := x - 1; y' = 1^d \cup y := y - 1; z' = 1^d \cup z := z - 1)^* \rangle (x < 0 \wedge y < 0 \wedge z < 0)$$

Exercise 3 (Clockwork ω).* How often does the winning region construction need to be iterated to justify validity of

$$\langle (?y < 0; x := x - 1; y' = 1^d \cup ?z < 0; y := y - 1; z' = 1^d \cup z := z - 1)^* \rangle (x < 0 \wedge y < 0 \wedge z < 0)$$

References

- [Pla12] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012. [doi:10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla13] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.
- [Pla14] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014. [arXiv:1408.1980](https://arxiv.org/abs/1408.1980).

- [QP12] Jan-David Quesel and André Platzer. Playing hybrid games with KeYmaera. In Bernhard Gramlich, Dale Miller, and Ulrike Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 439–453. Springer, 2012. [doi:10.1007/978-3-642-31365-3_34](https://doi.org/10.1007/978-3-642-31365-3_34).

Lecture Notes on Winning & Proving Hybrid Games

[André Platzer](#)

Carnegie Mellon University
Lecture 22

1 Introduction

This lecture continues the study of hybrid games and their logic, differential game logic [[Pla13](#), [Pla14](#)], whose syntax was introduced in [Lecture 20 on Hybrid Systems & Games](#) and whose semantics was developed in [Lecture 21 on Winning Strategies & Regions](#). Today's lecture furthers the development of differential game logic to the third leg of the logical trinity: its axiomatics. This lecture will focus on the development of rigorous reasoning techniques for hybrid games as models for CPS with adversarial dynamics. Without such analysis and reasoning techniques, a logic that only comes with syntax and semantics can be used as a specification language with a precise meaning, but would not be very helpful for actually analyzing and verifying hybrid games. It is the logical trinity of syntax, semantics, and axiomatics that gives logics the power of serving as well-founded specification and verification languages with a (preferably concise) syntax, an unambiguous semantics, and actionable analytic reasoning principles. Thus, today's lecture is the hybrid games analogue of [Lecture 5 on Dynamical Systems and Dynamic Axioms](#). Indeed, after the logical sophistication we reached throughout the semester, this lecture will settle for a Hilbert-type calculus as in [Lecture 5 on Dynamical Systems and Dynamic Axioms](#) as opposed to the more refined and more automatable sequent calculus from [Lecture 6 on Truth and Proof](#) and subsequent lectures.

Before submerging completely into the development of rigorous reasoning techniques for hybrid games as models for CPS with adversarial dynamics, however, it will be wise to take a short detour by investigating a semantical simplification of the meaning of repetition by an implicit characterization of its winning region rather than the explicit construction by iteration from [Lecture 21](#).

These lecture notes are based on [[Pla13](#), [Pla14](#)], where more information can be found on logic and hybrid games. The most important learning goals of this lecture are:

Modeling and Control: We advance our understanding of the core principles behind CPS with hybrid games by understanding analytically and semantically how discrete, continuous, and the adversarial dynamics resulting, e.g., from multiple agents are integrated and interact in CPS. This lecture also uncovers nuances in the semantics of adversarial repetitions that makes them conceptually better behaved than the highly transfinite iterated winning region construction from [Lecture 21](#). A byproduct of this development shows fixpoints in actions, which play a prominent role in the understanding of other classes of models as well and provides one important aspect for the subsequent development of reasoning techniques.

Computational Thinking: This lecture is devoted to the development of rigorous reasoning techniques for CPS models involving adversarial dynamics, which is critical to getting CPS with such interactions right. Hybrid games provide even more subtle interactions than hybrid systems did, which make it even more challenging to say for sure whether and why a design is correct without sufficient rigor in their analysis. After [Lecture 21](#) captured the semantics of differential game logic and hybrid games compositionally, this lecture exploits the compositional meaning to develop compositional reasoning principles for hybrid games. This lecture systematically develops one reasoning principle for each of the operators of hybrid programs, resulting in a compositional verification approach. A compositional semantics is de facto a necessary but not a sufficient condition for the existence of compositional reasoning principles. Despite the widely generalized semantics of hybrid games compared to hybrid systems, this lecture will strive to generalize reasoning techniques for hybrid systems to hybrid games as smoothly as possible. This leads to a modular way of integrating adversariality into the realm of hybrid systems models also in terms of their analysis while simultaneously taming their complexity. This lecture provides an *axiomatization* of differential game logic dGCL [[Pla13](#), [Pla14](#)] to lift dGCL from a specification language to a verification language for CPS with adversarial dynamics.

CPS Skills: We will develop a deep understanding of the semantics of CPS models with adversariality by carefully relating their semantics to their reasoning principles and aligning them in perfect unison. This understanding will also enable us to develop a better intuition for the operational effects involved in CPS. This lecture also shows insightful and influential nuances on the semantics of repetitions in CPS models with adversarial dynamics.

In our quest to develop rigorous reasoning principles for hybrid games, we will strive to identify compositional reasoning principles that align in perfect unison with the compositional semantics of hybrid games developed in [Lecture 21 on Winning Strategies & Regions](#). This enterprise will be enlightening and, for the most part, quite successful. And, in fact, the reader is encouraged to start right away with the development of a proof calculus for differential game logic and later compare it with the one that these lecture notes develop. The part, where this will turn out to be rather difficult is repe-

tion, which is why the lecture notes take a scenic detour through characterizing their semantics.

2 Characterizing Winning Repetitions Implicitly

[Lecture 21 on Winning Strategies & Regions](#) culminated in a semantics of repetition defined as the union of all winning regions for all ordinals by an explicit (albeit wildly infinite) construction:

$$\begin{aligned} \varsigma_{\alpha^*}(X) &= \varsigma^{\infty}(\alpha)X = \bigcup_{\kappa \text{ ordinal}} \varsigma_{\alpha}^{\kappa}(X) \quad \text{where} \\ \varsigma_{\alpha}^0(X) &\stackrel{\text{def}}{=} X \\ \varsigma_{\alpha}^{\kappa+1}(X) &\stackrel{\text{def}}{=} X \cup \varsigma_{\alpha}(\varsigma_{\alpha}^{\kappa}(X)) \\ \varsigma_{\alpha}^{\lambda}(X) &\stackrel{\text{def}}{=} \bigcup_{\kappa < \lambda} \varsigma_{\alpha}^{\kappa}(X) \quad \lambda \neq 0 \text{ a limit ordinal} \end{aligned}$$

Is there a more immediate way of characterizing the winning region $\varsigma_{\alpha^*}(X)$ of repetition implicitly rather than by explicit construction? This thought will lead to a beautiful illustration of Bertrand Russell's enlightening bonmot:

The advantages of implicit definition over construction are roughly those of theft over honest toil. — Bertrand Russell (slightly paraphrased)

The above iterated winning region construction describes the semantics of repetition by iterating from below, i.e. starting from $\varsigma_{\alpha}^0(X) = X$ and adding states. Maybe the semantics of repetition could be characterized more indirectly but more concisely from above? With an implicit characterization.

Note 1 (+1 argument). *Whenever a set Z is in the winning region $\varsigma_{\alpha^*}(X)$ of repetition, then $\varsigma_{\alpha}(Z)$ also should be in the winning region $\varsigma_{\alpha^*}(X)$, because it is just one step away from Z and α^* could simply repeat once more. That is*

$$Z \subseteq \varsigma_{\alpha^*}(X) \text{ then } \varsigma_{\alpha}(Z) \subseteq \varsigma_{\alpha^*}(X)$$

This holds for any set $Z \subseteq \varsigma_{\alpha^*}(X)$. In particular, the set $Z \stackrel{\text{def}}{=} \varsigma_{\alpha^*}(X)$ itself satisfies

$$\varsigma_{\alpha}(\varsigma_{\alpha^*}(X)) \subseteq \varsigma_{\alpha^*}(X) \tag{1}$$

by [Note 1](#). After all, repeating α once more from the winning region $\varsigma_{\alpha^*}(X)$ of repetition of α cannot give us any states that did not already have a winning strategy in α^* , because α^* could have been repeated one more time. Consequently, if a set $Z \subseteq \mathcal{S}$ claims to be the winning region $\varsigma_{\alpha^*}(X)$ of repetition, it at least has to satisfy

$$\varsigma_{\alpha}(Z) \subseteq Z \tag{2}$$

because, by (1), the true winning region $\varsigma_{\alpha^*}(X)$ does satisfy (2). Thus, strategizing along α from Z does not give anything that Z would not already know about.

Is there anything else that such a set Z needs to satisfy to be the winning region $\varsigma_{\alpha^*}(X)$ of repetition? Is there only one choice? Or multiple? If there are multiple choices, which Z is it? Does such a Z always exist, even?

Before you read on, see if you can find the answer for yourself.

One such Z always exists, even though it may be rather boring. The empty set $Z \stackrel{\text{def}}{=} \emptyset$ looks like it would satisfy (2) because it is rather hard to win a game that requires Angel to enter the empty set of states to win.

On second thought, $\varsigma_\alpha(\emptyset) \subseteq \emptyset$ does not actually always hold for all hybrid games α . It is violated for states from which Angel can make sure Demon violates the rules of the game α by losing a challenge or failing to comply with evolution domain constraints. For example, when H is a nontrivial test like $x > 0$:

$$\varsigma_{?H^d}(\emptyset) = \varsigma_{?H}(\emptyset^c)^c = (\llbracket H \rrbracket^I \cap \mathcal{S})^c = (\llbracket H \rrbracket^I)^c = \llbracket \neg H \rrbracket^I \not\subseteq \emptyset$$

Yet, then the set of states that make Demon violate the rules satisfies (2) instead of \emptyset :

$$\varsigma_{?H^d}(\llbracket \neg H \rrbracket^I) = \varsigma_{?H}((\llbracket \neg H \rrbracket^I)^c)^c = \varsigma_{?H}(\llbracket H \rrbracket^I)^c = (\llbracket H \rrbracket^I \cap \llbracket H \rrbracket^I)^c = \llbracket \neg H \rrbracket^I \subseteq \llbracket \neg H \rrbracket^I$$

But even if the empty set \emptyset satisfies (2), it may be a bit small. Likewise, even if $\llbracket \neg H \rrbracket^I$ satisfies (2) for $\alpha \equiv ?H^d$, the set $\llbracket \neg H \rrbracket^I$ may still be a bit small. Angel is still in charge of repetition and can decide how often to repeat and whether to repeat at all. The winning region $\varsigma_{\alpha^*}(X)$ of repetition of α should at least contain the winning condition X , because the winning condition X is particularly easy to reach when already starting in X by simply suggesting Angel should repeat zero times. Angel would certainly love to do that, because it does not sound like a lot of work to repeat something zero times. Consequently, if a set $Z \subseteq \mathcal{S}$ claims to be the winning region $\varsigma_{\alpha^*}(X)$, then it has to satisfy (2) and

$$X \subseteq Z \tag{3}$$

Both conditions (2) and (3) together can be summarized in a single condition as follows:

Note 2 (Pre-fixpoint). Every candidate Z for the winning region $\varsigma_{\alpha^*}(X)$ satisfies:

$$X \cup \varsigma_\alpha(Z) \subseteq Z \tag{4}$$

A set Z satisfying condition (4) is called a pre-fixpoint.

Again: what is this set Z that satisfies (4)? Is there only one choice? Or multiple? If there are multiple choices, which Z is the right one for the semantics of repetition? Does such a Z always exist, even?

Before you read on, see if you can find the answer for yourself.

One such Z certainly exists. The empty set does not qualify unless $X = \emptyset$ (and even then \emptyset actually only works if Demon cannot be tricked into violating the rules of the game). The set X itself is too small as well unless the game has no incentive to start repeating, because $\varsigma_\alpha(X) \subseteq X$. But the full state space $Z \stackrel{\text{def}}{=} \mathcal{S}$ always satisfies (4) trivially so (4) has a solution. Now, the whole space is a little too big to call it Angel's winning region independently of the hybrid game α . Even if the full space may very well be the winning region for some particularly Demonophobic Angel-friendly hybrid games like

$$\langle (x := 1; x' = 1^d \cup x := x - 1)^* \rangle (0 \leq x < 1) \quad (5)$$

the full state space is hardly the right winning region for any arbitrary α^* . It definitely depends on the hybrid game α and the winning condition ϕ whether Angel has a winning strategy for $\langle \alpha \rangle \phi$ or not. For example for Demon's favorite game where he always wins, $\varsigma_{\alpha^*}(X)$ had better be \emptyset , not \mathcal{S} . Thus, the largest solution Z of (4) hardly qualifies.

So which solution Z of (4) do we define to be $\varsigma_{\alpha^*}(X)$ now?

Before you read on, see if you can find the answer for yourself.

Among the many Z that solve (4), the largest one is not informative, because the largest Z simply degrades to \mathcal{S} . So smaller solutions Z are preferable. Which one? How do multiple solutions even relate to each other? Suppose Y, Z are both solutions of (4). That is

$$X \cup \varsigma_\alpha(Y) \subseteq Y \quad (6)$$

$$X \cup \varsigma_\alpha(Z) \subseteq Z \quad (7)$$

Then, by the monotonicity lemma from [Lecture 21](#) (repeated in [Lemma 3](#) below):

$$X \cup \varsigma_\alpha(Y \cap Z) \stackrel{\text{mon}}{\subseteq} X \cup (\varsigma_\alpha(Y) \cap \varsigma_\alpha(Z)) \stackrel{(6),(7)}{\subseteq} Y \cap Z \quad (8)$$

Hence, by (8), the intersection $Y \cap Z$ of solutions Y and Z of (4) also is a solution of (4):

Lemma 1 (Intersection closure). *For any two solutions Y, Z of the prefix condition (4), a smaller solution of (4) can be obtained by intersection $Y \cap Z$.*

Whenever there are two solutions Z_1, Z_2 of (4), their intersection $Y_1 \cap Z_2$ solves (4) as well. When there's yet another solution Z_3 of (4), their intersection $Y_1 \cap Y_2 \cap Y_3$ also solves (4). Similarly for *any* larger family of solutions whose intersection will solve (4). If we keep on intersecting solutions, we will arrive at smaller and smaller solutions until, some fine day, there's not going to be a smaller one. This yields the smallest solution Z of (4) which can be characterized directly.

Note 4 (Semantics of repetitions). *Among the many Z that solve (4), $\varsigma_\alpha^*(X)$ is defined to be the smallest Z that solves (4):*

$$\varsigma_\alpha^*(X) = \bigcap \{Z \subseteq \mathcal{S} : X \cup \varsigma_\alpha(Z) \subseteq Z\} \quad (9)$$

The characterization in terms of iterated winning regions from [Lecture 21](#) leads to the same set $\varsigma_\alpha^*(X)$, but the (least pre-fixpoint or) fixpoint characterization (9) is easier.

The set on the right-hand side of (9) is an intersection of solutions, thus, a solution by [Lemma 1](#) (or its counterpart for families of solutions). Hence $\varsigma_\alpha^*(X)$ itself satisfies (4):

$$X \cup \varsigma_\alpha(\varsigma_\alpha^*(X)) \subseteq \varsigma_\alpha^*(X) \quad (10)$$

Also compare this with where we came from when we argued for (1). Could it be the case that the inclusion in (10) is strict, i.e. not equals? No this cannot happen, because $\varsigma_\alpha^*(X)$ is the smallest such set. That is, by (10), the set $Z \stackrel{\text{def}}{=} X \cup \varsigma_\alpha(\varsigma_\alpha^*(X))$ satisfies $Z \subseteq \varsigma_\alpha^*(X)$ and, thus, by [Lemma 3](#):

$$X \cup \varsigma_\alpha(Z) \stackrel{\text{mon}}{\subseteq} X \cup \varsigma_\alpha(\varsigma_\alpha^*(X)) = Z$$

Consequently, both inclusions hold, so $\varsigma_{\alpha^*}(X)$ actually satisfies not just the inclusion (4) but even the equation

$$X \cup \varsigma_{\alpha}(\varsigma_{\alpha^*}(X)) = \varsigma_{\alpha^*}(X) \quad (11)$$

Note 5 (Semantics of repetitions, fixpoint formulation). *That is, $\varsigma_{\alpha^*}(X)$ is a fixpoint solving the equation*

$$X \cup \varsigma_{\alpha}(Z) = Z \quad (12)$$

and it is the least fixpoint, i.e. the smallest Z solving the equation (12), i.e. it satisfies

$$\varsigma_{\alpha^*}(X) = \bigcap \{Z \subseteq \mathcal{S} : X \cup \varsigma_{\alpha}(Z) = Z\}$$

The fact that $\varsigma_{\alpha^*}(X)$ is defined as the least of the fixpoints makes sure that Angel only wins games by a well-founded number of repetitions. That is, she only wins a repetition if she ultimately stops repeating, not by postponing termination forever. See [Pla13, Pla14] for more details.

It is also worth noting that it would still have been possible to make the iteration of winning region constructions work out using the seminal fixpoint theorem of Knaster-Tarski. Yet, this requires the iterated winning region constructions to go significantly transfinite [Pla13, Pla14] way beyond the first infinite ordinal ω .

3 Semantics of Hybrid Games

The semantics of differential game logic from [Lecture 21](#) was still pending a definition of the winning regions $\varsigma_{\alpha}(\cdot)$ and $\delta_{\alpha}(\cdot)$ for Angel and Demon, respectively, in the hybrid game α . Rather than taking a detour for understanding those by operational game semantics (as in [Lecture 20](#)), or in terms of transfinitely iterated winning region constructions, the winning regions of hybrid games can be defined directly, giving a denotational semantics to hybrid games.

The only difference of the following semantics compared to the definition in [Lecture 21](#) is the new case of repetition α^* .

Definition 2 (Semantics of hybrid games). The *semantics of a hybrid game* α is a function $\varsigma_\alpha(\cdot)$ that, for each interpretation I and each set of Angel's winning states $X \subseteq \mathcal{S}$, gives the *winning region*, i.e. the set of states $\varsigma_\alpha(X)$ from which Angel has a winning strategy to achieve X (whatever strategy Demon chooses). It is defined inductively as follows

1. $\varsigma_{x=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
2. $\varsigma_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for some } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
3. $\varsigma_{?H}(X) = \llbracket H \rrbracket^I \cap X$
4. $\varsigma_{\alpha \cup \beta}(X) = \varsigma_\alpha(X) \cup \varsigma_\beta(X)$
5. $\varsigma_{\alpha; \beta}(X) = \varsigma_\alpha(\varsigma_\beta(X))$
6. $\varsigma_{\alpha^*}(X) = \bigcap \{Z \subseteq \mathcal{S} : X \cup \varsigma_\alpha(Z) \subseteq Z\}$
7. $\varsigma_{\alpha^d}(X) = (\varsigma_\alpha(X^{\mathbb{L}}))^{\mathbb{L}}$

The *winning region* of Demon, i.e. the set of states $\delta_\alpha(X)$ from which Demon has a winning strategy to achieve X (whatever strategy Angel chooses) is defined inductively as follows

1. $\delta_{x=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
2. $\delta_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for all } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
3. $\delta_{?H}(X) = (\llbracket H \rrbracket^I)^{\mathbb{L}} \cup X$
4. $\delta_{\alpha \cup \beta}(X) = \delta_\alpha(X) \cap \delta_\beta(X)$
5. $\delta_{\alpha; \beta}(X) = \delta_\alpha(\delta_\beta(X))$
6. $\delta_{\alpha^*}(X) = \bigcup \{Z \subseteq \mathcal{S} : Z \subseteq X \cap \delta_\alpha(Z)\}$
7. $\delta_{\alpha^d}(X) = (\delta_\alpha(X^{\mathbb{L}}))^{\mathbb{L}}$

This notation uses $\varsigma_\alpha(X)$ instead of $\varsigma_\alpha^I(X)$ and $\delta_\alpha(X)$ instead of $\delta_\alpha^I(X)$, because the interpretation I that gives a semantics to predicate symbols in tests and evolution domains is clear from the context. Strategies do not occur explicitly in the **dGL** semantics, because it is based on the existence of winning strategies, not on the strategies themselves.

Just as the semantics **dL**, the semantics of **dGL** is *compositional*, i.e. the semantics of a compound **dGL** formula is a simple function of the semantics of its pieces, and the semantics of a compound hybrid game is a function of the semantics of its pieces. Fur-

thermore, existence of a strategy in hybrid game α to achieve X is independent of any game and dG \mathcal{L} formula surrounding α , but just depends on the remaining game α itself and the goal X . By a simple inductive argument, this shows that one can focus on memoryless strategies, because the existence of strategies does not depend on the context, hence, by working bottom up, the strategy itself cannot depend on past states and choices, only the current state, remaining game, and goal. This also follows from a generalization of a classical result by Zermelo. Furthermore, the semantics is monotone, i.e. larger sets of winning states induce larger winning regions.

Monotonicity is what [Lecture 21](#) looked into for the case of hybrid games without repetition. But it continues to hold for general hybrid games.

Lemma 3 (Monotonicity [[Pla13](#)]). *The semantics is monotone, i.e. $\varsigma_\alpha(X) \subseteq \varsigma_\alpha(Y)$ and $\delta_\alpha(X) \subseteq \delta_\alpha(Y)$ for all $X \subseteq Y$.*

Proof. A simple check based on the observation that X only occurs with an even number of negations in the semantics. For example, $\varsigma_{\alpha^*}(X) = \bigcap \{Z \subseteq \mathcal{S} : X \cup \varsigma_\alpha(Z) \subseteq Z\} \subseteq \bigcap \{Z \subseteq \mathcal{S} : Y \cup \varsigma_\alpha(Z) \subseteq Z\} = \varsigma_{\alpha^*}(Y)$ if $X \subseteq Y$. Likewise, $X \subseteq Y$ implies $X^\complement \supseteq Y^\complement$, hence $\varsigma_\alpha(X^\complement) \supseteq \varsigma_\alpha(Y^\complement)$, so $\varsigma_{\alpha^d}(X) = (\varsigma_\alpha(X^\complement))^\complement \subseteq (\varsigma_\alpha(Y^\complement))^\complement = \varsigma_{\alpha^d}(Y)$. \square

Monotonicity implies that the least fixpoint in $\varsigma_{\alpha^*}(X)$ and the greatest fixpoint in $\delta_{\alpha^*}(X)$ are well-defined [[HKT00](#), Lemma 1.7]. The semantics of $\varsigma_{\alpha^*}(X)$ is a least fixpoint, which results in a well-founded repetition of α , i.e. Angel can repeat any number of times but she ultimately needs to stop at a state in X in order to win. The semantics of $\delta_{\alpha^*}(X)$ is a greatest fixpoint, instead, for which Demon needs to achieve a state in X after every number of repetitions, because Angel could choose to stop at any time, but Demon still wins if he only postpones X^\complement forever, because Angel ultimately has to stop repeating. Thus, for the formula $\langle \alpha^* \rangle \phi$, Demon already has a winning strategy if he only has a strategy that is not losing by preventing ϕ indefinitely, because Angel eventually has to stop repeating anyhow and will then end up in a state not satisfying ϕ , which makes her lose. The situation for $[\alpha^*]\phi$ is dual.

4 Determinacy

Every particular game play in a hybrid game is won by exactly one player, because hybrid games are zero-sum and there are no draws. Hybrid games actually satisfy a much stronger property: *determinacy*, i.e. that, from any initial situation, either one of the players always has a winning strategy to force a win, regardless of how the other player chooses to play.

If, from the same initial state, both Angel and Demon had a winning strategy for opposing winning conditions, then something would be terribly inconsistent. It cannot happen that Angel has a winning strategy in hybrid game α to get to a state where $\neg\phi$ and, from the same initial state, Demon supposedly also has a winning strategy in the same hybrid game α to get to a state where ϕ holds. After all, a winning strategy is

a strategy that makes that player win no matter what strategy the opponent follows. Hence, for any initial state, at most one player can have a winning strategy for complementary winning conditions. This argues for the validity of $\models \neg([\alpha]\phi \wedge \langle\alpha\rangle\neg\phi)$, which can also be proved (Theorem 4).

So it cannot happen that both players have a winning strategy for complementary winning conditions. But it might still happen that no one has a winning strategy, i.e. both players can let the other player win, but cannot win strategically themselves (recall, e.g., the filibuster example from Lecture 20, which first appeared as if no player might have a winning strategy but then turned out to make Demon win). This does not happen for hybrid games, though, because at least one (hence exactly one) player has a winning strategy for complementary winning conditions from any initial state.

Theorem 4 (Consistency & determinacy [Pla13, Pla14]). *Hybrid games are consistent and determined, i.e. $\models \neg\langle\alpha\rangle\neg\phi \leftrightarrow [\alpha]\phi$.*

Proof. The proof shows by induction on the structure of α that $\varsigma_\alpha(X^\complement)^\complement = \delta_\alpha(X)$ for all $X \subseteq \mathcal{S}$ and all I with some set of states \mathcal{S} , which implies the validity of $\neg\langle\alpha\rangle\neg\phi \leftrightarrow [\alpha]\phi$ using $X \stackrel{\text{def}}{=} \llbracket\phi\rrbracket^I$.

1. $\varsigma_{x=\theta}(X^\complement)^\complement = \{\nu \in \mathcal{S} : \nu_x^{\llbracket\theta\rrbracket} \notin X\}^\complement = \varsigma_{x=\theta}(X) = \delta_{x=\theta}(X)$
2. $\varsigma_{x'=\theta \& H}(X^\complement)^\complement = \{\varphi(0) \in \mathcal{S} : \varphi(r) \notin X \text{ for some } 0 \leq r \in \mathbb{R} \text{ and some (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket\theta\rrbracket_{\varphi(\zeta)} \text{ and } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ for all } 0 \leq \zeta \leq r\}^\complement = \delta_{x'=\theta \& H}(X)$, because the set of states from which there is no winning strategy for Angel to reach a state in X^\complement prior to leaving $\llbracket H \rrbracket^I$ along $x' = \theta \& H$ is exactly the set of states from which $x' = \theta \& H$ always stays in X (until leaving $\llbracket H \rrbracket^I$ in case that ever happens).
3. $\varsigma_{?H}(X^\complement)^\complement = (\llbracket H \rrbracket^I \cap X^\complement)^\complement = (\llbracket H \rrbracket^I)^\complement \cup (X^\complement)^\complement = \delta_{?H}(X)$
4. $\varsigma_{\alpha \cup \beta}(X^\complement)^\complement = (\varsigma_\alpha(X^\complement) \cup \varsigma_\beta(X^\complement))^\complement = \varsigma_\alpha(X^\complement)^\complement \cap \varsigma_\beta(X^\complement)^\complement = \delta_\alpha(X) \cap \delta_\beta(X) = \delta_{\alpha \cup \beta}(X)$
5. $\varsigma_{\alpha;\beta}(X^\complement)^\complement = \varsigma_\alpha(\varsigma_\beta(X^\complement))^\complement = \varsigma_\alpha(\delta_\beta(X)^\complement)^\complement = \delta_\alpha(\delta_\beta(X)) = \delta_{\alpha;\beta}(X)$
6. $\varsigma_{\alpha^*}(X^\complement)^\complement = \left(\bigcap\{Z \subseteq \mathcal{S} : X^\complement \cup \varsigma_\alpha(Z) \subseteq Z\}\right)^\complement = \left(\bigcap\{Z \subseteq \mathcal{S} : (X \cap \varsigma_\alpha(Z)^\complement)^\complement \subseteq Z\}\right)^\complement = \left(\bigcap\{Z \subseteq \mathcal{S} : (X \cap \delta_\alpha(Z^\complement))^\complement \subseteq Z\}\right)^\complement = \bigcup\{Z \subseteq \mathcal{S} : Z \subseteq X \cap \delta_\alpha(Z)\} = \delta_{\alpha^*}(X)$.¹
7. $\varsigma_{\alpha^d}(X^\complement)^\complement = (\varsigma_\alpha((X^\complement)^\complement))^\complement = \delta_\alpha(X^\complement)^\complement = \delta_{\alpha^d}(X)$ □

¹The penultimate equation follows from the μ -calculus equivalence $\nu Z. \Upsilon(Z) \equiv \neg\mu Z. \neg\Upsilon(\neg Z)$ and the fact that least pre-fixpoints are fixpoints and that greatest post-fixpoints are fixpoints for monotone functions.

5 Hybrid Game Axioms

An axiomatization for differential game logic has been found in [Pla13, Pla14], where we refer to for more details.

Note 9 (Differential game logic axiomatization [Pla13, Pla14]).

$$([\cdot]) \quad [\alpha]\phi \leftrightarrow \neg\langle\alpha\rangle\neg\phi$$

$$(\langle:=\rangle) \quad \langle x := \theta \rangle \phi(x) \leftrightarrow \phi(\theta)$$

$$(\langle'\rangle) \quad \langle x' = \theta \rangle \phi \leftrightarrow \exists t \geq 0 \langle x := y(t) \rangle \phi \quad (y'(t) = \theta)$$

$$(\langle?\rangle) \quad \langle ?H \rangle \phi \leftrightarrow (H \wedge \phi)$$

$$(\langle\cup\rangle) \quad \langle \alpha \cup \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi$$

$$(\langle;\rangle) \quad \langle \alpha; \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \langle \beta \rangle \phi$$

$$(\langle*\rangle) \quad \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi \rightarrow \langle \alpha^* \rangle \phi$$

$$(\langle^d\rangle) \quad \langle \alpha^d \rangle \phi \leftrightarrow \neg\langle \alpha \rangle \neg\phi$$

$$(M) \quad \frac{\phi \rightarrow \psi}{\langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \psi}$$

$$(FP) \quad \frac{\phi \vee \langle \alpha \rangle \psi \rightarrow \psi}{\langle \alpha^* \rangle \phi \rightarrow \psi}$$

$$(ind) \quad \frac{\phi \rightarrow [\alpha]\phi}{\phi \rightarrow [\alpha^*]\phi}$$

The determinacy axiom $[\cdot]$ describes the duality of winning strategies for complementary winning conditions of Angel and Demon, i.e. that Demon has a winning strategy to achieve ϕ in hybrid game α if and only if Angel does not have a counter strategy, i.e. winning strategy to achieve $\neg\phi$ in the same game α . The determinacy axiom $[\cdot]$ internalizes Theorem 4. Axiom $\langle:=\rangle$ is Hoare's assignment rule. Formula $\phi(\theta)$ is obtained from $\phi(x)$ by *substituting* θ for x at all occurrences of x , provided x does not occur in the scope of a quantifier or modality binding x or a variable of θ . A modality containing $x :=$ or x' outside the scope of tests $?H$ or evolution domain constraints *binds* x , because it may change the value of x . In the differential equation axiom $\langle'\rangle$, $y(\cdot)$ is the unique [Wal98, Theorem 10.VI] solution of the symbolic initial value problem $y'(t) = \theta, y(0) = x$. The duration t how long to follow solution y is for Angel to decide, hence existentially quantified. It goes without saying that variables like t are fresh in Fig. 9.

Axioms $\langle?\rangle$, $\langle\cup\rangle$, and $\langle;\rangle$ are as in differential dynamic logic [Pla12] except that their meaning is quite different, because they refer to winning strategies of hybrid games instead of reachability relations of systems. The challenge axiom $\langle?\rangle$ expresses that Angel

has a winning strategy to achieve ϕ in the test game $?H$ exactly from those positions that are already in ϕ (because $?H$ does not change the state) and that satisfy H for otherwise she would fail the test and lose the game immediately. The axiom of choice $\langle \cup \rangle$ expresses that Angel has a winning strategy in a game of choice $\alpha \cup \beta$ to achieve ϕ iff she has a winning strategy in either hybrid game α or in β , because she can choose which one to play. The sequential game axiom $\langle ; \rangle$ expresses that Angel has a winning strategy in a sequential game $\alpha; \beta$ to achieve ϕ iff she has a winning strategy in game α to achieve $\langle \beta \rangle \phi$, i.e. to get to a position from which she has a winning strategy in game β to achieve ϕ . The iteration axiom $\langle * \rangle$ characterizes $\langle \alpha^* \rangle \phi$ as a pre-fixpoint. It expresses that, if the game is already in a state satisfying ϕ or if Angel has a winning strategy for game α to achieve $\langle \alpha^* \rangle \phi$, i.e. to get to a position from which she has a winning strategy for game α^* to achieve ϕ , then, either way, Angel has a winning strategy to achieve ϕ in game α^* . The converse of $\langle * \rangle$ can be derived² and is also denoted by $\langle * \rangle$. The dual axiom $\langle ^d \rangle$ characterizes dual games. It says that Angel has a winning strategy to achieve ϕ in dual game α^d iff Angel does not have a winning strategy to achieve $\neg \phi$ in game α . Combining dual game axiom $\langle ^d \rangle$ with the determinacy axiom $[\cdot]$ yields $\langle \alpha^d \rangle \phi \leftrightarrow [\alpha] \phi$, i.e. that Angel has a winning strategy to achieve ϕ in α^d iff Demon has a winning strategy to achieve ϕ in α . Similar reasoning derives $[\alpha^d] \phi \leftrightarrow \langle \alpha \rangle \phi$.

Monotonicity rule **M** is the generalization rule of monotonic modal logic **C** [Che80] and internalizes Lemma 3. It expresses that, if the implication $\phi \rightarrow \psi$ is valid, then, from wherever Angel has a winning strategy in a hybrid game α to achieve ϕ , she also has a winning strategy to achieve ψ , because ψ holds wherever ϕ does. So rule **M** expresses that easier objectives are easier to win. Fixpoint rule **FP** characterizes $\langle \alpha^* \rangle \phi$ as a *least* pre-fixpoint. It says that, if ψ is another formula that is a pre-fixpoint, i.e. that holds in all states that satisfy ϕ or from which Angel has a winning strategy in game α to achieve that condition ψ , then ψ also holds wherever $\langle \alpha^* \rangle \phi$ does, i.e. in all states from which Angel has a winning strategy in game α^* to achieve ϕ .

The proof rules **FP** and the induction rule **ind** are equivalent in the sense that one can be derived from the other in the dGCL calculus [Pla13, Pla14].

Example 5. The dual filibuster game formula from Lecture 20 proves easily in the dGCL calculus by going back and forth between players [Pla13] using the abbreviations \cap, \times :

$$\begin{array}{c}
 \mathbb{R} \frac{*}{x = 0 \vdash 0 = 0 \vee 1 = 0} \\
 \langle := \rangle \frac{x = 0 \vdash 0 = 0 \vee 1 = 0}{x = 0 \vdash \langle x := 0 \rangle x = 0 \vee \langle x := 1 \rangle x = 0} \\
 \langle \cup \rangle \frac{x = 0 \vdash \langle x := 0 \rangle x = 0 \vee \langle x := 1 \rangle x = 0}{x = 0 \vdash \langle x := 0 \cup x := 1 \rangle x = 0} \\
 \langle ^d \rangle \frac{x = 0 \vdash \langle x := 0 \cup x := 1 \rangle x = 0}{x = 0 \vdash \neg \langle x := 0 \cap x := 1 \rangle \neg x = 0} \\
 [\cdot] \frac{x = 0 \vdash \neg \langle x := 0 \cap x := 1 \rangle \neg x = 0}{x = 0 \vdash [x := 0 \cap x := 1] x = 0} \\
 \text{ind} \frac{x = 0 \vdash [x := 0 \cap x := 1] x = 0}{x = 0 \vdash [(x := 0 \cap x := 1)^*] x = 0} \\
 \langle ^d \rangle \frac{x = 0 \vdash [(x := 0 \cap x := 1)^*] x = 0}{x = 0 \vdash \langle (x := 0 \cup x := 1)^\times \rangle x = 0}
 \end{array}$$

² $\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi \rightarrow \langle \alpha^* \rangle \phi$ derives by $\langle * \rangle$. Thus, $\langle \alpha \rangle (\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi) \rightarrow \langle \alpha \rangle \langle \alpha^* \rangle \phi$ by **M**. Hence, $\phi \vee \langle \alpha \rangle (\phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi) \rightarrow \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi$ by propositional congruence. Consequently, $\langle \alpha^* \rangle \phi \rightarrow \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi$ by **FP**.

Theorem 6 (Soundness [Pla13, Pla14]). *The dGL proof calculus in Fig. 9 is sound, i.e. all provable formulas are valid.*

Proof. The full proof can be found in [Pla13, Pla14]. We just consider a few cases to exemplify the fundamentally more general semantics of hybrid games arguments compared to hybrid systems arguments. To prove soundness of an equivalence axiom $\phi \leftrightarrow \psi$, show $\llbracket \phi \rrbracket^I = \llbracket \psi \rrbracket^I$ for all interpretations I with any set of states \mathcal{S} .

$$\langle \cup \rangle \quad \llbracket \langle \alpha \cup \beta \rangle \phi \rrbracket^I = \varsigma_{\alpha \cup \beta}(\llbracket \phi \rrbracket^I) = \varsigma_{\alpha}(\llbracket \phi \rrbracket^I) \cup \varsigma_{\beta}(\llbracket \phi \rrbracket^I) = \llbracket \langle \alpha \rangle \phi \rrbracket^I \cup \llbracket \langle \beta \rangle \phi \rrbracket^I = \llbracket \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi \rrbracket^I$$

$$\langle ; \rangle \quad \llbracket \langle \alpha ; \beta \rangle \phi \rrbracket^I = \varsigma_{\alpha; \beta}(\llbracket \phi \rrbracket^I) = \varsigma_{\alpha}(\varsigma_{\beta}(\llbracket \phi \rrbracket^I)) = \varsigma_{\alpha}(\llbracket \langle \beta \rangle \phi \rrbracket^I) = \llbracket \langle \alpha \rangle \langle \beta \rangle \phi \rrbracket^I.$$

$$\langle ? \rangle \quad \llbracket \langle ?H \rangle \phi \rrbracket^I = \varsigma_{?H}(\llbracket \phi \rrbracket^I) = \llbracket H \rrbracket^I \cap \llbracket \phi \rrbracket^I = \llbracket H \wedge \phi \rrbracket^I$$

$\llbracket \cdot \rrbracket$ is sound by Theorem 4.

M Assume the premise $\phi \rightarrow \psi$ is valid in interpretation I , i.e. $\llbracket \phi \rrbracket^I \subseteq \llbracket \psi \rrbracket^I$. Then the conclusion $\langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \psi$ is valid in I , i.e. $\llbracket \langle \alpha \rangle \phi \rrbracket^I = \varsigma_{\alpha}(\llbracket \phi \rrbracket^I) \subseteq \varsigma_{\alpha}(\llbracket \psi \rrbracket^I) = \llbracket \langle \alpha \rangle \psi \rrbracket^I$ by monotonicity (Lemma 3).

FP Assume the premise $\phi \vee \langle \alpha \rangle \psi \rightarrow \psi$ is valid in I , i.e. $\llbracket \phi \vee \langle \alpha \rangle \psi \rrbracket^I \subseteq \llbracket \psi \rrbracket^I$. That is, $\llbracket \phi \rrbracket^I \cup \varsigma_{\alpha}(\llbracket \psi \rrbracket^I) = \llbracket \phi \rrbracket^I \cup \llbracket \langle \alpha \rangle \psi \rrbracket^I = \llbracket \phi \vee \langle \alpha \rangle \psi \rrbracket^I \subseteq \llbracket \psi \rrbracket^I$. Thus, ψ is a pre-fixpoint of $Z = \llbracket \phi \rrbracket^I \cup \varsigma_{\alpha}(Z)$. Now using Lemma 3, $\llbracket \langle \alpha^* \rangle \phi \rrbracket^I = \varsigma_{\alpha^*}(\llbracket \phi \rrbracket^I)$ is the least fixpoint and the least pre-fixpoint [Koz06, Appendix A] of $Z = \llbracket \phi \rrbracket^I \cup \varsigma_{\alpha}(Z)$. Since $\llbracket \langle \alpha^* \rangle \phi \rrbracket^I$ is the least pre-fixpoint and $\llbracket \psi \rrbracket^I$ is a pre-fixpoint, $\llbracket \langle \alpha^* \rangle \phi \rrbracket^I \subseteq \llbracket \psi \rrbracket^I$ holds, which implies that $\langle \alpha^* \rangle \phi \rightarrow \psi$ is valid in I . \square

So that is quite wonderful. This gives us a sound proof approach for CPSs that are as challenging as hybrid games. Now what exactly did we prove the axioms sound for? What does sound mean and entail exactly?

Note 11 (The miracle of soundness). *Soundness of the dGL proof calculus means that all dGL formulas that are provable using the dGL calculus are valid. A condition sine qua non for logic, i.e. a condition without which logic could not be. It would not make sense to prove a formula if that would not even entail its validity.*

For a proof calculus to be sound, every formula that it proves with any proof has to be valid. Fortunately, proofs are composed by proof rules from axioms. So all we need to do to ensure that a proof calculus is sound is to prove the few axioms to be sound and then everything we ever derive from them by sound proof rules is correct as well, no matter how big and complicated. A proof is a long combination of many simple arguments, each of which just involve one of the axioms or proof rules. Once each of those finitely many axioms and proof rules are proved to be sound, all those infinitely many proofs that can be conducted in the dGL proof calculus become sound as well. That is compositionality in its finest form for the soundness argument. It is soundness that ultimately links semantics and axiomatics into perfect unison^a so that axiomatic proof analysis coincides with semantic truth, an important aspect of the logical trinity.

One minor subtlety is that a proof could use many possible instances of the same finite axiom list, so that the soundness proof for the axioms has to work for any instance. This aspect is often left implicit in soundness arguments, although a rigorous treatment can be given by distinguishing axioms from axiom schemes [Pla13, Pla14].

^aIn search of perfection, completeness is another important aspect in achieving perfect unison, which, incidentally, holds for differential game logic as well [Pla13, Pla14]

6 Relating Differential Game Logic and Differential Dynamic Logic

Now that we have come to appreciate the value of soundness, couldn't we have known about that, for the most part, before Theorem 6? Most dGL axioms look rather familiar, except for $\langle \cdot \rangle$ versus $[\cdot]$ dualities, when we compare them to the dL axioms from [Lecture 5 on Dynamical Systems and Dynamic Axioms](#). Does that not mean that these same axioms are already trivially sound? Why did we go through the (rather minor) trouble of proving Theorem 6?

Before you read on, see if you can find the answer for yourself.

It is not quite so easy. After all, we could have given the same syntactical operator \cup an entirely different meaning for hybrid games than before for hybrid systems. Maybe we could have been silly and flipped the meaning of $;$ and \cup around. The fact of the matter is, of course, that we did not. The operator \cup still means choice, just for hybrid games rather than hybrid systems. So could we deduce the soundness of the dGL axioms in Fig. 9 from the soundness of the corresponding dL axioms from [Lecture 5 on Dynamical Systems and Dynamic Axioms](#) and focus on the new axioms, only?

Before we do anything of the kind, we first need to convince ourselves that the dL semantics really coincides with the more general dGL semantics in case there are no games involved. How could that be done? Maybe by proving validity of all formulas of the following form

$$\underbrace{\langle \alpha \rangle \phi}_{\text{in dL}} \leftrightarrow \underbrace{\langle \alpha \rangle \phi}_{\text{in dGL}} \quad (13)$$

for dual-free hybrid games α , i.e. those that do not mention d (not even indirectly hidden in the abbreviation $\cap, ^\times$).

Before you read on, see if you can find the answer for yourself.

The problem with (13) is that it is not directly a formula in any logic, because the \leftrightarrow operator could hardly be applied meaningfully to two formulas from different logics. Well, of course, every $\text{d}\mathcal{L}$ formula is a $\text{dG}\mathcal{L}$ formula, so the left-hand side of (13) could be embedded into $\text{dG}\mathcal{L}$, but then (13) becomes well-defined but is only stating a mere triviality.

Instead, a proper approach would be to rephrase the well-intended (13) semantically:

$$\underbrace{\nu \models \langle \alpha \rangle \phi}_{\text{in } \text{d}\mathcal{L}} \text{ iff } \underbrace{\nu \in \llbracket \langle \alpha \rangle \phi \rrbracket^I}_{\text{in } \text{dG}\mathcal{L}} \quad (14)$$

which is equivalent to

$$\underbrace{(\omega \models \phi \text{ for some } \omega \text{ with } (\nu, \omega) \in \rho(\alpha))}_{\text{statement about reachability in } \text{d}\mathcal{L}} \text{ iff } \underbrace{\nu \in \varsigma_\alpha(\llbracket \phi \rrbracket^I)}_{\text{winning in } \text{dG}\mathcal{L}}$$

Equivalence (14) can be shown. In fact, an exercise in [Lecture 3 on Choice & Control](#) already developed an understanding of the $\text{d}\mathcal{L}$ semantics based on sets of states, preparing for (14).

The trouble is that, besides requiring a proof itself, the equivalence (14) will still not quite justify soundness of the $\text{dG}\mathcal{L}$ axioms in Fig. 9 that look innocuously like $\text{d}\mathcal{L}$ axioms. Equivalence (14) is for dual-free hybrid games α . But even if the top-level operator in axiom $\langle \cup \rangle$ is not d , that dual operator could still occur within α or β , which requires a game semantics to make sense of.

Consequently, we are better off proving soundness for the $\text{dG}\mathcal{L}$ axioms according to their actual semantics, like in Theorem 6, as opposed to trying half-witted ways out that only make soundness matters worse.

Exercises

Exercise 1. Explain how often you will have to repeat the winning region construction to show that the following $\text{dG}\mathcal{L}$ formula is valid:

$$\langle (x := x + 1; x' = 1^d \cup x := x - 1)^* \rangle (0 \leq x < 1)$$

Exercise 2. Can you find $\text{dG}\mathcal{L}$ formulas for which the winning region construction takes even longer to terminate? How far can you push this?

Exercise 3. Carefully identify how determinacy relates to the two possible understandings of the filibuster example discussed in an earlier lecture.

Exercise 4. Prove the elided cases of Lemma 3.

Exercise 5. Find the appropriate soundness notion for the axioms of $\text{dG}\mathcal{L}$ and prove that the axioms are sound.

Exercise 6. Write down a valid formula that characterizes an interesting game between two robots.

References

- [Che80] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge Univ. Press, 1980.
- [HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT Press, 2000.
- [Koz06] Dexter Kozen. *Theory of Computation*. Springer, 2006.
- [Pla12] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012. [doi:10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla13] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.
- [Pla14] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014. [arXiv:1408.1980](https://arxiv.org/abs/1408.1980).
- [Wal98] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1998.

Lecture Notes on Game Proofs & Separations

[André Platzer](#)

Carnegie Mellon University
Lecture 23

1 Introduction

This lecture continues the study of hybrid games and their logic, differential game logic [Pla13]. [Lecture 20 on Hybrid Systems & Games](#) introduced hybrid games, [Lecture 21 on Winning Strategies & Regions](#) studied the winning region semantics, and [Lecture 22 on Winning & Proving Hybrid Games](#) identified the winning region semantics for loops in hybrid games as well as a study of the axioms of hybrid games.

These lecture notes are based on [Pla13], where more information can be found on logic and hybrid games.

2 Recall: Semantics of Hybrid Games

Recall the semantics of hybrid games and two results from [Lecture 22 on Winning & Proving Hybrid Games](#).

Definition 1 (Semantics of hybrid games). The *semantics of a hybrid game* α is a function $\varsigma_\alpha(\cdot)$ that, for each interpretation I and each set of Angel's winning states $X \subseteq \mathcal{S}$, gives the *winning region*, i.e. the set of states $\varsigma_\alpha(X)$ from which Angel has a winning strategy to achieve X (whatever strategy Demon chooses). It is defined inductively as follows^a

1. $\varsigma_{x:=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
2. $\varsigma_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for some } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
3. $\varsigma_{?H}(X) = \llbracket H \rrbracket^I \cap X$
4. $\varsigma_{\alpha \cup \beta}(X) = \varsigma_\alpha(X) \cup \varsigma_\beta(X)$
5. $\varsigma_{\alpha; \beta}(X) = \varsigma_\alpha(\varsigma_\beta(X))$
6. $\varsigma_{\alpha^*}(X) = \bigcap \{Z \subseteq \mathcal{S} : X \cup \varsigma_\alpha(Z) \subseteq Z\}$
7. $\varsigma_{\alpha^d}(X) = (\varsigma_\alpha(X^{\mathbb{C}}))^{\mathbb{C}}$

The *winning region* of Demon, i.e. the set of states $\delta_\alpha(X)$ from which Demon has a winning strategy to achieve X (whatever strategy Angel chooses) is defined inductively as follows

1. $\delta_{x:=\theta}(X) = \{\nu \in \mathcal{S} : \nu_x^{\llbracket \theta \rrbracket} \in X\}$
2. $\delta_{x'=\theta \& H}(X) = \{\varphi(0) \in \mathcal{S} : \varphi(r) \in X \text{ for all } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0, r] \rightarrow \mathcal{S} \text{ such that } \varphi(\zeta) \in \llbracket H \rrbracket^I \text{ and } \frac{d\varphi(t)(x)}{dt}(\zeta) = \llbracket \theta \rrbracket_{\varphi(\zeta)} \text{ for all } 0 \leq \zeta \leq r\}$
3. $\delta_{?H}(X) = (\llbracket H \rrbracket^I)^{\mathbb{C}} \cup X$
4. $\delta_{\alpha \cup \beta}(X) = \delta_\alpha(X) \cap \delta_\beta(X)$
5. $\delta_{\alpha; \beta}(X) = \delta_\alpha(\delta_\beta(X))$
6. $\delta_{\alpha^*}(X) = \bigcup \{Z \subseteq \mathcal{S} : Z \subseteq X \cap \delta_\alpha(Z)\}$
7. $\delta_{\alpha^d}(X) = (\delta_\alpha(X^{\mathbb{C}}))^{\mathbb{C}}$

^a The semantics of a hybrid game is not merely a reachability relation between states as for hybrid systems [Pla12], because the adversarial dynamic interactions and nested choices of the players have to be taken into account.

Lemma 2 (Monotonicity [Pla13]). *The semantics is monotone, i.e. $\varsigma_\alpha(X) \subseteq \varsigma_\alpha(Y)$ and $\delta_\alpha(X) \subseteq \delta_\alpha(Y)$ for all $X \subseteq Y$.*

Theorem 3 (Consistency & determinacy [Pla13]). *Hybrid games are consistent and determined, i.e. $\models \neg\langle\alpha\rangle\neg\phi \leftrightarrow [\alpha]\phi$.*

3 Hybrid Game Proofs

An axiomatization for differential game logic has been found in previous work [Pla13], where we refer to for more details.

Note 4 (Differential game logic axiomatization [Pla13]).

$$([\cdot]) \quad [\alpha]\phi \leftrightarrow \neg\langle\alpha\rangle\neg\phi$$

$$(\langle:=\rangle) \quad \langle x := \theta \rangle \phi(x) \leftrightarrow \phi(\theta)$$

$$(\langle'\rangle) \quad \langle x' = \theta \rangle \phi \leftrightarrow \exists t \geq 0 \langle x := y(t) \rangle \phi \quad (y'(t) = \theta)$$

$$(\langle?\rangle) \quad \langle ?H \rangle \phi \leftrightarrow (H \wedge \phi)$$

$$(\langle\cup\rangle) \quad \langle \alpha \cup \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi$$

$$(\langle;\rangle) \quad \langle \alpha; \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \langle \beta \rangle \phi$$

$$(\langle*\rangle) \quad \phi \vee \langle \alpha \rangle \langle \alpha^* \rangle \phi \rightarrow \langle \alpha^* \rangle \phi$$

$$(\langle^d\rangle) \quad \langle \alpha^d \rangle \phi \leftrightarrow \neg\langle \alpha \rangle \neg\phi$$

$$(M) \quad \frac{\phi \rightarrow \psi}{\langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \psi}$$

$$(FP) \quad \frac{\phi \vee \langle \alpha \rangle \psi \rightarrow \psi}{\langle \alpha^* \rangle \phi \rightarrow \psi}$$

$$(ind) \quad \frac{\phi \rightarrow [\alpha]\phi}{\phi \rightarrow [\alpha^*]\phi}$$

The proof rules **FP** and **ind** are equivalent in the sense that one can be derived from the other in the dGL calculus [Pla13].

Example 4. The dual filibuster game formula from [Lecture 20](#) proves easily in the dGL

calculus by going back and forth between players [Pla13]:

$$\begin{array}{c}
 \mathbb{R} \frac{*}{x = 0 \rightarrow 0 = 0 \vee 1 = 0} \\
 \langle := \rangle \frac{x = 0 \rightarrow \langle x := 0 \rangle x = 0 \vee \langle x := 1 \rangle x = 0}{x = 0 \rightarrow \langle x := 0 \cup x := 1 \rangle x = 0} \\
 \langle \cup \rangle \frac{x = 0 \rightarrow \langle x := 0 \cup x := 1 \rangle x = 0}{x = 0 \rightarrow \neg \langle x := 0 \cap x := 1 \rangle \neg x = 0} \\
 \langle ^d \rangle \frac{x = 0 \rightarrow \neg \langle x := 0 \cap x := 1 \rangle \neg x = 0}{x = 0 \rightarrow [x := 0 \cap x := 1] x = 0} \\
 [\cdot] \frac{x = 0 \rightarrow [x := 0 \cap x := 1] x = 0}{x = 0 \rightarrow [(x := 0 \cap x := 1)^*] x = 0} \\
 \text{ind} \frac{x = 0 \rightarrow [(x := 0 \cap x := 1)^*] x = 0}{x = 0 \rightarrow \langle (x := 0 \cup x := 1)^\times \rangle x = 0} \\
 \langle ^d \rangle \frac{x = 0 \rightarrow \langle (x := 0 \cup x := 1)^\times \rangle x = 0}{x = 0 \rightarrow \langle (x := 0 \cup x := 1)^\times \rangle x = 0}
 \end{array}$$

4 Soundness

Theorem 5 (Soundness [Pla13]). *The dGL proof calculus in Fig. 4 is sound, i.e. all provable formulas are valid.*

Proof. The full proof can be found in [Pla13]. We just consider a few cases to exemplify the fundamentally more general semantics of hybrid games arguments compared to hybrid systems arguments. To prove soundness of an equivalence axiom $\phi \leftrightarrow \psi$, show $\llbracket \phi \rrbracket^I = \llbracket \psi \rrbracket^I$ for all interpretations I with any set of states \mathcal{S} .

$$\langle \cup \rangle \llbracket \langle \alpha \cup \beta \rangle \phi \rrbracket^I = \varsigma_{\alpha \cup \beta}(\llbracket \phi \rrbracket^I) = \varsigma_\alpha(\llbracket \phi \rrbracket^I) \cup \varsigma_\beta(\llbracket \phi \rrbracket^I) = \llbracket \langle \alpha \rangle \phi \rrbracket^I \cup \llbracket \langle \beta \rangle \phi \rrbracket^I = \llbracket \langle \alpha \rangle \phi \vee \langle \beta \rangle \phi \rrbracket^I$$

$$\langle ; \rangle \llbracket \langle \alpha ; \beta \rangle \phi \rrbracket^I = \varsigma_{\alpha ; \beta}(\llbracket \phi \rrbracket^I) = \varsigma_\alpha(\varsigma_\beta(\llbracket \phi \rrbracket^I)) = \varsigma_\alpha(\llbracket \langle \beta \rangle \phi \rrbracket^I) = \llbracket \langle \alpha \rangle \langle \beta \rangle \phi \rrbracket^I.$$

$$\langle ? \rangle \llbracket \langle ?H \rangle \phi \rrbracket^I = \varsigma_{?H}(\llbracket \phi \rrbracket^I) = \llbracket H \rrbracket^I \cap \llbracket \phi \rrbracket^I = \llbracket H \wedge \phi \rrbracket^I$$

$[\cdot]$ is sound by Theorem 3.

M Assume the premise $\phi \rightarrow \psi$ is valid in interpretation I , i.e. $\llbracket \phi \rrbracket^I \subseteq \llbracket \psi \rrbracket^I$. Then the conclusion $\langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \psi$ is valid in I , i.e. $\llbracket \langle \alpha \rangle \phi \rrbracket^I = \varsigma_\alpha(\llbracket \phi \rrbracket^I) \subseteq \varsigma_\alpha(\llbracket \psi \rrbracket^I) = \llbracket \langle \alpha \rangle \psi \rrbracket^I$ by monotonicity (Lemma 2). \square

5 Separating Axioms

The axioms of differential game logic in Fig. 4 are sound for hybrid systems as well, because every hybrid system is a (single player) hybrid game. With a few exceptions, they look surprisingly close to the axioms for hybrid systems from Lecture 5. In order to understand the fundamental difference between hybrid systems and hybrid games, it is instructive to also investigate separating axioms, i.e. axioms of hybrid systems that are not sound for hybrid games. Some of these are summarized in Fig. 1, referring to [Pla13] for details.

\mathcal{K} $[\alpha](\phi \rightarrow \psi) \rightarrow ([\alpha]\phi \rightarrow [\alpha]\psi)$	\mathcal{M} $\langle \alpha \rangle \phi \vee \langle \alpha \rangle \psi \rightarrow \langle \alpha \rangle (\phi \vee \psi)$
\mathcal{G} $\frac{\phi}{[\alpha]\phi}$	$\mathcal{M}_{[\cdot]}$ $\frac{\phi \rightarrow \psi}{[\beta]\phi \rightarrow [\beta]\psi}$
\mathcal{R} $\frac{\phi_1 \wedge \phi_2 \rightarrow \psi}{[\alpha]\phi_1 \wedge [\alpha]\phi_2 \rightarrow [\alpha]\psi}$	
\mathcal{B} $\langle \alpha \rangle \exists x \phi \rightarrow \exists x \langle \alpha \rangle \phi \quad (x \notin \alpha)$	$\overleftarrow{\mathcal{B}}$ $\exists x \langle \alpha \rangle \phi \rightarrow \langle \alpha \rangle \exists x \phi \quad (x \notin \alpha)$
\mathcal{X} $[\alpha^*](\phi \rightarrow [\alpha]\phi) \rightarrow (\phi \rightarrow [\alpha^*]\phi)$	
\mathcal{EA} $\langle \alpha^* \rangle \phi \rightarrow \phi \vee \langle \alpha^* \rangle (\neg \phi \wedge \langle \alpha \rangle \phi)$	

Figure 1: Separating axioms: The axioms and rules on the left are sound for hybrid systems but not for hybrid games. The related axioms on the right are sound for hybrid games.

6 Repetitive Diamonds – Convergence vs. Iteration

More fundamental differences between hybrid systems and hybrid games also exist in terms of convergence rules, even if these have played a less prominent role in this course so far. These differences are discussed in detail elsewhere [Pla13]. In a nutshell, Harel’s convergence rule [HMP77] is not a separating axiom, because it is sound for dGL, just unnecessary, and, furthermore, not even particularly useful for hybrid games [Pla13]. The hybrid version of Harel’s convergence rule [Pla08] for dL reads as follows (it assumes that v does not occur in α):

$$(\text{con}) \frac{\varphi(v+1) \wedge v+1 > 0 \vdash \langle \alpha \rangle \varphi(v)}{\Gamma, \exists v \varphi(v) \vdash \langle \alpha^* \rangle \exists v \leq 0 \varphi(v), \Delta}$$

The dL proof rule **con** expresses that the variant $\varphi(v)$ holds for some real number $v \leq 0$ after repeating α sufficiently often if $\varphi(v)$ holds for some real number at all in the beginning (antecedent) and, by premise, $\varphi(v)$ can decrease after some execution of α by 1 (or another positive real constant) if $v > 0$. This rule can be used to show positive progress (by 1) with respect to $\varphi(v)$ by executing α . Just like the induction rule **ind** is often used with a separate premiss for the initial and postcondition check (*ind'* from [Lecture 7 on Loops & Invariants](#)), rule **con** is often used in the following derived form:

$$(\text{con}') \frac{\Gamma \vdash \exists v \varphi(v), \Delta \quad \forall v > 0 (\varphi(v) \rightarrow \langle \alpha \rangle \varphi(v-1)) \quad \exists v \leq 0 \varphi(v) \vdash \psi}{\Gamma \vdash \langle \alpha^* \rangle \psi, \Delta}$$

The following sequent proof shows how convergence rule *con'* can be used to prove a simple dL liveness property of a hybrid program:

$$\begin{array}{c}
 \text{con}' \frac{\mathbb{R} \frac{*}{x \geq 0 \vdash \exists n x < n+1} \quad \langle := \rangle \frac{\mathbb{R} \frac{*}{x < n+2 \wedge n+1 > 0 \vdash x-1 < n+1}}{x < n+2 \wedge n+1 > 0 \vdash \langle x := x-1 \rangle x < n+1} \quad \mathbb{R} \frac{*}{\exists n \leq 0 x < n+1 \vdash x < 1}}{x \geq 0 \vdash \langle (x := x-1)^* \rangle x < 1} \\
 \hline
 \rightarrow r \frac{}{x \geq 0 \rightarrow \langle (x := x-1)^* \rangle x < 1}
 \end{array}$$

Let's compare how dGL proves diamond properties of repetitions based on the iteration axiom $\langle * \rangle$.

Example 6 (Non-game system). The same simple non-game dGL formula

$$x \geq 0 \rightarrow \langle (x := x-1)^* \rangle 0 \leq x < 1$$

as above is provable without *con'*, as shown in Fig. 2, where $\langle \alpha^* \rangle 0 \leq x < 1$ is short for $\langle (x := x-1)^* \rangle (0 \leq x < 1)$. Note that, as in many subsequent proofs, the extra

$$\begin{array}{c}
 \mathbb{R} \frac{}{\forall x (0 \leq x < 1 \vee p(x-1) \rightarrow p(x)) \rightarrow (x \geq 0 \rightarrow p(x))} \\
 \langle := \rangle \frac{}{\forall x (0 \leq x < 1 \vee \langle x := x-1 \rangle p(x) \rightarrow p(x)) \rightarrow (x \geq 0 \rightarrow p(x))} \\
 \text{US} \frac{}{\forall x (0 \leq x < 1 \vee \langle x := x-1 \rangle \langle \alpha^* \rangle 0 \leq x < 1 \rightarrow \langle \alpha^* \rangle 0 \leq x < 1) \rightarrow (x \geq 0 \rightarrow \langle \alpha^* \rangle 0 \leq x < 1)} \\
 \langle * \rangle, \forall, \text{MP} \frac{}{x \geq 0 \rightarrow \langle \alpha^* \rangle 0 \leq x < 1}
 \end{array}$$

Figure 2: dGL Angel proof for non-game system Example 6

$$x \geq 0 \rightarrow \langle (x := x-1)^* \rangle 0 \leq x < 1$$

assumption for MP near the bottom of the proof in Fig. 2 is provable easily using $\langle * \rangle, \forall$:

$$\begin{array}{c}
 * \\
 \langle * \rangle \frac{}{0 \leq x < 1 \vee \langle x := x-1 \rangle \langle \alpha^* \rangle 0 \leq x < 1 \rightarrow \langle \alpha^* \rangle 0 \leq x < 1} \\
 \forall r \frac{}{\forall x (0 \leq x < 1 \vee \langle x := x-1 \rangle \langle \alpha^* \rangle 0 \leq x < 1 \rightarrow \langle \alpha^* \rangle 0 \leq x < 1)}
 \end{array}$$

Example 7 (Choice game). The dGL formula

$$x = 1 \wedge a = 1 \rightarrow \langle (x := a; a := 0 \cap x := 0)^* \rangle x \neq 1$$

is provable as shown in Fig. 3, where $\beta \cap \gamma$ is short for $x := a; a := 0 \cap x := 0$ and $\langle (\beta \cap \gamma)^* \rangle x \neq 1$ short for $\langle (x := a; a := 0 \cap x := 0)^* \rangle x \neq 1$:

Example 8 (2-Nim-type game). The dGL formula

$$x \geq 0 \rightarrow \langle (x := x-1 \cap x := x-2)^* \rangle 0 \leq x < 2$$

is provable as shown in Fig. 3, where $\beta \cap \gamma$ is short for $x := x-1 \cap x := x-2$ and $\langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2$ short for $\langle (x := x-1 \cap x := x-2)^* \rangle 0 \leq x < 2$:

	*
\mathbb{R}	$\forall x (x \neq 1 \vee p(a, 0) \wedge p(0, a) \rightarrow p(x, a)) \rightarrow (true \rightarrow p(x, a))$
$\langle ; \rangle, \langle := \rangle$	$\forall x (x \neq 1 \vee \langle \beta \rangle p(x, a) \wedge \langle \gamma \rangle p(x, a) \rightarrow p(x, a)) \rightarrow (true \rightarrow p(x, a))$
$\langle \cup \rangle, \langle d \rangle$	$\forall x (x \neq 1 \vee \langle \beta \cap \gamma \rangle p(x, a) \rightarrow p(x, a)) \rightarrow (true \rightarrow p(x, a))$
US	$\forall x (x \neq 1 \vee \langle \beta \cap \gamma \rangle \langle (\beta \cap \gamma)^* \rangle x \neq 1 \rightarrow \langle (\beta \cap \gamma)^* \rangle x \neq 1) \rightarrow (true \rightarrow \langle (\beta \cap \gamma)^* \rangle x \neq 1)$
$\langle * \rangle, \forall, MP$	$true \rightarrow \langle (\beta \cap \gamma)^* \rangle x \neq 1$
\mathbb{R}	$x = 1 \wedge a = 1 \rightarrow \langle (\beta \cap \gamma)^* \rangle x \neq 1$

Figure 3: dGL Angel proof for choice game Example 7

$$x = 1 \wedge a = 1 \rightarrow \langle (x := a; a := 0 \cap x := 0)^* \rangle x \neq 1$$

	*
\mathbb{R}	$\forall x (0 \leq x < 2 \vee p(x - 1) \wedge p(x - 2) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle := \rangle$	$\forall x (0 \leq x < 2 \vee \langle \beta \rangle p(x) \wedge \langle \gamma \rangle p(x) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle \cup \rangle, \langle d \rangle$	$\forall x (0 \leq x < 2 \vee \langle \beta \cap \gamma \rangle p(x) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
US	$\forall x (0 \leq x < 2 \vee \langle \beta \cap \gamma \rangle \langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2 \rightarrow \langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2) \rightarrow (true \rightarrow \langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2)$
$\langle * \rangle, \forall, MP$	$true \rightarrow \langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2$
\mathbb{R}	$x \geq 0 \rightarrow \langle (\beta \cap \gamma)^* \rangle 0 \leq x < 2$

Figure 4: dGL Angel proof for 2-Nim-type game Example 8

$$x \geq 0 \rightarrow \langle (x := x - 1 \cap x := x - 2)^* \rangle 0 \leq x < 2$$

Example 9 (Hybrid game). The dGL formula

$$\langle (x := 1; x' = 1^d \cup x := x - 1)^* \rangle 0 \leq x < 1$$

is provable as shown in Fig. 5, where the notation $\langle (\beta \cup \gamma)^* \rangle 0 \leq x < 1$ is short for $\langle (x := 1; x' = 1^d \cup x := x - 1)^* \rangle (0 \leq x < 1)$: The proof steps for β use in $\langle ' \rangle$ that $t \mapsto x + t$

	*
\mathbb{R}	$\forall x (0 \leq x < 1 \vee \forall t \geq 0 p(1+t) \vee p(x-1) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle := \rangle$	$\forall x (0 \leq x < 1 \vee \langle x := 1 \rangle \neg \exists t \geq 0 \langle x := x + t \rangle \neg p(x) \vee p(x-1) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle ' \rangle$	$\forall x (0 \leq x < 1 \vee \langle x := 1 \rangle \neg \langle x' = 1 \rangle \neg p(x) \vee p(x-1) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle ; \rangle, \langle ' \rangle^d$	$\forall x (0 \leq x < 1 \vee \langle \beta \rangle p(x) \vee \langle \gamma \rangle p(x) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
$\langle \cup \rangle$	$\forall x (0 \leq x < 1 \vee \langle \beta \cup \gamma \rangle p(x) \rightarrow p(x)) \rightarrow (true \rightarrow p(x))$
US	$\forall x (0 \leq x < 1 \vee \langle \beta \cup \gamma \rangle \langle (\beta \cup \gamma)^* \rangle 0 \leq x < 1 \rightarrow \langle (\beta \cup \gamma)^* \rangle 0 \leq x < 1) \rightarrow (true \rightarrow \langle (\beta \cup \gamma)^* \rangle 0 \leq x < 1)$
$\langle * \rangle, \forall, MP$	$true \rightarrow \langle (\beta \cup \gamma)^* \rangle 0 \leq x < 1$

Figure 5: dGL Angel proof for hybrid game Example 9

$$\langle (x := 1; x' = 1^d \cup x := x - 1)^* \rangle 0 \leq x < 1$$

is the solution of the differential equation, so the subsequent use of $\langle := \rangle$ substitutes 1 in for x to obtain $t \mapsto 1 + t$. Recall from Lecture 22 that the winning regions for this formula need $>\omega$ iterations to converge. It is still provable easily.

7 There and Back Again Game

Quite unlike in hybrid systems and (poor test) differential dynamic logic [Pla08, Pla12], every hybrid game containing a differential equation $x' = \theta \& H$ with evolution domain constraints H can be replaced equivalently by a hybrid game without evolution domain constraints (even using poor tests, i.e. each test $?H$ uses only first-order formulas H). Evolution domains are definable in hybrid games and can, thus, be removed equivalently.

Lemma 10 (Domain reduction [Pla13, Pla12]). *Evolution domains of differential equations are definable as hybrid games: For every hybrid game there is an equivalent hybrid game that has no evolution domain constraints, i.e. all continuous evolutions are of the form $x' = \theta$.*

Proof. For notational convenience, assume the (vectorial) differential equation $x' = \theta(x)$ to contain a clock $x'_0 = 1$ and that t_0 and z are fresh variables. Then $x' = \theta(x) \& H(x)$ is equivalent to the hybrid game:

$$t_0 := x_0; x' = \theta(x); (z := x; z' = -\theta(z))^d; ?(z_0 \geq t_0 \rightarrow H(z)) \quad (1)$$

See Fig. 6 for an illustration. Suppose the current player is Angel. The idea behind

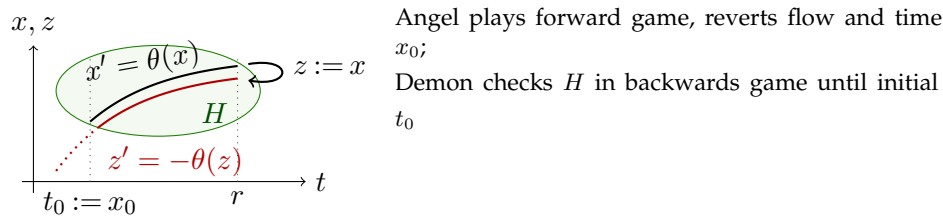


Figure 6: “There and back again game”: Angel evolves x forwards in time along $x' = \theta(x)$, Demon checks evolution domain backwards in time along $z' = -\theta(z)$ on a copy z of the state vector x

game equivalence (1) is that the fresh variable t_0 remembers the initial time x_0 , and Angel then evolves forward along $x' = \theta(x)$ for any amount of time (Angel’s choice). Afterwards, the opponent Demon copies the state x into a fresh variable (vector) z that he can evolve backwards along $(z' = -\theta(z))^d$ for any amount of time (Demon’s choice). The original player Angel must then pass the challenge $?(z_0 \geq t_0 \rightarrow H(z))$, i.e. Angel loses immediately if Demon was able to evolve backwards and leave region $H(z)$ while satisfying $z_0 \geq t_0$, which checks that Demon did not evolve backward for longer than Angel evolved forward. Otherwise, when Angel passes the test, the extra variables t_0, z become irrelevant (they are fresh) and the game continues from the current state x that Angel chose in the first place (by selecting a duration for the evolution that Demon could not invalidate). \square

Lemma 10 can eliminate all evolution domain constraints equivalently in hybrid games from now on. While evolution domain constraints are fundamental parts of standard hybrid systems [Hen96, HKPV95, ACHH92, Pla08], they turn out to be mere convenience notation for hybrid games. In that sense, hybrid games are more fundamental than hybrid systems, because they feature elementary operators.

Exercises

*Exercise 1 (**).* The following formula was proved using dGL’s hybrid games type proof rules in Fig. 2

$$x \geq 0 \rightarrow \langle (x := x - 1)^* \rangle 0 \leq x < 1$$

Try to prove it using the convergence rule *con'* instead.

References

- [ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P.

- Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.
- [Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, Los Alamitos, 1996. IEEE Computer Society. doi:[10.1109/LICS.1996.561342](https://doi.org/10.1109/LICS.1996.561342).
- [HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What’s decidable about hybrid automata? In Frank Thomson Leighton and Allan Borodin, editors, *STOC*, pages 373–382. ACM, 1995. doi:[10.1145/225058.225162](https://doi.org/10.1145/225058.225162).
- [HMP77] David Harel, Albert R. Meyer, and Vaughan R. Pratt. Computability and completeness in logics of programs (preliminary report). In *STOC*, pages 261–268. ACM, 1977.
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:[10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla12] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla13] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.

Lecture Notes on Logical Theory & Completeness

André Platzer

Carnegie Mellon University
Lecture 24

1. Introduction

This course has studied a number of logics, first-order logic FOL in [Lecture 2](#), differential dynamic logic $d\mathcal{L}$ [[Pla08](#), [Pla10b](#), [Pla12c](#), [Pla12b](#)] in [Lecture 3](#) and [Lecture 4](#) and following, differential temporal dynamic logic dTL [[Pla07](#), [Pla10b](#), Chapter 4] in [Lecture 16](#) and [17](#), as well as differential game logic dGL [[Pla13](#)] since [Lecture 22](#). There are other logics for cyber-physical systems that have not been included in this course, but share similar principles for further dynamical aspects. Such logics include quantified differential dynamic logic $Qd\mathcal{L}$ for distributed hybrid systems [[Pla10c](#), [Pla12a](#)], which are systems that are simultaneously distributed systems and hybrid systems, as well as stochastic differential dynamic logic $Sd\mathcal{L}$ for stochastic hybrid systems [[Pla11](#)], which simultaneously involve stochastic dynamics and hybrid dynamics. Logics play a stellar role not just in cyber-physical systems, but also many other contexts. Other important logics include propositional logic, restrictions of first-order logic to certain theories, such as first-order logic of real arithmetic [[Tar51](#)], and higher-order logic [[And02](#)]. But there are numerous other important and successful logics for many purposes, both general and specific.

In this lecture, we take a step back and study some common important concepts in understandings logics as the objects of study themselves. This study will necessarily be hopelessly incomplete for lack of time. But it should give you a flavor of important principles and concepts in logic that we have not already run across explicitly in earlier lectures of this course, even if many have been foreshadowed. We will also have the opportunity to apply these more general concepts to cyber-physical systems.

These lecture notes are based on [[Sch12](#), [Pla10b](#), [Pla08](#), [Pla12c](#), [Pla12b](#), [Pla10d](#), [Pla14](#)]. The most important learning goals of this lecture are:

Modeling and Control: This lecture culminates in a deep and surprising intimate relationship between discrete and continuous dynamics. This relationship provides a deeper understanding than ever before of the core principles behind cyber-physical systems, because it makes it possible to completely align and proof-theoretically equate discrete dynamics, continuous dynamics, and joint hybrid dynamics. This understanding has an effect on the significance of identifying the relevant dynamical aspects of cyber-physical systems by emphasizing the source of their simplicity: the individual parts of CPS models are easier than understanding the whole at once.

Computational Thinking: This lecture showcases exemplary computational thinking in action by exploiting proof-theoretical relationships to draw conclusions about cyber-physical systems and their analysis. It also practices the logical trinity consisting of the relationship of syntax, semantics, and axiomatics in more detail for the case of first-order logic. Finally, the lecture identifies logical parallels and differences of general first-order logic, interpreted first-order logic of real arithmetic, and differential dynamic logic.

CPS Skills: This lecture has only a minor impact on CPS skills in the form of shining a light of surprising intimacy on the relationship between discrete and continuous dynamics of CPS, a phenomenon that reemphasizes the possible liberties with the characterization of their dynamical aspects. By the complete alignment, one person's discrete dynamics may be another person's continuous dynamics and vice versa. This confirms the significant role that the design of proper models that are adequate and suitable for analysis plays in CPS verification and validation.

2. Soundness

The most important parts of a logic \mathcal{L} are the following. The logic \mathcal{L} defines what the *syntactically well-formed formulas* are. Every well-formed formula carries meaning, which the *semantics of formulas* in \mathcal{L} defines. The semantics defines a relation \models between sets of formulas and formulas, in which $\Phi \models \phi$ holds iff ϕ is a semantic consequence of the set of formulas Φ , i.e. ϕ is true (usually written $\nu \models \phi$) in every interpretation ν for which all formulas $\psi \in \Phi$ are true. The most important case for our purposes is the case $\Phi = \emptyset$ of validity, in which case $\models \phi$ holds iff ϕ is valid, i.e. true ($\nu \models \phi$) in all interpretations ν of \mathcal{L} . An interpretation ν in which ϕ is true (i.e. $\nu \models \phi$) is also called a *model* of ϕ .

For the case of first-order logic FOL, [Lecture 2](#) defined both their syntax and semantics. The syntax and semantics of differential dynamic logic $\text{d}\mathcal{L}$ has been defined in [Lecture 3](#) and [Lecture 4](#).

The syntax of a logic \mathcal{L} defines what we can write down that carries meaning. The semantics of a logic \mathcal{L} then defines what the meaning of the syntactic formulas is. The semantics, in particular, defines which formulas express true facts about the world, either in a particular interpretation ν or about the world in general (for valid formulas,

which are true regardless of the interpretation). Yet, the semantics is usually highly ineffective, so that it cannot be used directly to find out whether a formula is valid. Just think of formulas in differential dynamic logic that express safety properties of hybrid systems. It would not get us very far if we were to try to establish the truth of such a formula by literally computing the semantics (which includes executing the hybrid system) in every initial state, of which there are uncountably infinitely many.

Instead, logics come with *proof calculi* that can be used to establish validity of logical formulas in the logic \mathcal{L} . Those proof calculi comprised *axioms* (Lecture 5) and *proof rules* (Lecture 6 and others), which can be combined to prove or derive logical formulas of the logic \mathcal{L} . The proof calculus of the logic \mathcal{L} defines a relation \vdash between sets of formulas and formulas, in which $\Phi \vdash \phi$ holds iff ϕ is provable from the set of formulas Φ . That is, there is a proof of ϕ in the proof calculus of \mathcal{L} that uses only assumptions from Φ . The most important case for our purposes is again $\Phi = \emptyset$, in which case $\vdash \phi$ holds iff ϕ is provable in the proof calculus of \mathcal{L} , i.e. there is a proof of ϕ .

Of course, only some formulas of \mathcal{L} are provable, not all of them. The formula $p \wedge \neg p$ should not be provable in any proper logic, because it is inconsistently *false* and, thus, cannot possibly be valid.

We could have written down any arbitrary axiom, or we could have accidentally had a typo in the axioms. So a crucial question we have to ask (and have asked every time we introduced an axiom in other lectures of this course) is whether the axioms and proof rules are sound. In a nutshell, a proof calculus is sound if all provable formulas are valid.

Theorem 1 (Soundness [Pla08, Pla10b, Pla12b]). *The proof calculus of differential dynamic logic is sound, i.e. $\vdash \subseteq \models$, which means that $\vdash \phi$ implies $\models \phi$ for all dL formulas ϕ . That is, all provable dL formulas are valid.*

The significance of soundness is that, whatever formula we derive by using the dL proof rules and axioms, we can rest assured that it is valid, i.e. true in all states. In particular, it does not matter how big and complicated the formula might be, we know that it is valid as long as we have a proof for it. About the axioms, we can easily convince ourselves using a soundness proof why they are valid, and then conclude that all provable formulas are also valid, because they follow from sound axioms by sound proof rules.

Note 2 (Necessity of soundness). *Soundness is a must for otherwise we could not trust our own proofs.*

3. Soundness Challenge for CPS

What good would it do to analyze safety of a CPS using a technique that is as faulty as the original CPS? If an unsound analysis technique says that a CPS is correct, we are,

fundamentally, not much better off than without any analysis, because all we can conclude is that we did not find problems, not that there are none.¹ After all, an unsound analysis technique could say “correct”, which might turn out to be a lie because the correctness statement itself was not valid.

Note 3 (Challenge of soundness). *In a domain that is as challenging as cyber-physical systems and hybrid systems, it is surprisingly easy for analysis techniques to become unsound due to subtle flaws. Necessary conditions for soundness and the numerical decidability frontier have been identified in the literature [PC07, Col07]. The crux of the matter is that hybrid systems are subject to a numerical analogue of the halting problem of Turing machines [PC07].*

There is a shockingly large number of approaches that, for subtle reasons, are subject to the unsoundness resulting from non-observance of the conditions identified in [PC07, Col07]. Consequently, such approaches need some of the additional assumptions identified in [PC07, Col07] to have a chance to become sound.

4. First-Order Logic

Even though this course primarily studied extensions of first-order logic by dynamic modalities for hybrid systems instead of pure first-order logic, the sequent proof rules of propositional logic and quantifiers (instantiation and Skolemization) give a suitable proof calculus for first-order logic. And this suitability of the proof calculus for first-order logic is a much stronger statement than soundness.

Soundness is the question whether all provable formulas are valid and is a minimal requirement for proper logics. Completeness studies the converse question whether all valid formulas are provable.

The first-order logic proof calculus can be shown to be both sound and complete, which is a result that originates from Gödel’s PhD thesis [Göd30], albeit in a different form.

¹Notwithstanding of the fact that unsound analysis techniques can still be very useful in practice, especially if they identify problems in system designs. Yet, we should exercise great care in concluding anything from unsound techniques that have not found a problem. As has been aptly phrased by Dijkstra [Dij70]: “Program testing can be used to show the presence of bugs, but never to show their absence!”

Theorem 2 (Soundness & completeness of first-order logic). *First-order logic is sound, i.e. $\vdash \subseteq \models$, which means that $\vdash \phi$ implies $\models \phi$ for all first-order formulas ϕ (all provable formulas are valid). First-order logic is complete, i.e. $\models \subseteq \vdash$, which means that $\models \phi$ implies $\vdash \phi$ for all first-order formulas ϕ (all valid formulas are provable). In particular, the provability relation \vdash and the validity relation \models coincide for first-order logic: $\vdash = \models$. The same holds in the presence of a set of assumptions Γ , i.e. $\Gamma \vdash \phi$ iff $\Gamma \models \phi$, that is, a first-order formula ϕ is provable from a set of first-order assumptions Γ in first-order logic if and only if ϕ is a consequence of Γ , i.e. entailed by Γ , i.e. true in all models of Γ .*

This lecture will not set out for a direct proof of this result, because the techniques used for those proofs are interesting but would lead us too far astray. An indirect justification for what makes first-order logic so special that Theorem 2 can hold will be discussed later.

The following central result about compactness of first-order logic is of similar importance. Compactness is involved in most proofs of Theorem 2, but, once Theorem 2 has been proved, also easily follows from Theorem 2. Compactness means that if a formula A is a consequence of a set of formulas Γ , then it already is a consequence of finitely many formulas.

Theorem 3 (Compactness of first-order logic). *First-order logic is compact, i.e.*

$$\Gamma \models A \iff E \models A \text{ for some finite } E \subseteq \Gamma \quad (1)$$

Proof. By Theorem 2, $\vdash = \models$. By completeness, the semantic compactness theorem (1) is equivalent to the syntactic compactness theorem:

$$\Gamma \vdash A \iff E \vdash A \text{ for some finite } E \subseteq \Gamma \quad (2)$$

Condition (2) is obvious, because provability implies that there is a proof, which can, by definition, only use finitely many assumptions $E \subseteq \Gamma$. \square

Compactness is equivalent to the finiteness property, which, for that reason, is usually simply referred to as compactness. The finiteness property says that a set of formulas Γ has a model if and only if all its finite subsets of formulas have a model.

Corollary 4 (Finiteness). *First-order logic satisfies the finiteness property, i.e.*

$$\Gamma \text{ has a model} \iff \text{all finite } E \subseteq \Gamma \text{ have a model} \quad (3)$$

Proof. Compactness (Theorem 3) implies the finiteness property. The key observation is that Γ has no model iff $\Gamma \models \text{false}$, because if Γ has no model, then false holds in all models of Γ of which there are none. Conversely, the only chance for false to hold in all models of Γ is if there are no such models, since false never holds. By Theorem 3,

$$\Gamma \models \text{false} \iff \exists \text{ finite } E \subseteq \Gamma \ E \models \text{false}$$

Hence,

Γ has a model $\iff \Gamma \not\models \text{false} \iff \forall \text{finite } E \subseteq \Gamma \ E \not\models \text{false} \iff \text{all finite } E \subseteq \Gamma \text{ have a model}$

It is worth noting that, conversely, the finiteness property implies compactness.

$$\begin{aligned} \Gamma \models A &\iff \Gamma \cup \{\neg A\} \text{ has no model} \\ &\iff \text{some finite } E \subseteq \Gamma \cup \{\neg A\} \text{ has no model} && \text{by finiteness} \\ &\iff E \models A \text{ for some finite } E \subseteq \Gamma \end{aligned}$$

The last equivalence uses that we might as well include $\neg A$ in E , because if E has no model then neither does $E \cup \{\neg A\}$. \square

5. Löwenheim-Skolem-Herbrand Theory in a Nutshell

A beautiful and, in the long term, quite impactful theory due to Leopold Löwenheim [Löw15], Thoralf Skolem [Sko20], and Jacques Herbrand [Her30], gave rise to the first automated theorem prover. Granted, it was quite a theoretical procedure at first, but it was the first one and ultimately had practical offspring in the form of instance-based methods. Herbrand's procedure reduces first-order logic validity to a sequence of validity questions in propositional logic, each of which are perfectly decidable by truth-tables (or more practically by SAT solvers). We focus on the parts of the Löwenheim-Skolem-Herbrand theory that are most important for the subsequent development, which, incidentally, are its syntactic aspects. The Löwenheim-Skolem-Herbrand theory is developed in more depth in Appendix A, including the nontrivial semantic justifications for the syntactic transformations.

The first ingredient is Herbrand's theorem on what are now called Herbrand-disjunctions, i.e. validity-preserving instantiations of existential quantifiers as a disjunction of finitely many terms. Observe the relationship to quantifier elimination in real arithmetic, which will be discussed in more detail later.

Theorem 5 (Herbrand's theorem: Herbrand disjunctions [Her30]). *For a quantifier-free formula $\phi(x)$ of a free variable x without equality*

$$\exists x \phi(x) \text{ valid} \iff \phi(t_1) \vee \dots \vee \phi(t_n) \text{ valid for some } n \in \mathbb{N} \text{ and ground terms } t_1, \dots, t_n$$

Observe that the formula $\phi(t_1) \vee \dots \vee \phi(t_n)$ is quantifier-free if $\phi(x)$ was quantifier-free. This reduces the validity of existential formulas of first-order logic to the validity of formulas in propositional logic. Not every first-order formula is of the form required in Theorem 5 but can be converted into that form without altering validity by Herbrandization, a dual of Skolemization:

Lemma 6 (Herbrandization). *With each first-order logic formula ψ , a formula*

$$\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$$

with quantifier-free $\phi(x_1, \dots, x_n)$ can be associated effectively that is valid if and only if ψ is. The formula $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ uses additional function symbols that do not occur in ψ .

Example 7. The formula $\forall x \exists y p(x, y)$ is valid iff the formula $\exists y p(a, y)$ is valid, where a is a new function symbol of arity zero. Obviously, if $\exists y p(a, y)$ is valid, it needs to be true no matter what the interpretation of a is, which implies that $\forall x \exists y p(x, y)$ is valid.

Likewise, $\forall x \exists y \forall z p(x, y, z)$ is valid iff $\exists y p(a, y, b(y))$ is, where a is a new function symbol of arity 0 and b a new function symbol of arity 1, i.e. expecting one argument. This time, the term $b(y)$ ensures that the original formula is valid for any value of z , no matter what the particular value of y was, because b is a function that could have an arbitrary interpretation.

Even if semidecidability was proved differently by Gödel first [Göd29], Herbrand's subsequent result Theorem 5 enables a straightforward and constructive proof of the semidecidability of the validity problem of first-order logic:

Theorem 8 (Semidecidability of first-order logic [Göd29]). *Validity in first-order logic is semidecidable, i.e. there is an algorithm that, given any formula ϕ of first-order logic, correctly reports whether ϕ is valid or not and that also terminates for all ϕ that indeed valid. The algorithm will not generally terminate when ϕ is not valid.*

Proof. The semidecision procedure for validity of first-order logic formulas ψ proceeds as follows:

1. Herbrandize ψ to obtain a formula $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ by Lemma 6, which preserves validity.
2. Enumerate all $m \in \mathbb{N}$ and all ground terms t_i^j ($1 \leq j \leq n, 1 \leq i \leq m$), over the new signature.
 - a) If the *propositional* formula

$$\phi(t_1^1, \dots, t_1^n) \vee \dots \vee \phi(t_m^1, \dots, t_m^n)$$

is valid, then so is $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ and, hence, ψ is valid.

By Theorem 5 and Lemma 6, the procedure terminates for all valid first-order formulas (yet fails to terminate for other formulas). \square

The procedure in this proof will always succeed but is rather silly, because it enumerates all terms for instantiation rather blindly. Nevertheless, refinements of this

idea lead to very successful automated theorem proving techniques for first-order logic known as *instance-based methods* [BT10], which restrict the instantiation to instantiation-on-demand in various ways to make the procedure more goal-directed. There are also many successful automatic theorem proving procedures for first-order logic that are based on different principles, including tableaux and resolution [Fit96].

6. Back to CPS

First-order logic is beautiful, elegant, expressive, and simple. Unfortunately, however, it is not expressive enough for hybrid systems [Pla10b, Pla12b, Pla13]. As soon as we come back to studying hybrid systems, the situation gets more difficult. And that is not by accident, but, instead, a fundamental property of first-order logic and of hybrid systems. Per Lindström characterized first-order logic in indirect ways to show which properties stronger logics could still possess and which ones they cannot [Lin69]. Hybrid systems themselves are also known not to be semidecidable, which shows that the semidecidable first-order logic cannot understand the full picture of hybrid systems.

Given that differential dynamic logic talks about properties of hybrid systems, and Turing machines are a special case of hybrid systems that just neglects the mention of differential equations, undecidability is not surprising. We show a very simple standalone proof of incompleteness by adapting a proof for programs, e.g., [Pla10d].

Theorem 9 (Incompactness). *Differential dynamic logic is not compact.*

Proof. It is easy to see that there is a set of formulas that has no model even though all finite subsets have a model, consider:

$$\{\langle x' = 1 \rangle x > y\} \cup \{\neg(x + n > y) : n \in \mathbb{N}\}$$

The same happens with

$$\{\langle (x := x + 1)^* \rangle x > y\} \cup \{\neg(x + n > y) : n \in \mathbb{N}\} \quad \square$$

Hence, differential dynamic logic does not have the finiteness property, which is equivalent to compactness (Corollary 4).

Since soundness and completeness imply compactness (see proof of Theorem 3), incompactness implies incompleteness², because dL is sound. An explicit proof is as follows:

Theorem 10 (Incompleteness [Pla08]). *Differential dynamic logic has no effective sound and complete calculus.*

²Strictly speaking, incompleteness only follows for effective calculi. *Relative* soundness and completeness can still be proved for dL [Pla08, Pla10b, Pla12b], which gives very insightful characterizations of the challenges and complexities of hybrid systems.

Proof. Suppose there was an effective sound and complete calculus for $d\mathcal{L}$. Consider a set Γ of formulas that has no model in which all finite subsets have a model, which exists by Theorem 9. Then $\Gamma \models (0 > 1)$ is valid, thus provable by completeness. But since the proof is effective, it can only use finitely many assumptions $E \subset \Gamma$. Thus $E \models (0 > 1)$ by soundness. But then the finite set E has no model, which is a contradiction. \square

Having said these negative (but necessary) results about differential dynamic logic (and, by classical arguments, any other approach for hybrid systems), let's return to the surprisingly amazing positive properties that differential dynamic logic possesses.

For one thing, the basis of differential dynamic logic is the first-order logic of real arithmetic, not arbitrary first-order logic. This enables a particularly pleasant form of Herbrand disjunctions (Theorem 5) resulting from quantifier elimination in real arithmetic (recall Lecture 18 and Lecture 19).

Definition 11 (Quantifier elimination). A first-order theory admits *quantifier elimination* if, with each formula ϕ , a quantifier-free formula $QE(\phi)$ can be associated effectively that is equivalent, i.e. $\phi \leftrightarrow QE(\phi)$ is valid (in that theory).

Theorem 12 (Tarski [Tar51]). *The first-order logic of real arithmetic admits quantifier elimination and is, thus, decidable.*

Also recall from Lecture 18 and Lecture 19 that the quantifier-free formula $QE(\phi)$ is constructed by substitution or virtual substitution from ϕ , with some side constraints on the parameter relations.

Note 14. *Virtual substitution in $FOL_{\mathbb{R}}$ essentially leads to an equivalence of the form*

$$\exists x F \leftrightarrow \bigvee_{t \in T} A_t \wedge F_x^t \quad (4)$$

for a suitable finite set T of extended terms that depends on the formula F and that gets substituted into F virtually, i.e. in a way that results in standard real arithmetic terms, not extended terms.

The quantifier-elimination instantiations (4) are more useful than Theorem 5, because the required terms T for instantiation can be computed effectively and the equivalence holds whether or not the original formula was valid. This makes first-order logic of real arithmetic decidable, while general first-order logic is only semidecidable, because the proof of Theorem 8 still involves search over the appropriate Herbrand terms t_i to use, which were constructed effectively for $FOL_{\mathbb{R}}$. This pleasant basis makes it possible to use the proof calculus of differential dynamic logic to synthesize constraints on the parameters to make an intended conjecture valid [Pla10b]. Aside from the fact that real

numbers are the appropriate basis for understanding real positions and velocities and the like in cyber-physical systems.

7. The Miracle of Hybrid Systems

The $\text{d}\mathcal{L}$ calculus is *sound* [Pla08, Pla12b], that is, every formula that is provable using the $\text{d}\mathcal{L}$ axioms and proof rules is valid, i.e., true in all states. That is, for all $\text{d}\mathcal{L}$ formulas ϕ :

$$\vdash \phi \text{ implies } \models \phi \quad (5)$$

Soundness should be *sine qua non* for formal verification, but, for fundamental reasons [PC07, Col07], is so complex for hybrid systems that it is sometimes inadvertently forsaken. In logic, we ensure soundness just by checking locally once for each axiom and proof rule. Thus, no matter how complicated a proof, the proven $\text{d}\mathcal{L}$ formula is valid, because it is a (complicated) consequence of lots simple valid proof steps.

More intriguingly, however, our logical setting also enables us to ask the converse: is the $\text{d}\mathcal{L}$ proof calculus *complete*, i.e., can it prove all that is true? That is, does the converse of (5) hold? Theorem 10 as well as a simple corollary to Gödel's incompleteness theorem show that already the fragments for discrete dynamical systems and for continuous dynamical systems are incomplete [Pla08].

In logic, the suitability of an axiomatization can still be established by showing completeness relative to a fragment [Coo78, HMP77]. This *relative completeness*, in which we assume we were able to prove valid formulas in a fragment and prove that we can then prove all others, also tells us how subproblems are related computationally. It tells us whether one subproblem dominates the others. Standard relative completeness [Coo78, HMP77], however, which works relative to the data logic, is inadequate for hybrid systems, whose complexity comes from the dynamics, not the data logic, first-order real arithmetic, which is perfectly decidable first-order real arithmetic [Tar51].

From Hybrid to Continuous. Using the proof calculus of $\text{d}\mathcal{L}$, the problem of proving properties of hybrid systems reduces completely to proving properties of elementary continuous systems [Pla08].

Theorem 13 (Continuous relative completeness of $\text{d}\mathcal{L}$ [Pla08, Pla12b]). *The $\text{d}\mathcal{L}$ calculus is a sound and complete axiomatization of hybrid systems relative to differential equations, i.e., every valid $\text{d}\mathcal{L}$ formula can be derived from elementary properties of differential equations.*

In particular, if we want to prove properties of hybrid systems, all we need to do is to prove properties of continuous systems, because the $\text{d}\mathcal{L}$ calculus completely handles all other steps in the proofs that deal with discrete or hybrid systems. Of course, one has to be able to handle continuous systems in order to understand hybrid systems, because continuous systems are a special case of hybrid systems. But it turns out that

this is actually all that one needs in order to verify hybrid systems, because the $d\mathcal{L}$ proof calculus completely axiomatizes all the rest of hybrid systems.

This central result shows that we can prove properties of hybrid systems in the $d\mathcal{L}$ calculus exactly as good as properties of differential equations can be proved. One direction is obvious, because differential equations are part of hybrid systems, so we can only understand hybrid systems to the extent that we can reason about their differential equations. We have shown the other direction by proving that all true properties of hybrid systems can be reduced effectively to elementary properties of differential equations. Moreover, the $d\mathcal{L}$ proof calculus for hybrid systems can perform this reduction constructively and, vice versa, provides a provably perfect lifting of every approach for differential equations to hybrid systems.

Another important consequence of this result is that decomposition can be successful in taming the complexity of hybrid systems. The $d\mathcal{L}$ proof calculus is strictly compositional. All proof rules prove logical formulas or properties of HPs by reducing them to structurally simpler $d\mathcal{L}$ formulas. As soon as we understand that the hybrid systems complexity comes from a combination of several simpler aspects, we can, hence, tame the system complexity by reducing it to analyzing the dynamical effects of simpler parts. This decomposition principle is exactly how $d\mathcal{L}$ proofs can scale to interesting systems in practice. Theorem 13 gives the theoretical evidence why this principle works in general, not just in the case studies that have been considered so far. This is a good illustration of our principle of multi-dynamical systems and even a proof that the decompositions behind the multi-dynamical systems approach are successful. Note that, even though Theorem 13 proves (constructively) that every true property of hybrid systems can be proved in the $d\mathcal{L}$ calculus by decomposition from elementary properties of differential equations, it is still an interesting question which decompositions are most efficient.

From Hybrid to Discrete. In a certain sense, it may appear to be more complicated to handle continuous dynamics than discrete dynamics. If the continuous dynamics are not just subsuming discrete dynamics but if they were “inherently more”, then one might wonder whether hybrid systems verification could be understood with a discrete dynamical system like a classical computer at all. Of course, such a naïve consideration would be quite insufficient, because, e.g., properties of objects in uncountable continuous spaces can very well follow from properties of finitary discrete objects. Finite $d\mathcal{L}$ proof objects, for example, already entail properties about uncountable continuous state spaces of systems.

Fortunately, all such worries about the insufficiency of discrete ways of understanding continuous phenomena can be settled once and for all by studying the proof-theoretical relationship between discrete and continuous dynamics. We have shown not only that the axiomatization of $d\mathcal{L}$ is complete relative to differential equations, but that it is also complete relative discrete systems [Pla12b].

Theorem 14 (Discrete relative completeness of $\text{d}\mathcal{L}$ [Pla12b]). *The $\text{d}\mathcal{L}$ calculus is a sound and complete axiomatization of hybrid systems relative to discrete systems, i.e., every valid $\text{d}\mathcal{L}$ formula can be derived from elementary properties of discrete systems.*

Thus, the $\text{d}\mathcal{L}$ calculus can also prove properties of hybrid systems exactly as good as properties of discrete systems can be proved. Again, the proof of Theorem 14 is constructive, entailing that there is a constructive way of reducing properties of hybrid systems to properties of discrete systems using the $\text{d}\mathcal{L}$ calculus. Furthermore, the $\text{d}\mathcal{L}$ calculus defines a decision procedure for $\text{d}\mathcal{L}$ sentences relative to an oracle for discrete systems [Pla12b]. Theorems 13 and 14 lead to a surprising result aligning discrete and continuous systems properties.

Theorem 15 ($\text{d}\mathcal{L}$ equi-expressibility [Pla12b]). *The logic $\text{d}\mathcal{L}$ is expressible in both its discrete and in its continuous fragment: for each $\text{d}\mathcal{L}$ formula ϕ there is a continuous formula ϕ^b that is equivalent, i.e., $\models \phi \leftrightarrow \phi^b$ and a discrete formula $\phi^\#$ that is equivalent, i.e., $\models \phi \leftrightarrow \phi^\#$. The converse holds trivially. Furthermore, the construction of ϕ^b and $\phi^\#$ is effective (and the equivalences are provable in the $\text{d}\mathcal{L}$ calculus).*

The proof of the surprising result Theorem 15 is constructive but rather nontrivial (some 20 pages). Consequently, all hybrid questions (and, thus, also all discrete questions) can be formulated constructively equivalently as purely continuous questions and all hybrid questions (also all continuous questions) can be formulated constructively equivalently as purely discrete questions. There is a constructive and provable reduction from either side to the other.

Note 18 (Complete logical alignment). *As a corollary to Theorems 13 and 14, we can proof-theoretically and constructively equate*

$$\text{hybrid} = \text{continuous} = \text{discrete}$$

by a complete logical alignment in the sense that proving properties of either of those classes of dynamical systems is the same as proving properties of any other of those classes, because all properties of one system can be provably reduced in a complete, constructive, and equivalent way to any of the other system classes.

Even though each kind of dynamics comes from fundamentally different principles, they all meet in terms of their proof problems being interreducible, even constructively; see Fig. 1. The proof problem of hybrid systems, the proof problem of continuous systems, and the proof problem of discrete systems are, thus, equivalent. Any proof technique for one of these classes of systems completely lifts to proof techniques for the other class of systems.

Since the proof problems interreduce constructively, every technique that is successful for one kind of dynamics lifts to the other kind of dynamics through the $\text{d}\mathcal{L}$ calculus

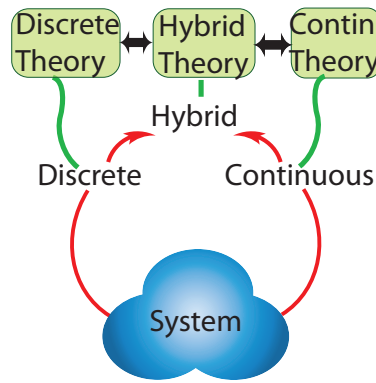


Figure 1: The proof theory of hybrid systems provides a complete proof-theoretical bridge aligning the theory of discrete systems and the theory of continuous systems

in a provably perfect way. Induction, for example, is the primary technique for proving properties of discrete systems. Hence, by Theorem 14, there is a corresponding induction technique for continuous systems and for hybrid systems. And, indeed, *differential invariants* [Pla10a, Pla12d] are such an induction technique for differential equations that has been used very successfully for verifying hybrid systems with more advanced differential equations [PC08, PC09a, PC09b, PQ09, Pla10b, MGP13]. In fact, differential invariants had already been introduced in 2008 [Pla10a] before Theorem 14 was proved [Pla12b], but Theorem 14 implies that a differential invariant induction technique has to exist. These results also show that there are sound ways of using discretization for differential equations [Pla12b] and that numerical integration schemes like, e.g., Euler’s method or more elaborate methods can be used for hybrid systems verification, which is not at all clear a priori due to inherent numerical approximation errors, which may blur decisions either way [PC07].

Challenges with Hybrid Relations. Theorem 15 is a hybrid miracle. Naïve ways of relating discrete and continuous dynamical systems are bound to fail. It is, for example, not generally the case that a property F transfers from a continuous system to its Euler discretization, nor vice versa. That is, neither the following equivalence nor the left-to-right implication nor the right-to-left implication generally holds:

$$[x' = \theta]F \stackrel{?}{\leftrightarrow} [(x := x + h\theta)^*]F \quad (6)$$

This formula would relate a property F of a continuous dynamical system $x' = \theta$ to property F of its Euler discretization $(x := x + h\theta)^*$ with discretization step size $h > 0$ *if only it were true*. Unfortunately, as such, the formula is not generally valid. Fig. 2 illustrates a counterexample to formula (6) from prior work [Pla12b], to which we refer for further details. The error of the Euler discretization grows quickly compared to the true solution in Fig. 2. For example, $F \equiv (x^2 + y^2 = 1)$ is an invariant of the true

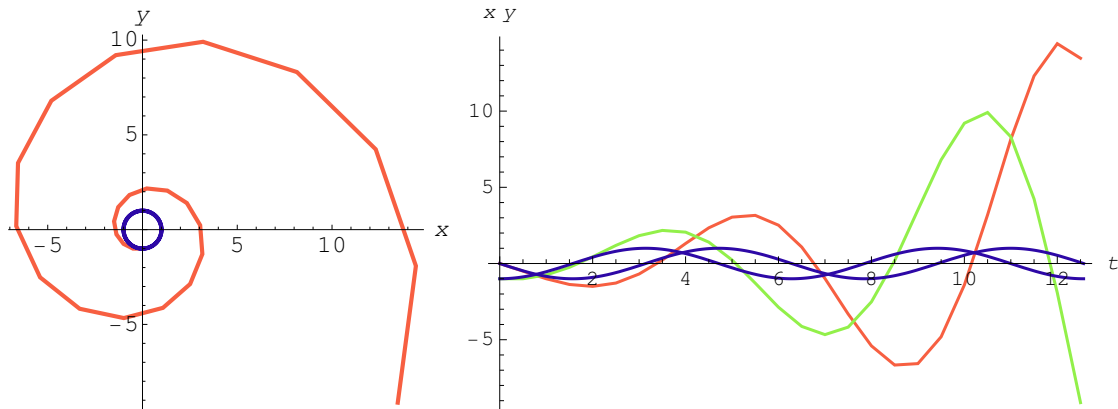


Figure 2: **(left)** Dark blue circle shows true solution, light red line segments show Euler approximation for discretization step $h = \frac{1}{2}$ **(right)** Dark blue true bounded trigonometric solution and Euler approximation in lighter colors with increasing errors over time t

solution but not its approximation. On the bright side, the error can be smaller for *some* (not all) smaller discretization steps h and the error is quite reasonable for a certain period of time.

A. Löwenheim-Skolem-Herbrand-Theory in a Nutshell

The value of a logical formula is subject to interpretation in the semantics of the logic. The truth-value of a formula generally³ depends on the interpretation of its symbols (such as its free variables or function symbols). In a certain sense maybe the most naïve interpretation of first-order logic interprets all terms as themselves. Such an interpretation I is called *Herbrand model*. It stubbornly interprets a term $f(g(a), h(b))$ in the logic as itself: $\llbracket f(g(a), h(b)) \rrbracket_I = f(g(a), h(b))$. And likewise for all other ground terms.

That may sound like a surprising and stubborn interpretation. But, even more surprisingly, it is not at all an un insightful one, at least for first-order logic. So insightful, that it even deserves a name: Herbrand models. Certainly, it is one of the many permitted interpretations.

³Except of the best formulas, which are valid, i.e. true in all interpretations. But we still first need to understand how those valid formulas are interpreted before we could call them valid.

Definition 16 (Herbrand Model). An interpretation I is called *Herbrand model* if it has the free semantics for ground terms, i.e.:

1. The domain D is the ground terms (i.e. terms without variables) $\text{Trm}^0(\Sigma)$ over Σ
2. $I(f) : D^n \rightarrow D; (t_1, \dots, t_n) \mapsto f(t_1, \dots, t_n)$ for each function symbol f of arity n

Let Γ be a set of closed universal formulas. $\text{Trm}^0(\Sigma)(\Gamma)$ is the set of all ground term instances of the formulas in Γ , i.e. with (all possible) ground terms in $\text{Trm}^0(\Sigma)$ instantiated for the variables of the universal quantifier prefix.

$$\text{Trm}^0(\Sigma)(\Gamma) = \{ \phi(t_1, t_2, \dots, t_n) : (\forall x_1 \forall x_2 \dots \forall x_n \phi(x_1, x_2, \dots, x_n)) \in \Gamma \\ t_1, \dots, t_n \in \text{Trm}^0(\Sigma), \text{ for any } n \in \mathbb{N} \}$$

That is, for any $n \in \mathbb{N}$ and for any formula

$$\forall x_1 \forall x_2 \dots \forall x_n \phi(x_1, x_2, \dots, x_n)$$

in Γ and for any ground terms $t_1, \dots, t_n \in \text{Trm}^0(\Sigma)$, the set $\text{Trm}^0(\Sigma)(\Gamma)$ contains the following ground instance of ϕ :

$$\phi(t_1, t_2, \dots, t_n)$$

Theorem 17 (Herbrand [Her30]). Let Γ be a (suitable) set of first-order formulas (i.e. closed universal formulas without equality and with signature Σ having at least one constant). Then

$$\begin{aligned} \Gamma \text{ has a model} &\iff \Gamma \text{ has a Herbrand model} \\ &\iff \text{ground term instances } \text{Trm}^0(\Sigma)(\Gamma) \text{ of } \Gamma \text{ have a model} \end{aligned}$$

Using the Herbrand theorem twice gives:

$$\Gamma \text{ has a model} \iff \text{ground term instances } \text{Trm}^0(\Sigma)(\Gamma) \text{ of } \Gamma \text{ have a Herbrand model}$$

Theorem 8 (Semidecidability of first-order logic [Göd29]). Validity in first-order logic is semidecidable, i.e. there is an algorithm that, given any formula ϕ of first-order logic, correctly reports whether ϕ is valid or not and that also terminates for all ϕ that indeed valid. The algorithm will not generally terminate when ϕ is not valid.

Proof. For suitable first-order formulas F (i.e. $\neg F$ satisfies the assumptions of Theorem 17), semidecidability follows from the following reductions:

$$\begin{aligned} F \text{ valid} &\iff \neg F \text{ unsatisfiable} \\ &\iff \text{Trm}^0(\Sigma)(\neg F) \text{ have no model} && \text{by Theorem 17} \\ &\iff \text{some finite subset of } \text{Trm}^0(\Sigma)(\neg F) \text{ has no Herbrand model} && \text{by Corollary 4} \end{aligned}$$

Thus, it remains to consider the assumptions in Theorem 17 whether first-order formulas that are not suitable can be turned into formulas that are suitable. First of all, Σ can be assumed without loss of generality to have at least one constant symbol for, otherwise, a constant can be added to Σ without changing validity of F . Furthermore, a formula F is valid iff its universal closure is, where the universal closure of a formula F is obtained by prefixing F with universal quantifiers $\forall x$ for each variable x that occurs free in F . Finally, existential quantifiers in first-order formula $\neg F$ can be removed without affecting satisfiability by Skolemization, which introduces new function symbols much like the quantifier proof rules from Lecture 6 did. \square

Note 22 (Limitations of Herbrand models). *Herbrand models are not the cure for everything in first-order logic, because they unwittingly forget about the intimate relationship of the term $2 + 5$ to the term $5 + 2$ and, for that matter, to the term $8 - 1$. All those terms ought to denote the same identical object, but end up denoting different ground terms in Herbrand models. In particular, a Herbrand model would not mind at all if a unary predicate p would hold of $2 + 5$ but not hold for $5 + 2$ even though both ought to denote the same object. Thus, Herbrand models are a little weak in arithmetic, but otherwise incredibly powerful.*

Herbrand's theorem has a second form with a close resemblance to the core arguments of quantifier elimination in first order logic of real arithmetic from Lecture 18 and Lecture 19.

Theorem 5 (Herbrand's theorem: Herbrand disjunctions [Her30]). *For a quantifier-free formula $\phi(x)$ of a free variable x without equality*

$$\exists x \phi(x) \text{ valid} \iff \phi(t_1) \vee \dots \vee \phi(t_n) \text{ valid for some } n \in \mathbb{N} \text{ and ground terms } t_1, \dots, t_n$$

Proof. The proof follows directly from Theorem 17 and Corollary 4:

$\exists x \phi(x)$ valid

$$\iff \neg \exists x \phi(x) \text{ unsatisfiable}$$

$$\iff \forall x \neg \phi(x) \text{ has no model}$$

$$\iff \text{Trm}^0(\Sigma)(\forall x \neg \phi(x)) \text{ has no model} \quad \text{by Theorem 17}$$

$$\iff \{\neg \phi(t) : t \text{ ground term}\} \text{ has no model} \quad \text{by definition}$$

$$\iff \{\neg \phi(t_1), \dots, \neg \phi(t_n)\} \text{ has no model for some } t_1, \dots, t_n \text{ and some } n \quad \text{by Corollary 4}$$

$$\iff \neg \phi(t_1) \wedge \dots \wedge \neg \phi(t_n) \text{ has no model for some } n \text{ and some } t_1, \dots, t_n$$

$$\iff \phi(t_1) \vee \dots \vee \phi(t_n) \text{ valid for some } n \text{ and some } t_1, \dots, t_n \quad \square$$

Theorem 5 continues to hold for first-order formulas $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ with multiple existential quantifiers. More general forms of the Herbrand theorem hold for arbitrary first-order formulas that are not in the specific form assumed above [Her30].

These more general Herbrand theorems won't be necessary for us, because, for validity purposes, first-order formulas can be turned into the form $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ with quantifier-free $\phi(x_1, \dots, x_n)$ by introducing new function symbols for the universal quantifiers using essentially the quantifier proof rules from [Lecture 6](#):⁴

$$(\forall r) \frac{\Gamma \vdash \phi(s(X_1, \dots, X_n)), \Delta_1}{\Gamma \vdash \forall x \phi(x), \Delta} \quad (\exists l) \frac{\Gamma, \phi(s(X_1, \dots, X_n)) \vdash \Delta_1}{\Gamma, \exists x \phi(x) \vdash \Delta}$$

¹ s is a new (Skolem-Herbrand) function and X_1, \dots, X_n are all (existential) free logical variables of $\forall x \phi(x)$.

The clou about quantifier rules $\forall r, \exists l$ is that they preserve validity. By soundness, if their premiss is valid then so is their conclusion. Yet, in the case of rules $\forall r, \exists l$ the converse actually holds as well. If their conclusion is valid then so is their premiss. For rule $\forall r$, for example, the conclusion says that $\phi(x)$ holds for all values of x in all interpretations where Γ holds and Δ does not. Consequently, in those interpretations, $\phi(s(X_1, \dots, X_n))$ holds whatever the interpretation of s is, because s is a fresh function symbol, which, thus, does not appear in Γ, Δ .

Lemma 18 (Herbrandization). *With each first-order logic formula ψ , a formula*

$$\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$$

with quantifier-free $\phi(x_1, \dots, x_n)$ can be associated effectively that is valid if and only if ψ is. The formula $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$ uses additional function symbols that do not occur in ψ .

The procedure in this proof will always succeed but it enumerates the ground terms for instantiation rather blindly, which can cause for quite a bit of waiting. Nevertheless, refinements of this idea lead to very successful automated theorem proving techniques for first-order logic known as *instance-based methods* [BT10], which restrict the instantiation to instantiation-on-demand in various ways to make the procedure more goal-directed. There are also many successful automatic theorem proving procedures for first-order logic that are based on different principles, including tableaux and resolution [Fit96].

Exercises

Exercise 1. The arguments for incompleteness and compactness of \mathcal{dL} hardly depend on \mathcal{dL} , but, rather, only on \mathcal{dL} 's ability to characterize natural numbers. Incompleteness

⁴The new function symbols are usually called Skolem functions and the process called Skolemization, because Thoralf Skolem introduced them in the first correct proof of the Skolem-Löwenheim theorem [Sko20]. Strictly speaking, however, Herbrand functions and Herbrandization are the more adequate names, because Jacques Herbrand introduced this dual notion for the first proof of the Herbrand theorem [Her30]. Skolemization and Herbrandization are duals. Skolemization preserves satisfiability while Herbrandization preserves validity.

and compactness hold for other logics that characterize natural numbers due to a famous result of Gödel [Göd31]. Both the discrete and the continuous fragment of $d\mathcal{L}$ can characterize the natural numbers [Pla08].

1. Show that the natural numbers can be characterized in the discrete fragment of $d\mathcal{L}$, i.e. only using assignments and repetition.
2. Then go on to show that the natural numbers can also be characterized in the continuous fragment of $d\mathcal{L}$, i.e. using only differential equations.
3. Conclude from this that both the discrete and the continuous fragment of $d\mathcal{L}$ are not compact, nor is any other logic that can characterize the natural numbers.

References

- [And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer, 2nd edition, 2002.
- [BT10] Peter Baumgartner and Evgenij Thorstensen. Instance based methods - a brief overview. *KI*, 24(1):35–42, 2010.
- [Col07] Pieter Collins. Optimal semicomputable approximations to reachable and invariant sets. *Theory Comput. Syst.*, 41(1):33–48, 2007. doi:10.1007/s00224-006-1338-3.
- [Coo78] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978.
- [Dij70] Edsger Wybe Dijkstra. Structured programming. In John Buxton and Brian Randell, editors, *Software Engineering Techniques. NATO Software Engineering Conference 1969*. NATO Scientific Committee, 1970.
- [Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, 2nd edition, 1996.
- [Göd29] Kurt Gödel. *Über die Vollständigkeit des Logikkalküls*. PhD thesis, Universität Wien, 1929.
- [Göd30] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktorenkalküls. *Mon. hefte Math. Phys.*, 37:349–360, 1930.
- [Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys.*, 38:173–198, 1931.
- [Her30] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie, Class III, Sciences Mathématiques et Physiques*, 33:33–160, 1930.
- [HMP77] David Harel, Albert R. Meyer, and Vaughan R. Pratt. Computability and completeness in logics of programs (preliminary report). In *STOC*, pages 261–268. ACM, 1977.

- [LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012.* IEEE, 2012.
- [Lin69] Per Lindström. On extensions of elementary logic. *Theoria*, 35:1–11, 1969. [doi:10.1111/j.1755-2567.1969.tb00356.x](https://doi.org/10.1111/j.1755-2567.1969.tb00356.x).
- [Löw15] Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915.
- [MGP13] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In Paul Newman, Dieter Fox, and David Hsu, editors, *Robotics: Science and Systems*, 2013.
- [PC07] André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors, *HSCC*, volume 4416 of *LNCS*, pages 473–486. Springer, 2007. [doi:10.1007/978-3-540-71493-4_37](https://doi.org/10.1007/978-3-540-71493-4_37).
- [PC08] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. [doi:10.1007/978-3-540-70545-1_17](https://doi.org/10.1007/978-3-540-70545-1_17).
- [PC09a] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. Special issue for selected papers from CAV’08. [doi:10.1007/s10703-009-0079-8](https://doi.org/10.1007/s10703-009-0079-8).
- [PC09b] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009. [doi:10.1007/978-3-642-05089-3_35](https://doi.org/10.1007/978-3-642-05089-3_35).
- [Pla07] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007. [doi:10.1007/978-3-540-72734-7_32](https://doi.org/10.1007/978-3-540-72734-7_32).
- [Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. [doi:10.1007/s10817-008-9103-8](https://doi.org/10.1007/s10817-008-9103-8).
- [Pla10a] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. [doi:10.1093/logcom/exn070](https://doi.org/10.1093/logcom/exn070).
- [Pla10b] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. [doi:10.1007/978-3-642-14509-4](https://doi.org/10.1007/978-3-642-14509-4).
- [Pla10c] André Platzer. Quantified differential dynamic logic for distributed hybrid systems. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *LNCS*, pages 469–483. Springer, 2010. [doi:10.1007/978-3-642-15205-4_36](https://doi.org/10.1007/978-3-642-15205-4_36).
- [Pla10d] André Platzer. Theory of dynamic logic. Lecture Notes 15-816 Modal Logic, Carnegie Mellon University, 2010. URL: <http://www.cs.cmu.edu/~fp/courses/15816-s10/lectures/25-DLtheo.pdf>.

- [Pla11] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 431–445. Springer, 2011. doi:[10.1007/978-3-642-22438-6_34](https://doi.org/10.1007/978-3-642-22438-6_34).
- [Pla12a] André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4):1–44, 2012. Special issue for selected papers from CSL’10. doi:[10.2168/LMCS-8\(4:17\)2012](https://doi.org/10.2168/LMCS-8(4:17)2012).
- [Pla12b] André Platzer. The complete proof theory of hybrid systems. In *LICS [LIC12]*, pages 541–550. doi:[10.1109/LICS.2012.64](https://doi.org/10.1109/LICS.2012.64).
- [Pla12c] André Platzer. Logics of dynamical systems. In *LICS [LIC12]*, pages 13–24. doi:[10.1109/LICS.2012.13](https://doi.org/10.1109/LICS.2012.13).
- [Pla12d] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. doi:[10.2168/LMCS-8\(4:16\)2012](https://doi.org/10.2168/LMCS-8(4:16)2012).
- [Pla13] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.
- [Pla14] André Platzer. Analog and hybrid computation: Dynamical systems and programming languages. *Bulletin of the EATCS*, 114, 2014. URL: <http://bulletin.eatcs.org/index.php/beatcs/article/viewFile/292/274>.
- [PQ09] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009. doi:[10.1007/978-3-642-10373-5_13](https://doi.org/10.1007/978-3-642-10373-5_13).
- [Sch12] Peter H. Schmitt. Formale Systeme. Vorlesungsskriptum Fakultät für Informatik, Universität Karlsruhe, 2012.
- [Sko20] Thoralf Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Videnskapsselskapet Skrifter, I. Matematisk-naturvidenskabelig Klasse*, 6:1–36, 1920.
- [Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.