15-424: Foundations of Cyber-Physical Systems

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 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 15-122 Principles of Imperative Computation, 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, 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 that 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 beyond the (clever) test cases that have been tried. 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 dy-

namic 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, 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.

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].

2 The Adventures of a Bouncing Ball

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

$$\Big(\big((?x - o > 5; a := a + 1) \cup a := -b\big); \\ x' = v, v' = a\Big)^* \tag{1}$$

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 bouncing 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

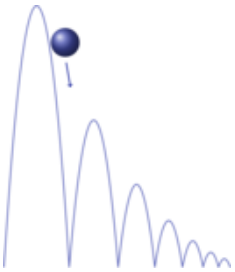

Figure 1: Sample trajectory of a bouncing ball (plotted as position over time)

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 h is falling subject to gravity:

$$h'' = -g$$

When it hits the ground, which is assumed at height $h = 0$, 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 $h = 0$:

$$\begin{array}{l} h'' = -g; \\ \texttt{if}(h = 0)\, h := h + 10 \end{array} \tag{2}$$

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:

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

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

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

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

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

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:

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

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 $h \geq 0$ is what the ball asks dear physics to obey, since the table is of rather sturdy built:

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

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 $h \neq 0$ 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). 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 $0 \leq h \leq H$ when following HP (7).

Scared of what otherwise might happen to it if $0 \leq h \leq H$ 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 `@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.

$$\begin{array}{l} \texttt{@ensures}(0 \leq h) \\ \texttt{@ensures}(h \leq H) \\ \big(h' = v, v' = -g \,\&\, h \geq 0; \\ \quad \texttt{if}(h = 0)\, v := -cv\big)^* \end{array} \tag{8}$$

[1] Safety 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 challenge.

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

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

$$\begin{array}{l} \texttt{@requires}(h = H) \\ \texttt{@requires}(0 \leq H) \\ \texttt{@ensures}(0 \leq h) \\ \texttt{@ensures}(h \leq H) \\ \big(h' = v, v' = -g \,\&\, h \geq 0; \\ \quad \texttt{if}(h = 0)\, v := -cv\big)^* \end{array} \tag{9}$$

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 $h \geq 0$:

$$\begin{array}{l}\texttt{@requires}(h = H)\\ \texttt{@requires}(0 \leq H)\\ \texttt{@ensures}(0 \leq h)\\ \texttt{@ensures}(h \leq H)\\ \big(h' = v, v' = -g \,\&\, h \geq 0;\\ \quad \texttt{if}(h = 0)\, v := -cv\big)^* \texttt{@invariant}(h \geq 0)\end{array} \tag{10}$$

On second thought, the little bouncing ball is less sure what exactly the $\texttt{@invariant}(F)$ 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 $\texttt{@invariant}(F)$ 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. 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 courses 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 introduced first-order logic of real arithmetic.

Note 1 (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 2 (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, 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.*

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

$$[\alpha]E$$

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

$$[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*]\, 0 \leq h \tag{11}$$

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

$$[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*]\, h \leq H \tag{12}$$

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:

$$\begin{aligned}&[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*]\, 0 \leq h\\ \wedge\ &[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*]\, h \leq H\end{aligned} \tag{13}$$

Stepping back, we could also have combined the two postconditions $\texttt{@ensures}(0 \leq h)$ and $\texttt{@ensures}(h \leq H)$ into a single postcondition $\texttt{@ensures}(0 \leq h \wedge h \leq H)$. 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:

$$[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*]\, (0 \leq h \wedge h \leq H) \tag{14}$$

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 dℒ 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 dℒ 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 dℒ 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 (dℒ formula)**.** The *formulas of differential dynamic logic* (dℒ) are defined by the grammar (where ϕ, ψ are dℒ 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 dℒ formulas that are also formulas of first-order real arithmetic (i.e. formulas without modalities), the semantics of dℒ 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 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 $[\![\theta_1]\!]_\nu = [\![\theta_2]\!]_\nu$.
- $\nu \models (\theta_1 \geq \theta_2)$ iff $[\![\theta_1]\!]_\nu \geq [\![\theta_2]\!]_\nu$.
- $\nu \models \neg\phi$ 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 ($\nu \models \phi$ and $\nu \models \psi$) or ($\nu \not\models \phi$ 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 $d \in \mathbb{R}$.
- $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$.
- $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$.

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 $\vDash \phi$, iff $\nu \models \phi$ for all states ν. A formula ϕ is a *consequence* of a set of formulas Γ, written $\Gamma \vDash \phi$, iff, for each ν: ($\nu \models \psi$ for all $\psi \in \Gamma$) implies that $\nu \models \phi$.

9 CPS Contracts in Logic

Now that we know what truth and validity are, let's go back to the previous question. Is dℒ formula (13) valid? Is (14) valid? Indeed, they are equivalent, i.e. the dℒ formula

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

is valid. Expanding the abbreviations that is the following dℒ formula is valid:

$$\begin{aligned}&\Big([(h'=v,v'=-g\,\&\,h\geq 0;\ \texttt{if}(h=0)\,v:=-cv)^*]\,0\leq h\\ &\wedge\,[(h'=v,v'=-g\,\&\,h\geq 0;\ \texttt{if}(h=0)\,v:=-cv)^*]\,h\leq H\Big)\\ &\leftrightarrow\ [(h'=v,v'=-g\,\&\,h\geq 0;\ \texttt{if}(h=0)\,v:=-cv)^*]\,(0\leq h\wedge h\leq H)\end{aligned}\tag{15}$$

So if (13) is valid, then so should (14) be (Exercise 1). But is (13) valid?

Certainly, (13) is not true in a state ν where $\nu(h) < 0$, because from that initial state, no repetitions of the loop (which is allowed by nondeterministic repetition, Exercise 3), will lead to a state $\omega \overset{\text{def}}{=} \nu$ in which $\omega \not\models 0 \leq h$. Thus, (13) only has a chance of being valid in initial states that satisfy further assumptions, including $0 \leq h$ and $h \leq H$. In fact, that is what the preconditions were meant for in Sect. 4. How can we express a precondition contract in a dL 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 dL has modalities for. Do we also need any extra logical operator to express preconditions?

The meaning of a precondition `@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 `@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 `@requires`(A) `@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

$$A \rightarrow [\alpha]B \tag{16}$$

because an implication is valid if, in every state, its left-hand side is false or its right-hand side true. The implication (16) is valid ($\vDash A \rightarrow [\alpha]B$), 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 dL 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 dL 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 5 (Contracts to dL Formulas)**.** *Consider a HP α with a CPS contract using a single* `@requires`(A) *precondition and a single* `@ensures`(B) *postcondition:*

`@requires`(A)
`@ensures`(B)
α

This CPS contract can be expressed directly as a logical formula in dL:

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

CPS contracts with multiple preconditions and multiple postconditions can directly be expressed as a dℒ formula as well (Exercise 4).

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

$$\begin{aligned}&\texttt{@requires}(0 \leq h \land h = H)\\&\texttt{@ensures}(0 \leq h \land h \leq H)\\&\big(h' = v, v' = -g \,\&\, h \geq 0;\\&\ \texttt{if}(h = 0)\, v := -cv\big)^*\end{aligned} \tag{17}$$

The dℒ formula expressing that the CPS contract for HP (17) holds is:

$$0 \leq h \land h = H \rightarrow \left[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*\right](0 \leq h \land h \leq H) \tag{18}$$

So to find out whether (17) satisfies its CPS contract, we ask whether the dℒ 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 good idea to check whether all required assumptions have been found that are necessary for the formula to hold. So let us scrutinize dℒ 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 an insightful example for illustrating how critical it is to identify the right requirements.

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) does not say. Certainly, if gravity were negative ($g < 0$), bouncing balls would function rather differently. They would suddenly be floating balls disappearing into the sky. So let's modify (18) to assume $g = 9.81$:

$$0 \leq h \land h = H \land g = 9.81 \rightarrow \left[\big(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\big)^*\right](0 \leq h \land h \leq H) \tag{19}$$

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 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.

Note 6 (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 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 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:

$$0 \leq h \land h = H \land g > 0 \rightarrow \left[\left(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\right)^*\right](0 \leq h \land h \leq H) \tag{20}$$

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

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

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?

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. 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 $h = H$. Consequently, (21) has to be modified to assume $v \leq 0$ holds initially:

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

Now there's finally assumptions about all parameters and variables of (22). That does not mean that we found the right assumptions, yet, obviously, 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 $v \leq 0 \land 0 \leq h \land h = H \land g > 0 \land 1 > c \geq 0$ 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. Under which circumstance this happens depends on the relationship of the initial velocity and height to the damping coefficient.

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:

$$0 \leq h \land h = H \land v = 0 \land g > 0 \land 1 > c \geq 0 \rightarrow \left[\left(h' = v, v' = -g \,\&\, h \geq 0;\ \texttt{if}(h = 0)\, v := -cv\right)^*\right](0 \leq h \land h \leq H) \quad (23)$$

Is dℒ 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 dℒ formula (23) valid so that the bouncing ball described in (23) satisfies the postcondition $0 \leq h \leq H$. 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 dℒ formula (23), however, we need to investigate how proving works. How can dℒ formulas be proved? And, since first-order formulas are dℒ 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 dℒ for proving purposes, let's simply (23) by getting rid of `if-then-else` (Exercise 7):

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

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 7 (Principle of Cartesian Doubt)**.** *In 1641, René Descartes suggested an attitude of systematic doubt where he would be skeptical about the truth of all believes 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 necessary.

11 Intermediate Conditions for CPS

Before proceeding any further with ways of proving dℒ formulas, let's simplify (24) grotesquely by removing the loop:

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

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 dℒ 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 dℒ formula (25) has a number of assumptions $0 \leq h \land h = H \land v = 0 \land g > 0 \land 1 > c \geq 0$ that can be used during the proof. It claims that the postcondition $0 \leq h \land h \leq H$ 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.[3]

The HP in (25) follows a differential equation first and then, after the sequential composition (;), proceeds to run a discrete program $(?h = 0; v := -cv \cup ?h \neq 0)$. 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 8 (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 $h(t)$ at time t will be

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

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).[4] 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 h, v, g, H?

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

[3] 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.

[4] 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:

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

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

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

Then substituting the first equation into the second yields

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

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

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

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 $0 \leq h \wedge h \leq H$.

Note 9 (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:

$$A \rightarrow [\alpha]E \qquad \textit{and} \qquad E \rightarrow [\beta]B$$

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:

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

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 $0 \leq h$ holds after all runs of $?h = 0; v := -cv \cup ?h \neq 0$ from all states that satisfy $2gh = 2gH - v^2$. That is a bit much to hope for, however, because $0 \leq h$ 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 $0 \leq h$ into the intermediate condition, thus, requiring us to prove:

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

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

$$A \rightarrow [\alpha \cup \beta]B \tag{30}$$

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 10 (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:

$$A \rightarrow [\alpha]B \qquad \textit{and} \qquad A \rightarrow [\beta]B$$

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:

$$\begin{aligned} 2gh = 2gH - v^2 \land h \geq 0 \rightarrow [?h = 0; v := -cv]\,(0 \leq h \land h \leq H) \\ 2gh = 2gH - v^2 \land h \geq 0 \rightarrow [?h \neq 0]\,(0 \leq h \land h \leq H) \end{aligned} \tag{31}$$

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

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

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:

$$A \to [?H]B \tag{33}$$

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

$$A \land H \to B$$

Note 11 (Proving tests). *For a HP that is a test $?H$, we can prove*

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

by proving the following dL formula:

$$A \land H \to B$$

Using this for the second formula of (31), Note 11 reduces proving the second formula of (31)

$$2gh = 2gH - v^2 \land h \geq 0 \to [?h \neq 0]\,(0 \leq h \land h \leq H)$$

to proving

$$2gh = 2gH - v^2 \land h \geq 0 \land h \neq 0 \to 0 \leq h \land h \leq H \tag{34}$$

Now we are left with arithmetic that we need to prove. Proofs for arithmetic and propositional logical operators such as $\land$ and $\to$ will be considered in a later lecture. For now, we notice that the formula $0 \leq h$ in the right-hand side of $\to$ seems justified

by assumption $h \geq 0$. And that $h \leq H$ does not exactly have a justification in (34), because we lost the assumptions about H somewhere.

How could that happen? We used to know $h \leq H$ 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 9. But similar difficulties can arise in other parts of proof attempts.

In this case, the fact that we lost $h \leq H$ can be fixed by including it in the intermediate conditions, because it can be shown to hold after the differential equation still. 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 somewhat 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 12. *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 aw in a correctness argument. A 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 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.

Exercises

Exercise 1. Let A, B be dL formulas. Suppose $A \leftrightarrow B$ is valid and A is valid. Is B valid? Prove or disprove.

Exercise 2. Let A, B be dL 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 3. 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 4. Suppose you have a HP α with a CPS contract using multiple preconditions $A_1, \ldots, A_n$ and multiple postconditions $B_1, \ldots, B_m$:

$$
\begin{array}{l}
\texttt{@requires}(A_1) \\
\texttt{@requires}(A_2) \\
\vdots \\
\texttt{@requires}(A_n) \\
\texttt{@ensures}(B_1) \\
\texttt{@ensures}(B_2) \\
\vdots \\
\texttt{@ensures}(B_m) \\
\alpha
\end{array}
$$

How can this CPS contract be expressed in a dL formula?

Exercise 5. For each of the following dL 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. $[(x' = 1 \,\&\, false)^*]true$.
15. $[(x' = 1 \,\&\, false)^*]false$.

Exercise 6. 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 7. 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 8 (**). 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?

References

[Asi42] Isaac Asimov. Runaround, 1942.

[DBL12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.

[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.

[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.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[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.

[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* [DBL12], 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* [DBL12], pages 13–24. doi: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.

[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.