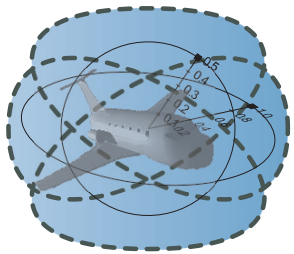
# The Significance of Symbolic Logic for Scientific Education

André Platzer

Karlsruhe Institute of Technology

Carnegie Mellon University

Alexander von
HUMBOLDT
STIFTUNG



Karlsruhe Institute of Technology

# Outline

# Outline

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Compiler Design

1. Y3+: logic program as rules
2. for dataflow analysis, parsing
3. for program semantics
4. for compiler optimizations

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Compiler Design

1. Y3+: logic program as rules
2. for dataflow analysis, parsing
3. for program semantics
4. for compiler optimizations

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Compiler Design

1. Y3+: logic program as rules
2. for dataflow analysis, parsing
3. for program semantics
4. for compiler optimizations

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Prog. Language Semantics

1. MS+: simplify semantics by logic
2. axiomatic semantics, complete
3. contextual equivalence
4. program transformation

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Prog. Language Semantics

1. MS+: simplify semantics by logic
2. axiomatic semantics, complete
3. contextual equivalence
4. program transformation

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

## Principles Imperative Computation

1. Y1: algo & data, imperative prog., program contracts
2. logical vs. operational program reasoning for correctness
3. pre/postconditions, invariants
4. Lab: Virtual machine for C in C0

## Constructive Logic

1. Y3: functional programming: proofs-as-programs Curry-How.
2. logic programming: propositions-as-programs
3. imperative programming: propositions-as-resources

## Prog. Language Semantics

1. MS+: simplify semantics by logic
2. axiomatic semantics, complete
3. contextual equivalence
4. program transformation

## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Logic in the Science Curriculum

See paper and symbolaris.com for more

## Prog. Language Semantics

1. MS+: simplify semantics by logic
2. axiomatic semantics, complete
3. contextual equivalence
4. program transformation

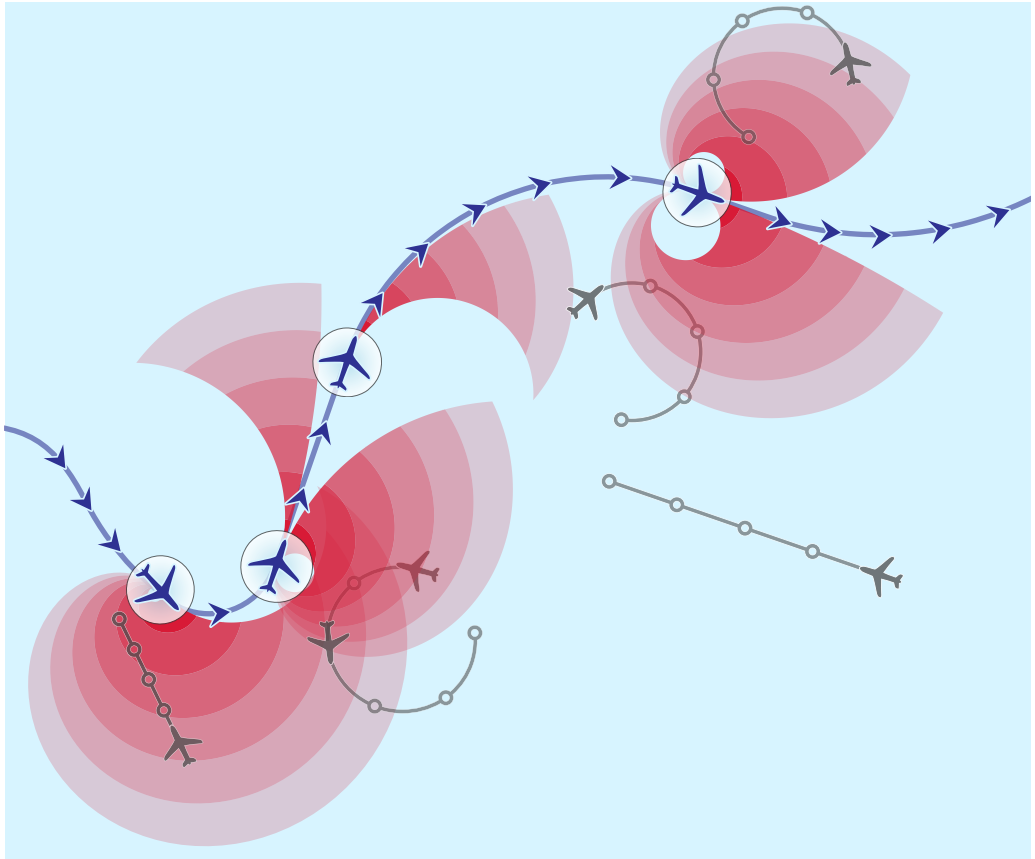
## Log. Foundations Cyber-Phys Sys

1. Y3+: design safe CPS
2. justify safety by logic & proof
3. KeYmaera X prover
4. active learning quiz, video, book

# Outline

# Cyber-Physical Systems Analysis: Aircraft Example

Which control decisions are safe for aircraft collision avoidance?



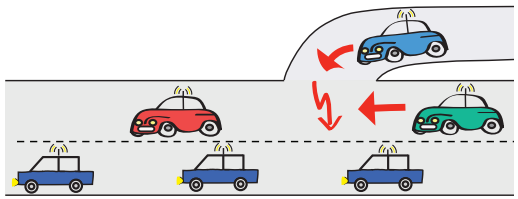
## Cyber-Physical Systems

CPSs combine cyber capabilities with physical capabilities to solve problems that neither part could solve alone.

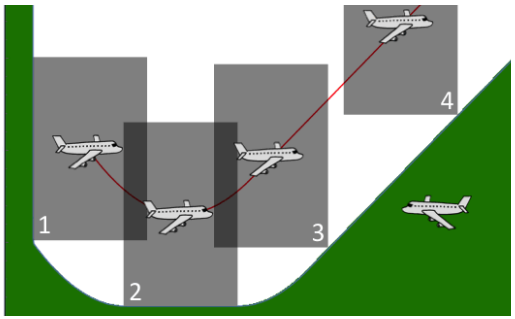
# Cyber-Physical Systems Promise Transformative Impact!

## Prospects: Safety & Efficiency

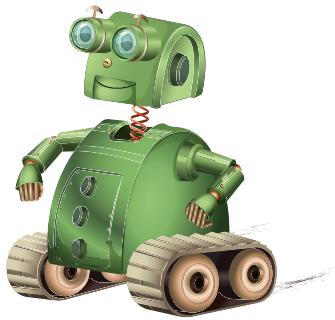
(Autonomous) cars



(Auto)Pilot support




Robots near humans



## Cyber-Physical Systems

CPSs combine cyber capabilities with physical capabilities to solve problems that neither part could solve alone.

# Can you trust a computer to control physics?

# Can you trust a computer to control physics?

1. Depends on how it has been programmed
2. And on what will happen if it malfunctions

## Rationale

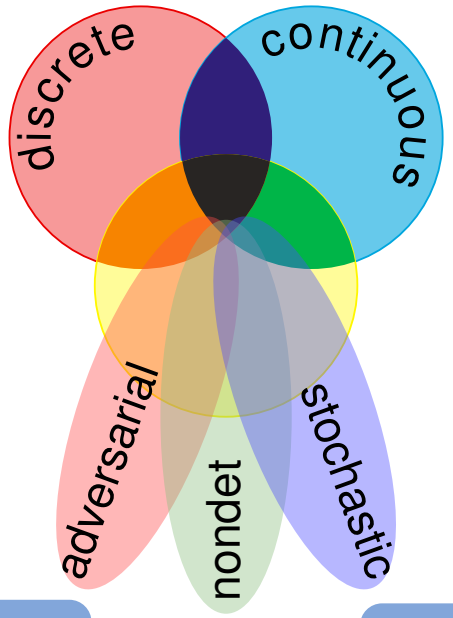
1. Safety guarantees require analytic foundations.
2. A common foundational core helps all application domains.
3. Foundations revolutionized digital computer science & our society.
4. Need even stronger foundations when software reaches out into our physical world.

# CPSs deserve proofs as safety evidence!

# CPSs are Multi-Dynamical Systems

## CPS Dynamics

CPS are characterized by multiple facets of dynamical systems.




## CPS Compositions

CPS combines multiple simple dynamical effects.

Descriptive simplification

## Tame Parts

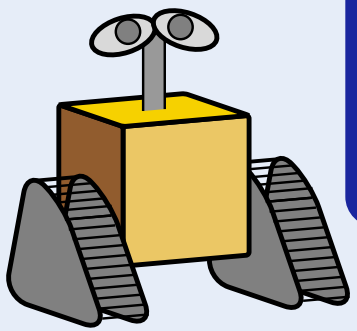
Exploiting compositionality tames CPS complexity.

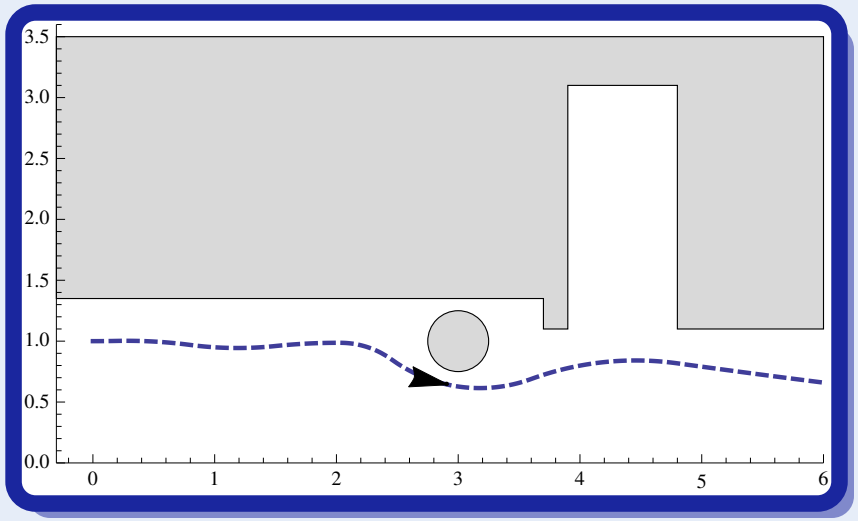
Analytic simplification

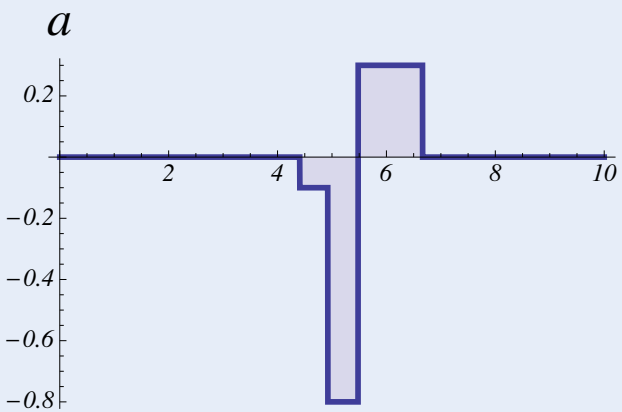
## Challenge (CPS)

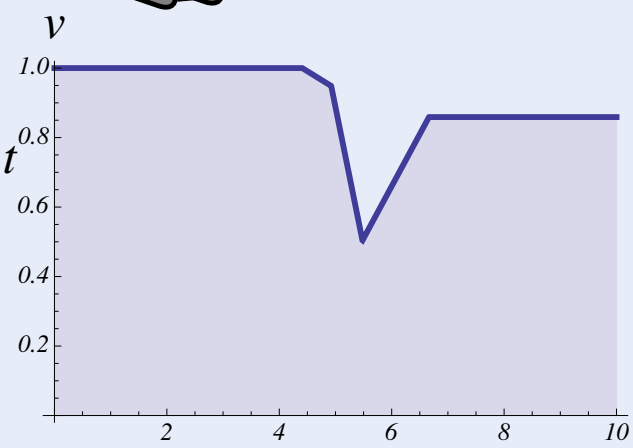
Fixed rule describing state evolution with both

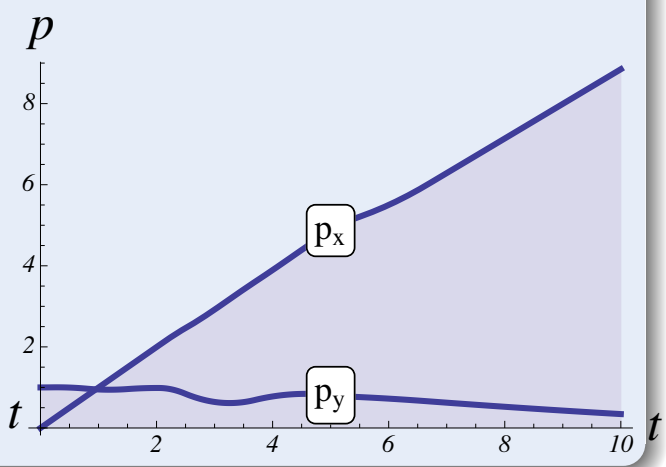
- Discrete dynamics (control decisions)
- Continuous dynamics (differential equations)





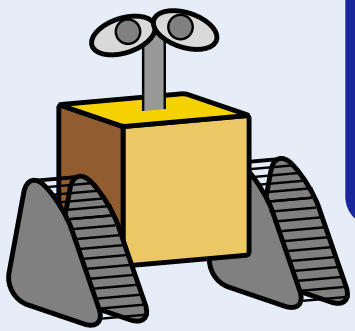



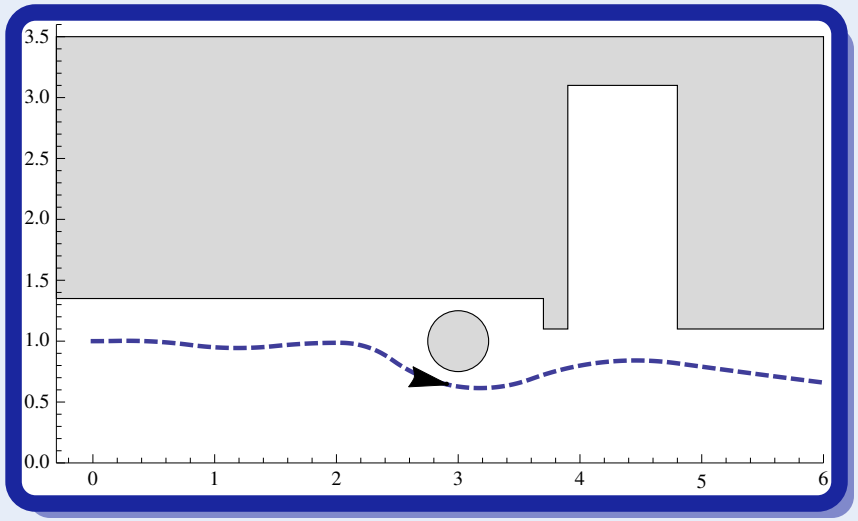



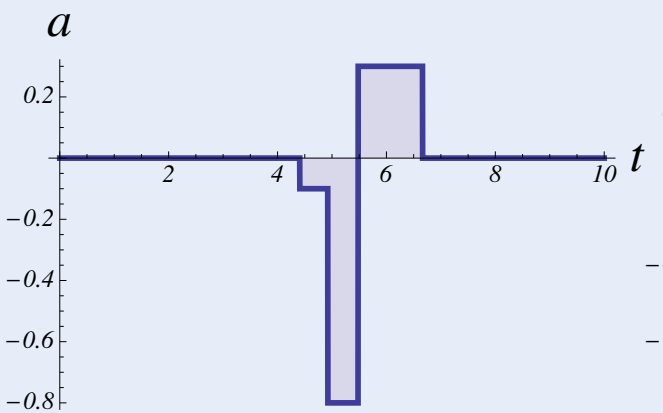
## Challenge (CPS)

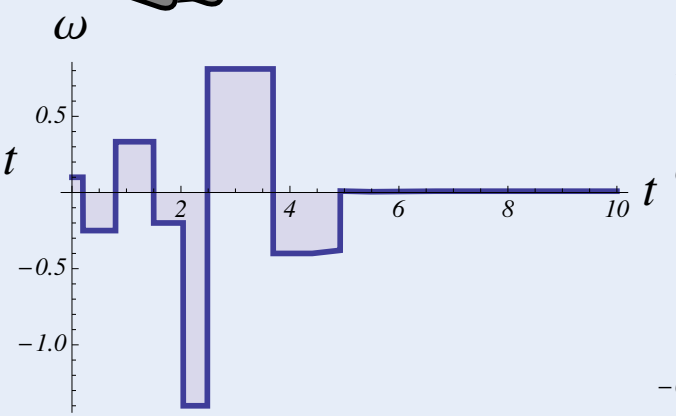
Fixed rule describing state evolution with both

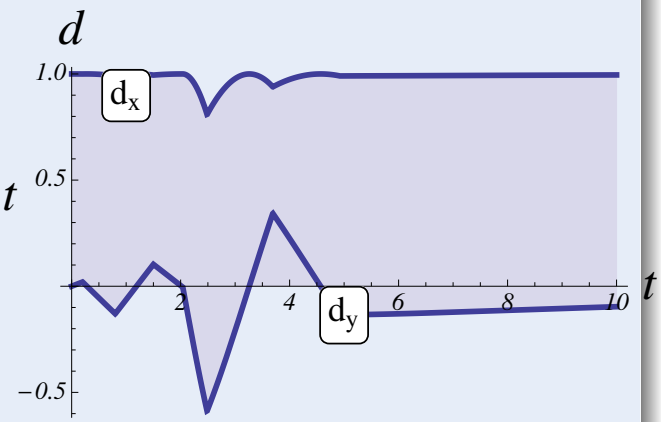
- Discrete dynamics (control decisions)
- Continuous dynamics (differential equations)





# Hybrid Systems Versus Cyber-Physical Systems

Mathematical model for complex physical systems:

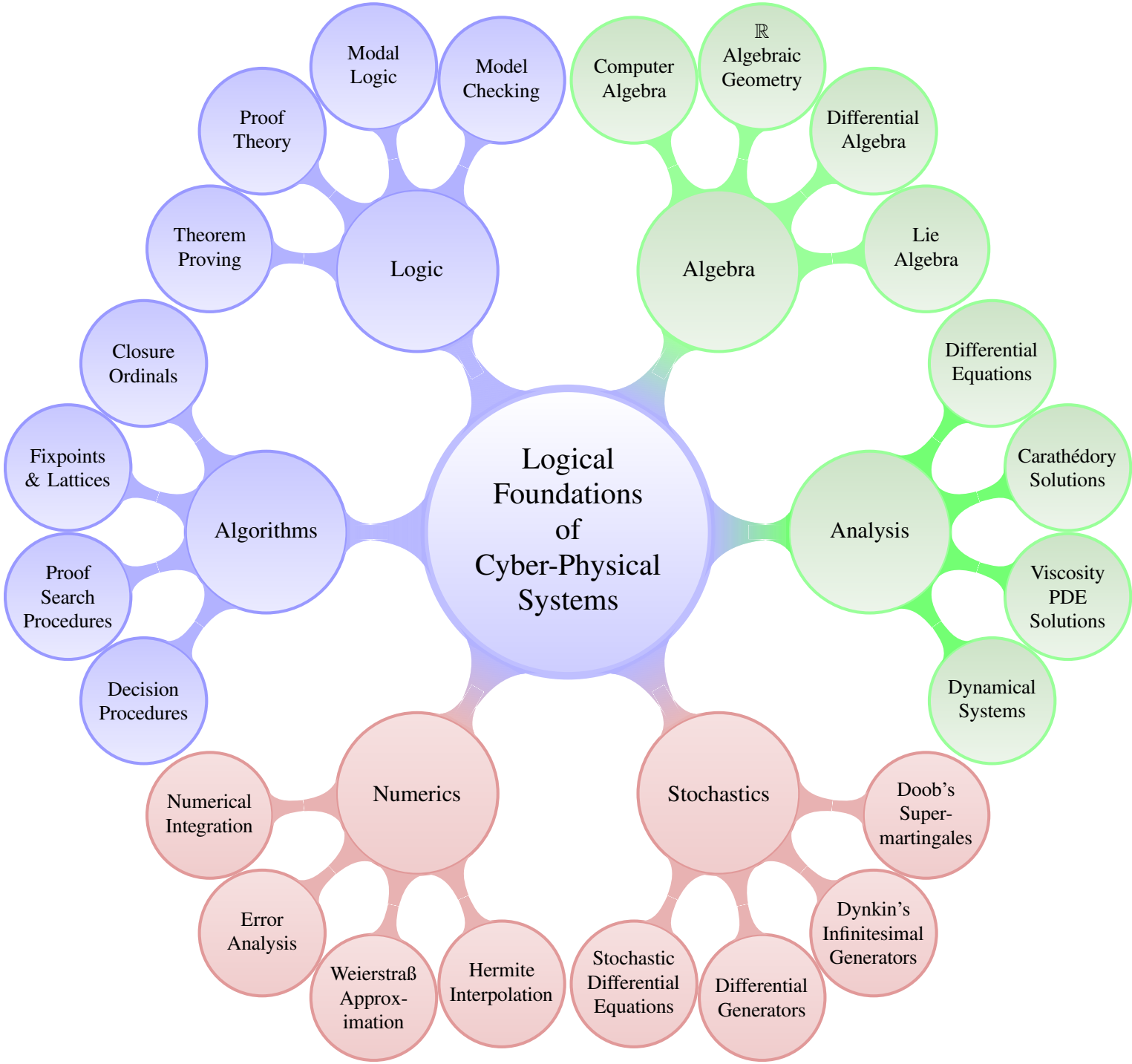
**Definition (Hybrid Systems)**

Systems with interacting discrete and continuous dynamics

Technical characteristics:

**Definition (Cyber-Physical Systems)**

(Distributed networks of) computerized control for physical system
Communication, computation, and control for physics


Logical Foundations of Cyber-Physical Systems
Logic
Modal Logic
Model Checking
Proof Theory
Theorem Proving
Algebra
Computer Algebra
ℝ Algebraic Geometry
Differential Algebra
Lie Algebra
Analysis
Differential Equations
Carathédory Solutions
Viscosity PDE Solutions
Dynamical Systems
Stochastics
Doob's Super-martingales
Dynkin's Infinitesimal Generators
Differential Generators
Stochastic Differential Equations
Numerics
Numerical Integration
Error Analysis
Weierstraß Approx-imation
Hermite Interpolation
Algorithms
Closure Ordinals
Fixpoints & Lattices
Proof Search Procedures
Decision Procedures

# How to Teach Cyber-Physical Systems?

## Onion Model

1. Going outside in
2. Unpeel layer by layer
3. Progress when all prereqs are covered
4. First study CS $\wedge$ math $\wedge$ engineering
5. Talk about CPS in the big finale

## Scenic Tour Model

1. Start at the heart: CPS
2. Go on scenic expeditions into various directions
3. Explore the world around us as we find the need
4. Stay on CPS the whole time
5. Leverage CPS as the guiding motivation for understanding more about connected areas

**Logical scrutiny, formalization, and correctness proofs are critical for CPS!**

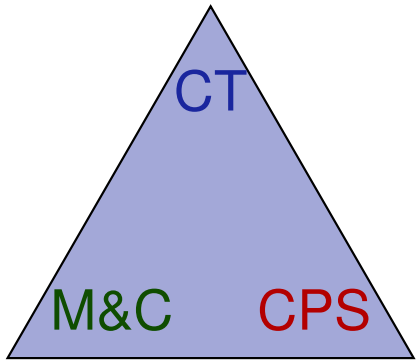
1. CPSs are so easy to get wrong.
2. Retrofitting CPSs for safety is not possible.
3. These logical aspects are an integral part of CPS design.
4. Critical to your understanding of the intricate complexities of CPS.
5. Tame complexity by a simple programming language for core aspects.

# About Logical Foundations of Cyber-Physical Systems

- Foundations!
- Modeling & Control
  1. Understand the core principles behind CPSs.
  2. Develop models and controls.
  3. Identify the relevant dynamical aspects.
- Computational Thinking
  1. Identify safety specifications and critical properties of CPSs.
  2. Understand abstraction in system design.
  3. Express pre- and postconditions for CPS models.
  4. Use design-by-invariant.
  5. Reason rigorously about CPS models.
  6. Verify CPS models of appropriate scale.
- CPS Skills
  1. Understand the semantics of a CPS model.
  2. Develop an intuition for operational effects.
  3. Identify control constraints.
  4. Understand opportunities and challenges in CPS and verification.
- Byproducts
  1. Well-motivated exposure to numerous math and science areas in action.

# Learning Objectives

identify safety specifications for CPS
rigorous reasoning about CPS
understand abstraction & architectures
programming languages for CPS
verify CPS models at scale

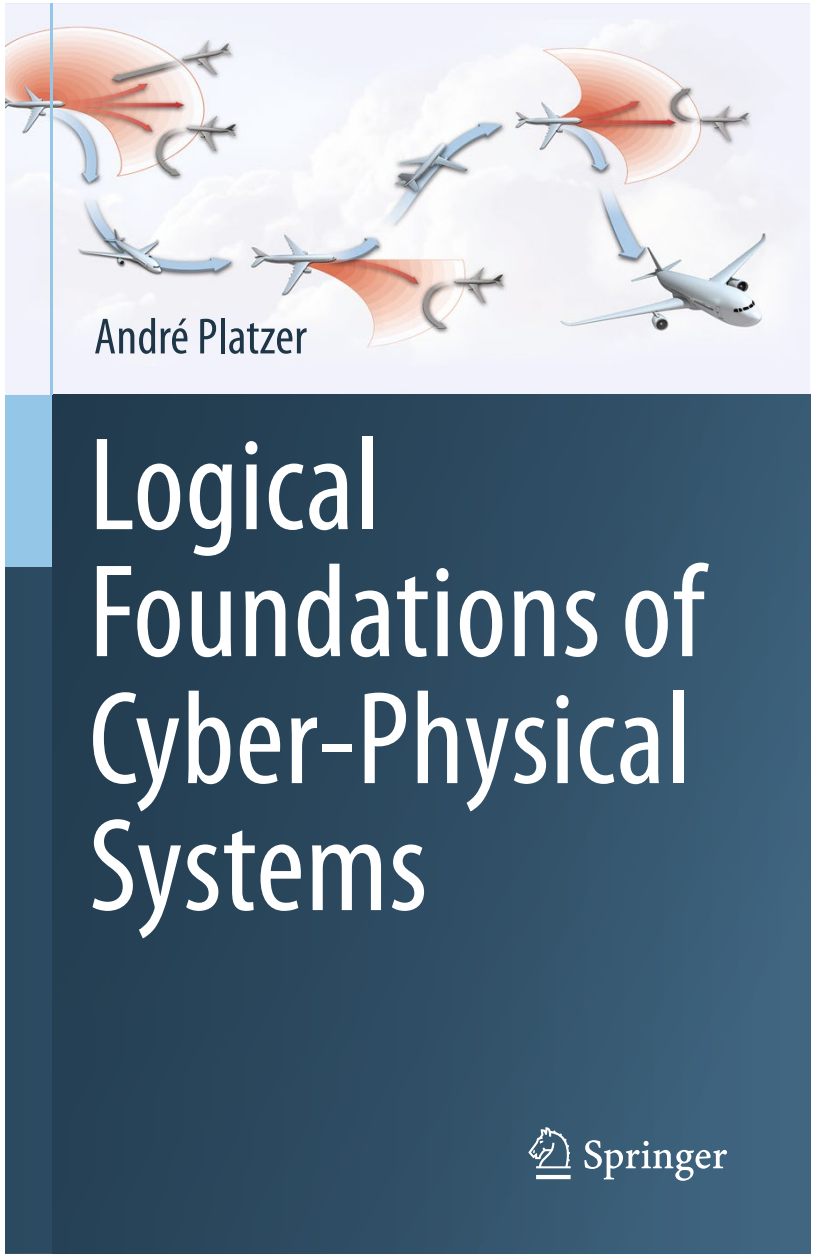


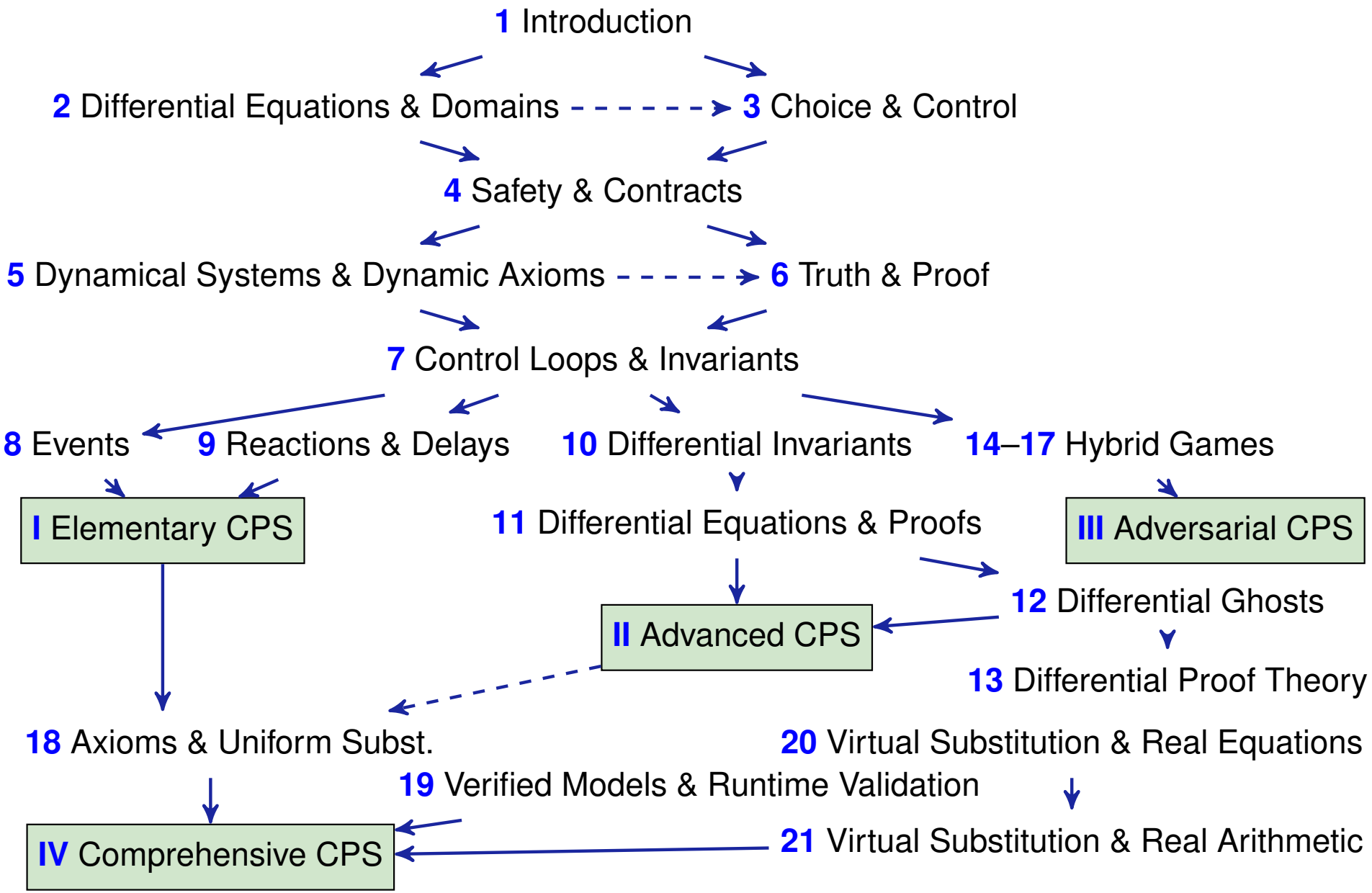

cyber+physics models
core principles of CPS
relate discrete+continuous

semantics of CPS models
operational effects
identify control constraints
opportunities and challenges

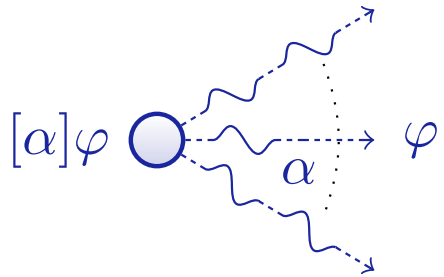
# Textbook and Course Outline

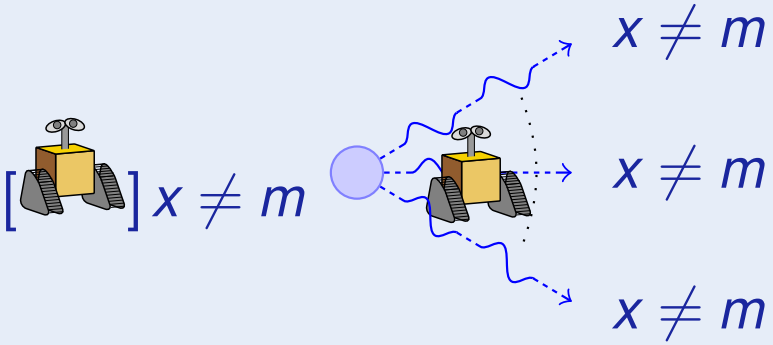




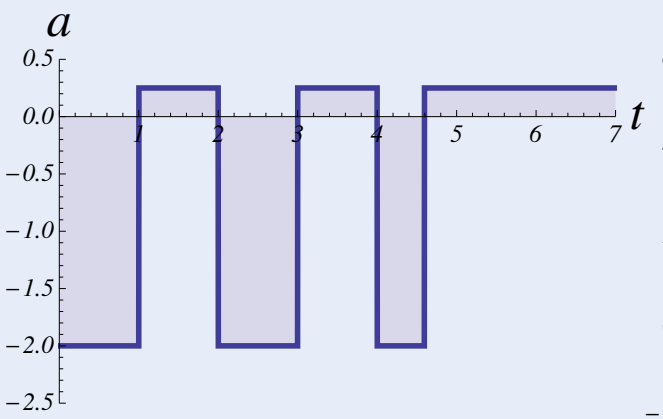

1 Introduction
2 Differential Equations & Domains
3 Choice & Control
4 Safety & Contracts
5 Dynamical Systems & Dynamic Axioms
6 Truth & Proof
7 Control Loops & Invariants
8 Events
9 Reactions & Delays
10 Differential Invariants
14–17 Hybrid Games
I Elementary CPS
11 Differential Equations & Proofs
III Adversarial CPS
12 Differential Ghosts
II Advanced CPS
13 Differential Proof Theory
18 Axioms & Uniform Subst.
20 Virtual Substitution & Real Equations
19 Verified Models & Runtime Validation
IV Comprehensive CPS
21 Virtual Substitution & Real Arithmetic

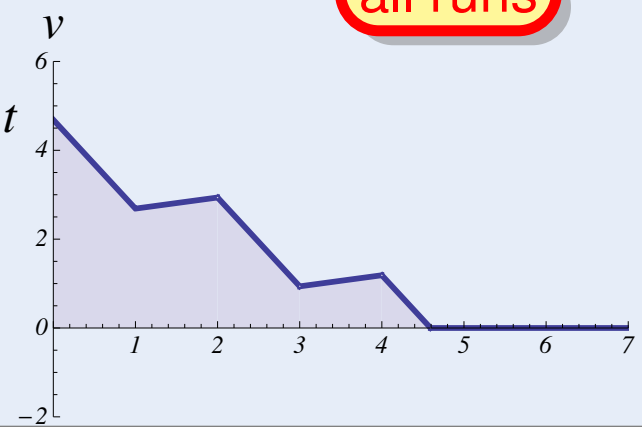
## Concept (Differential Dynamic Logic) (JAR'08,LICS'12)

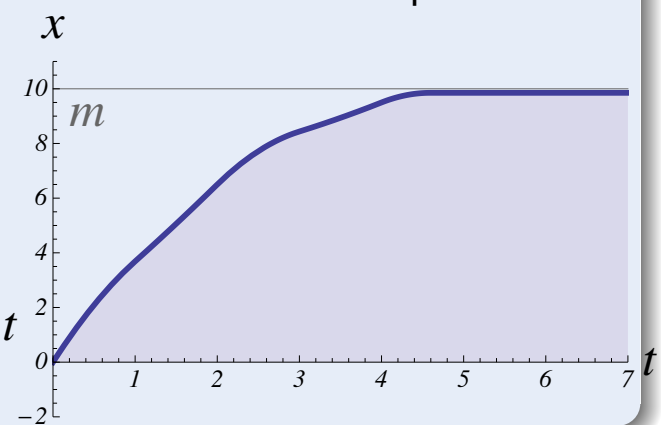



$$\underbrace{x \neq m \wedge b > 0}_{\text{init}} \rightarrow \Big[\underbrace{\big((\text{if}(\text{SB}(x,m)) \quad a := -b)\ ;\ x' = v, v' = a\big)^*}_{\text{all runs}}\Big] \underbrace{x \neq m}_{\text{post}}$$



# Differential Dynamic Logic dL: Axiomatization

equations of truth

$[:=]$ $[x := e]P(x) \leftrightarrow P(e)$

$[?]$ $[?Q]P \leftrightarrow (Q \rightarrow P)$

$[']$ $[x' = f(x)]P \leftrightarrow \forall t{\geq}0\,[x := y(t)]P$ $\qquad (y'(t) = f(y))$

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

$[;]$ $[\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$

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

laws of logic of
laws of physics

K $[\alpha](P \rightarrow Q) \rightarrow ([\alpha]P \rightarrow [\alpha]Q)$

I $[\alpha^*]P \leftrightarrow P \wedge [\alpha^*](P \rightarrow [\alpha]P)$

C $[\alpha^*]\forall v{>}0\,(P(v) \rightarrow \langle\alpha\rangle P(v{-}1)) \rightarrow \forall v\,(P(v) \rightarrow \langle\alpha^*\rangle \exists v{\leq}0\, P(v))$

LICS'12,JAR'17

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \quad [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \quad [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

$$[]\land \quad [\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$$

$$x \leq m \land b > 0 \rightarrow \big([a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, x \leq m \land [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, 0 \leq v\big)$$

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \; [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

$$[]\land \; [\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$$

$$x \leq m \land b > 0 \rightarrow \big([a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, x \leq m \land [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, 0 \leq v\big)$$

easy! True since evolution domain can never be left by construction.

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \; [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

$$[]\land \; [\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$$

$$x \leq m \land b > 0 \rightarrow \big([a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, x \leq m \land [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, 0 \leq v\big)$$

easy! True since evolution domain can never be left by construction.
hard!

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \; [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

$$[]\land \; [\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$$

$$x \leq m \land b > 0 \rightarrow \big([a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, x \leq m \land [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, 0 \leq v\big)$$

easy! True since evolution domain can never be left by construction.
hard! True?

# Example: Car Acceleration or Braking by dL Logic

$$x \leq m \land b > 0 \rightarrow [(a := A \cup a := -b); \{x' = v, v' = a \,\&\, v \geq 0\}](x \leq m \land 0 \leq v)$$

$$[;] \; [\alpha;\beta]P \leftrightarrow [\alpha][\beta]P$$

$$x \leq m \land b > 0 \rightarrow [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0](x \leq m \land 0 \leq v)$$

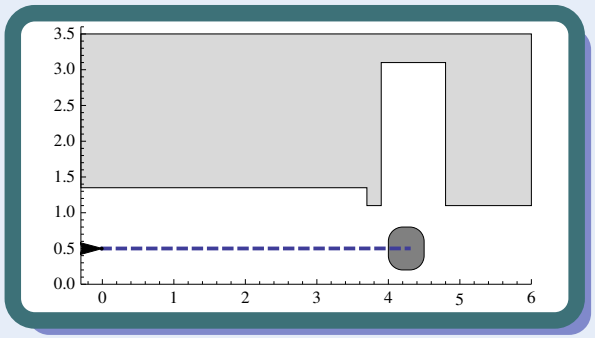
$$[]\land \; [\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$$

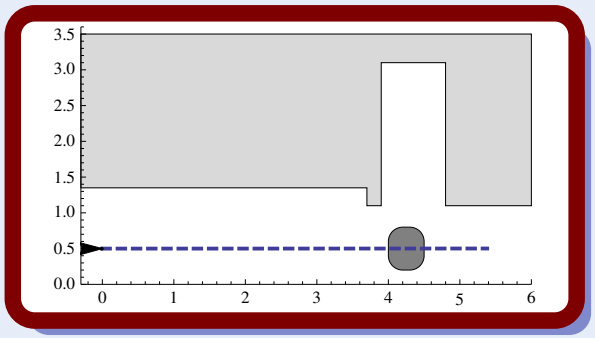
$$x \leq m \land b > 0 \rightarrow \big([a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, x \leq m \land [a := A \cup a := -b][x' = v, v' = a \,\&\, v \geq 0]\, 0 \leq v\big)$$

easy! True since evolution domain can never be left by construction.
hard! True? No but dL logic can find out when it's true.

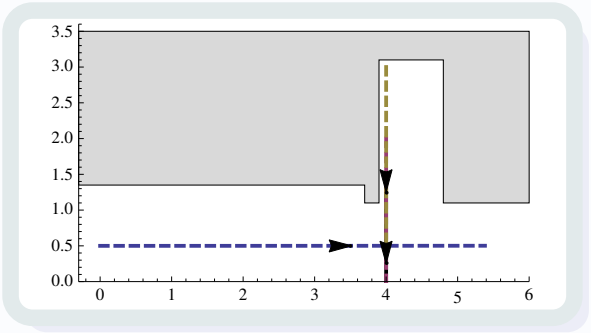
# LFCPS Betabot & Veribot Labs

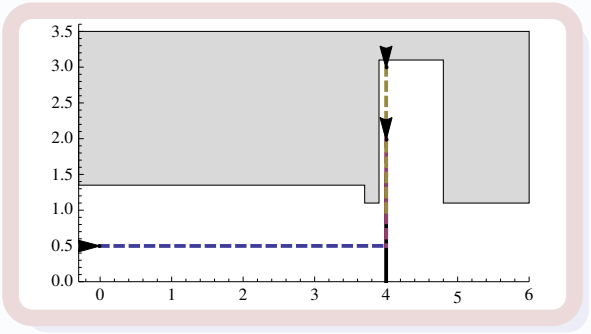
## 1: Charging Station



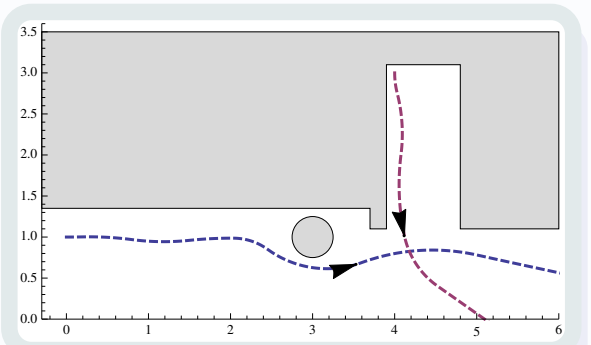



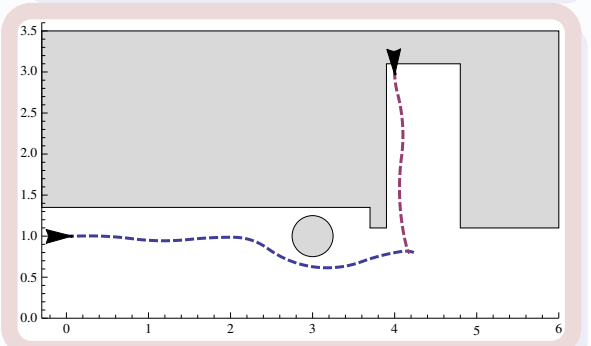

## 2: Follow the Leader




## 4: Obstacles

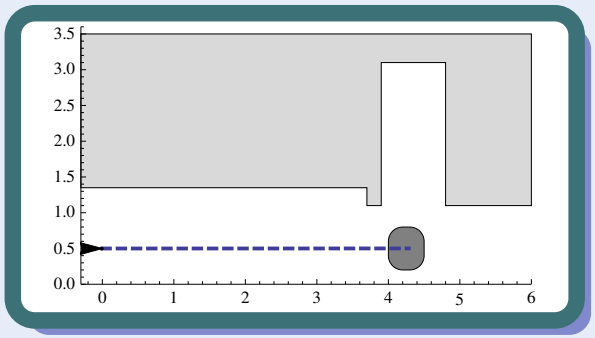


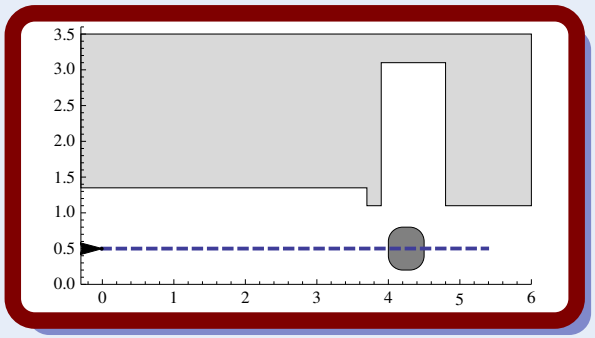




- ✓ Design, model
- ✓ Verify

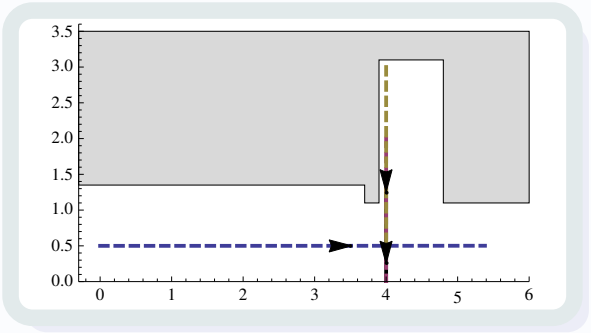
# LFCPS Betabot & Veribot Labs

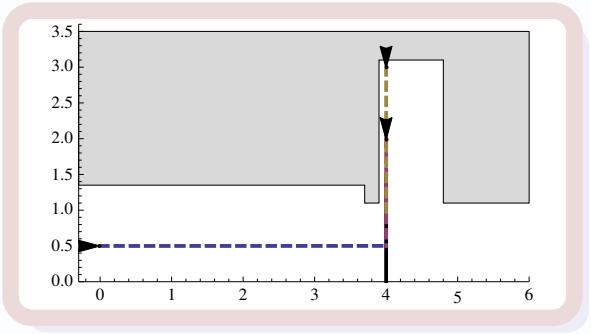
## 1: Charging Station



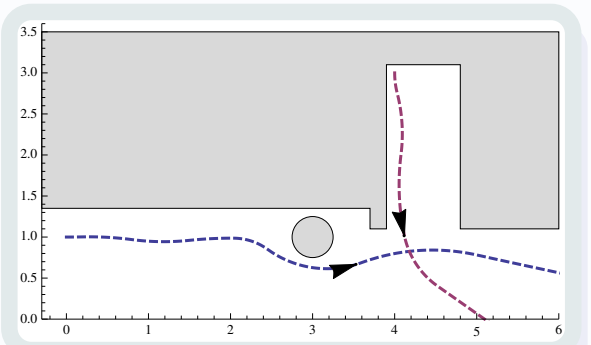



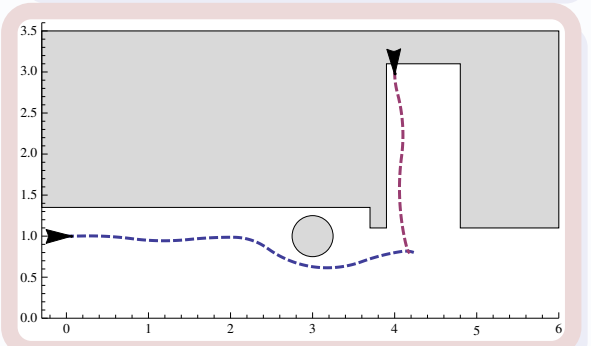

## 2: Follow the Leader




## 4: Obstacles

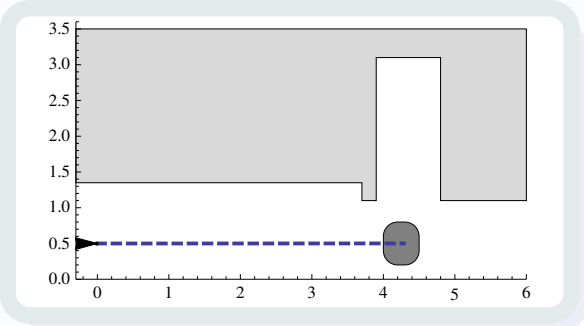


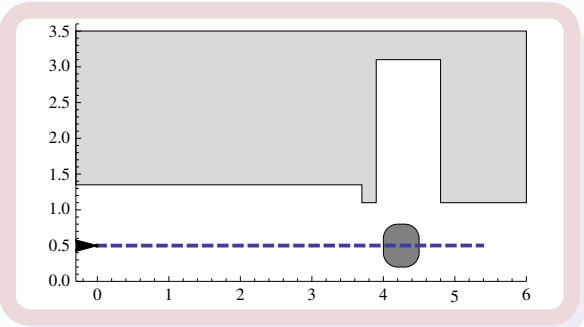




- ✓ Design, model
- ✓ Verify

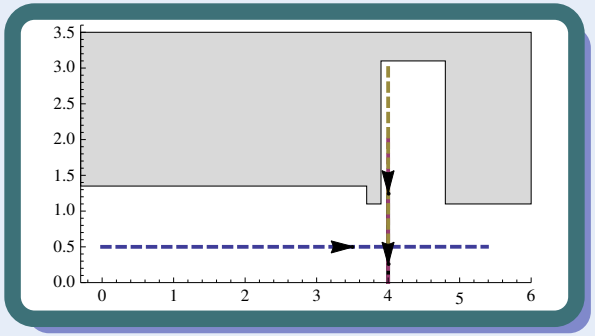
# LFCPS Betabot & Veribot Labs

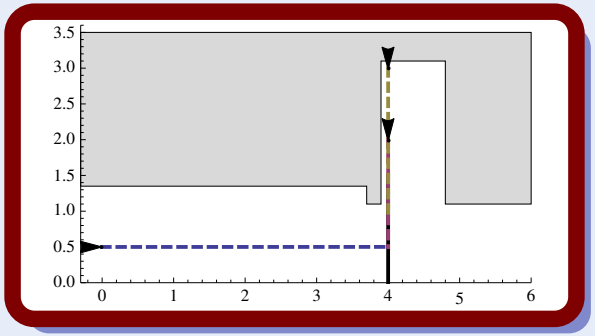
## 1: Charging Station



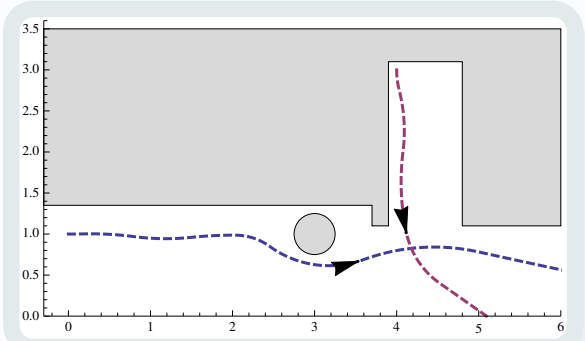



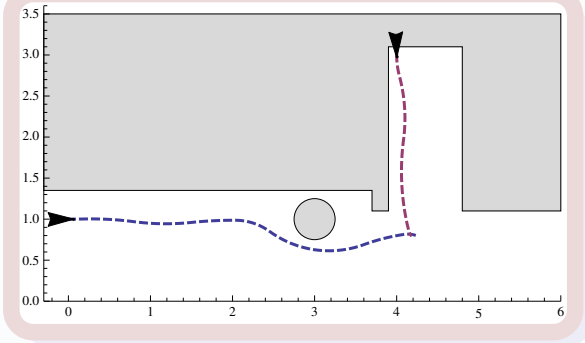

## 2: Follow the Leader




## 4: Obstacles

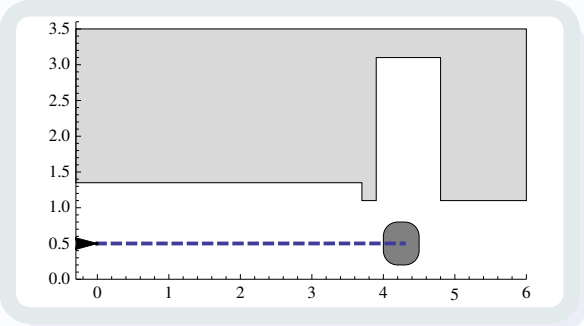


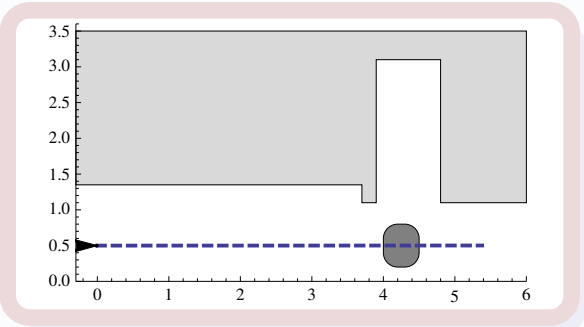




- ✓ Design, model
- ✓ Verify

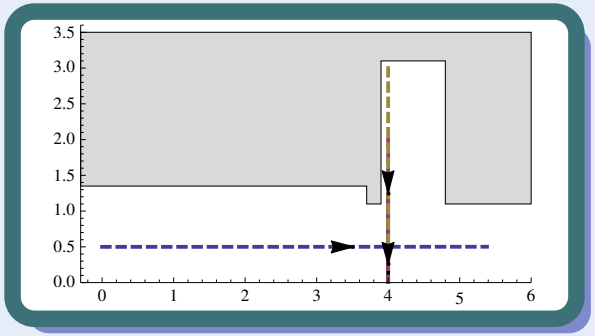
# LFCPS Betabot & Veribot Labs

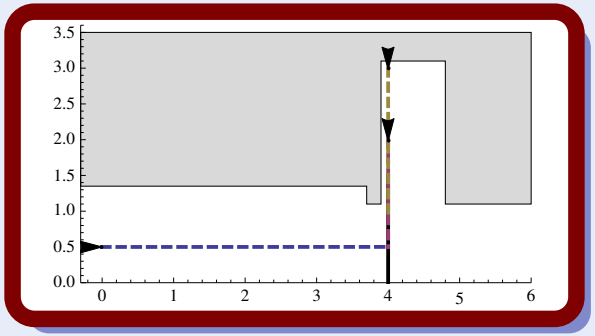
## 1: Charging Station



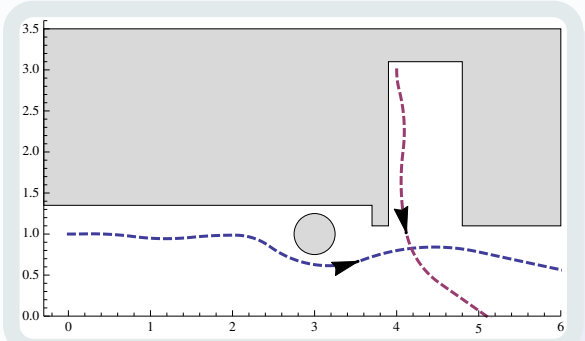



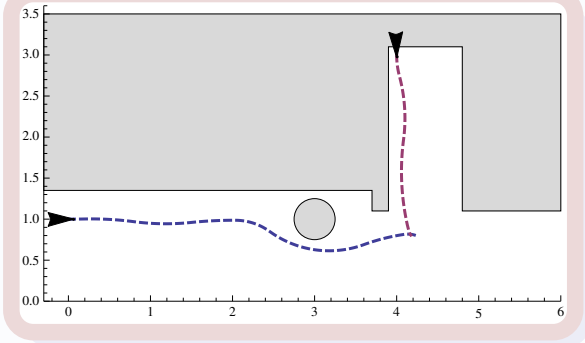

## 2: Follow the Leader




## 4: Obstacles

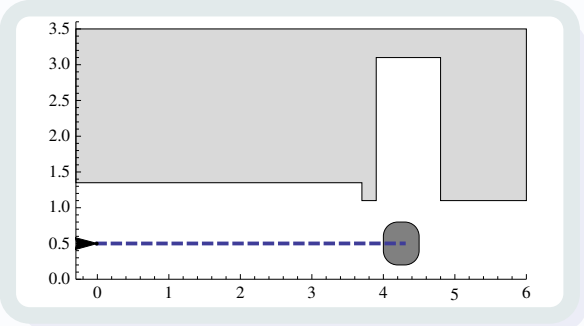


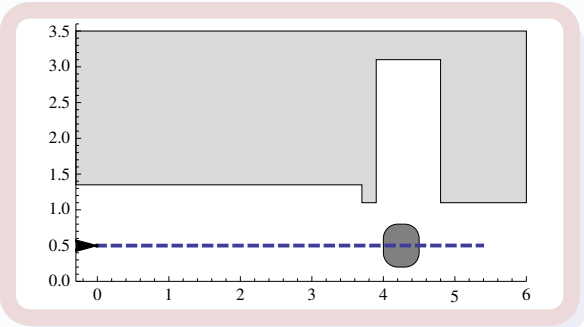




- ✓ Design, model
- ✓ Verify

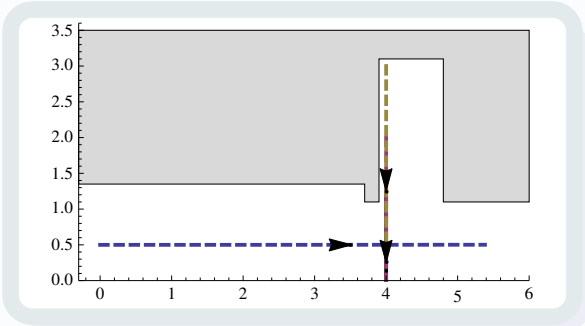
# LFCPS Betabot & Veribot Labs

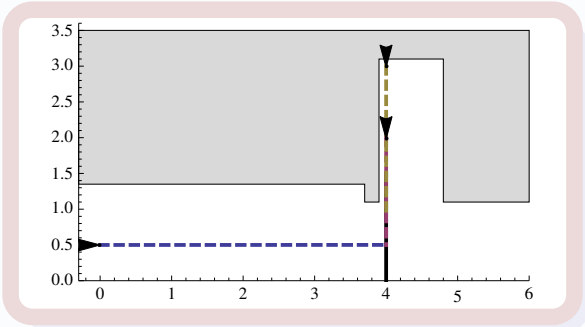
## 1: Charging Station



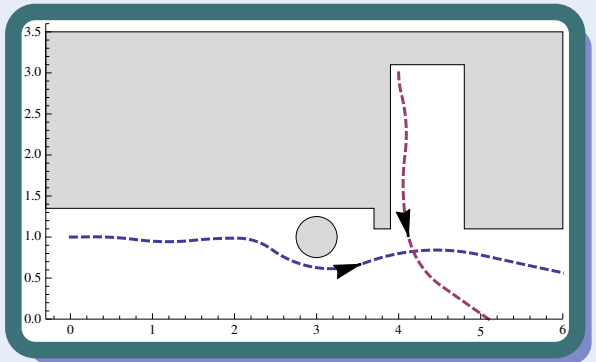



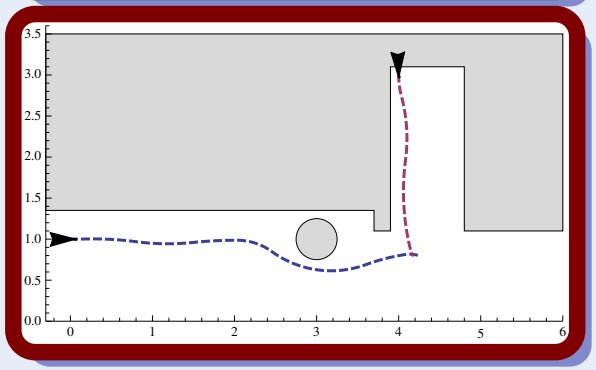

## 2: Follow the Leader




## 4: Obstacles

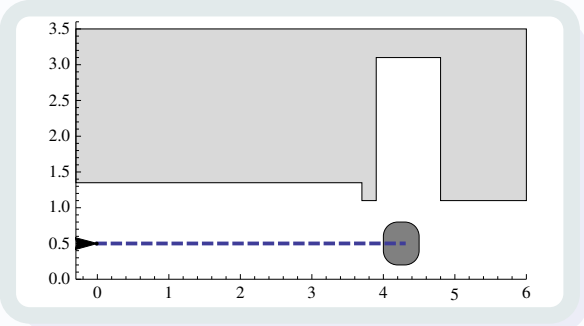


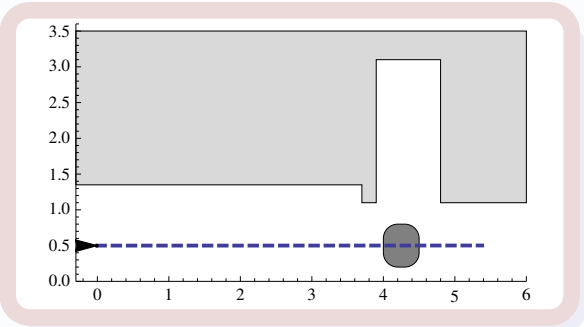




- ✓ Design, model
- ✓ Verify

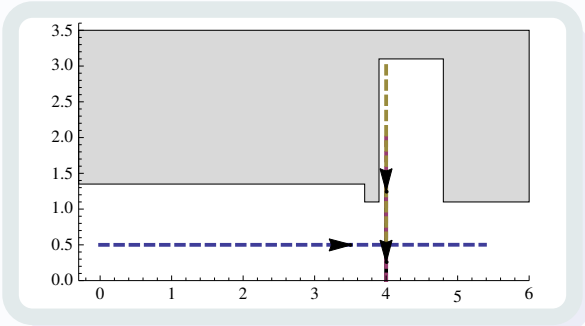
# LFCPS Betabot & Veribot Labs

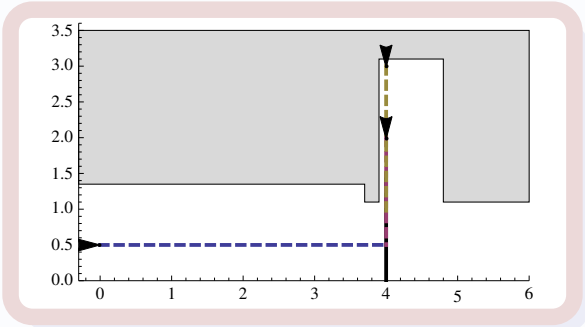
## 1: Charging Station



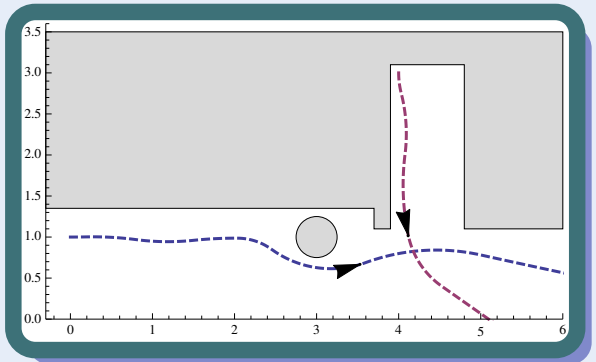



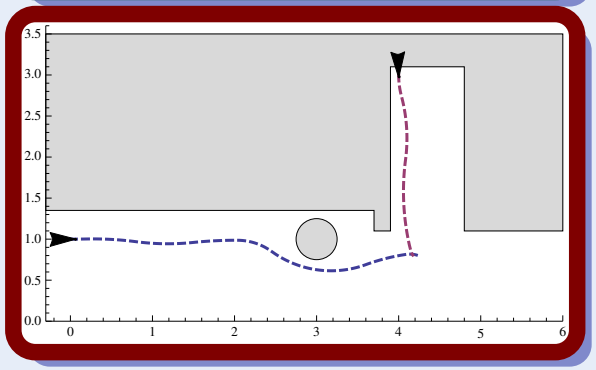

## 2: Follow the Leader




## 4: Obstacles

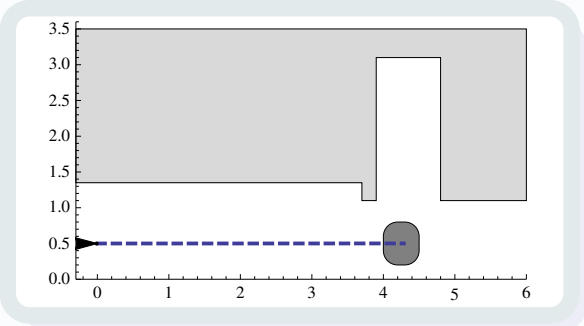


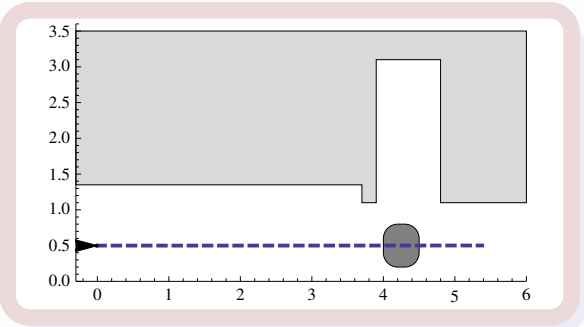




- ✓ Design, model
- ✓ Verify

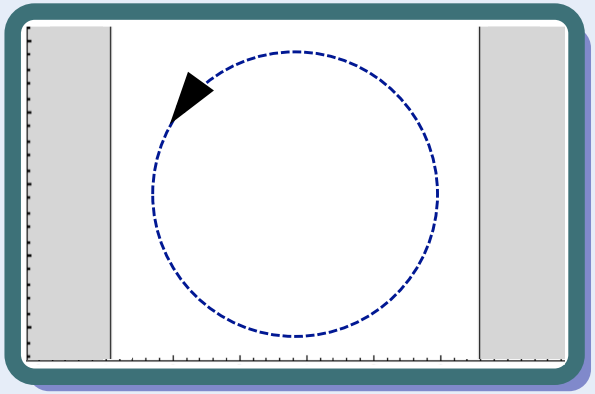
# LFCPS Betabot & Veribot Labs

## 1: Charging Station

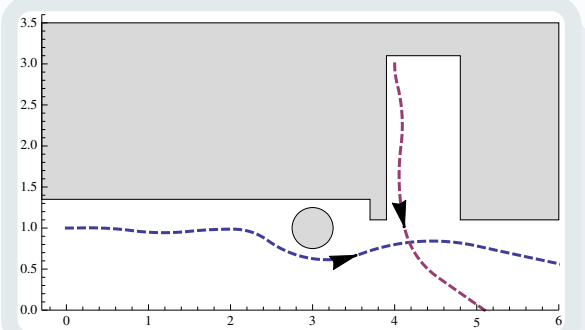


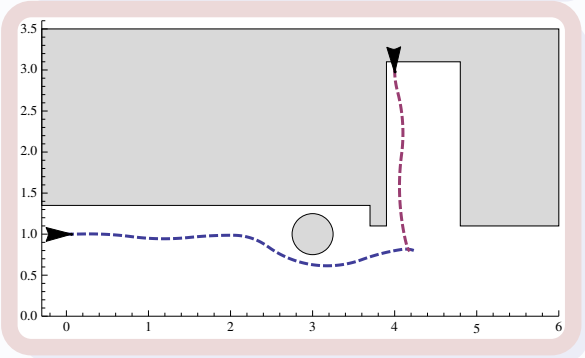




## 3: Racetrack



## 4: Obstacles



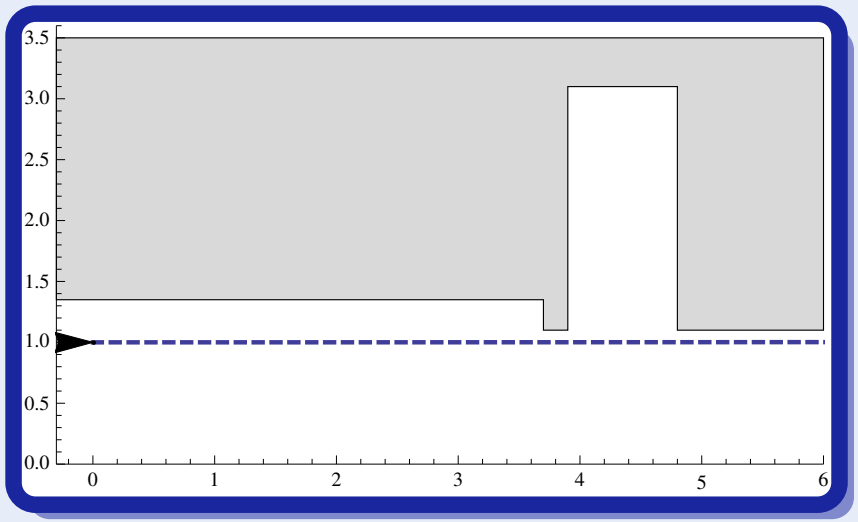



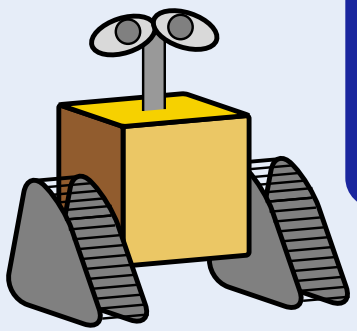

- ✓ Design, model
- ✓ Verify

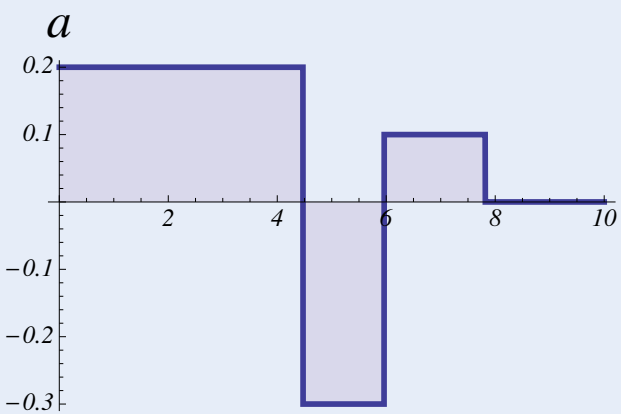
## Challenge (Hybrid Systems)

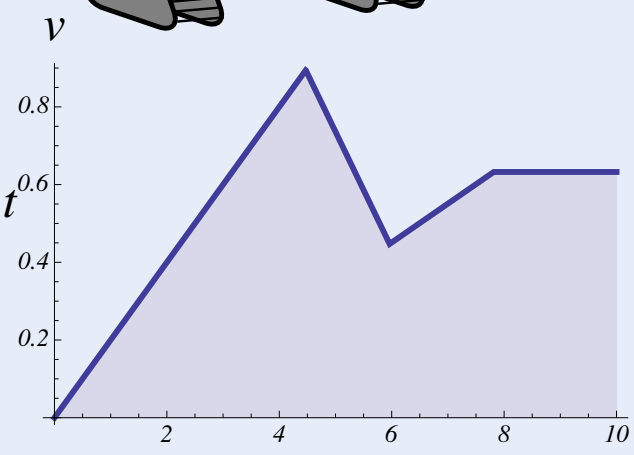
Design & verify controller for a robot avoiding obstacles

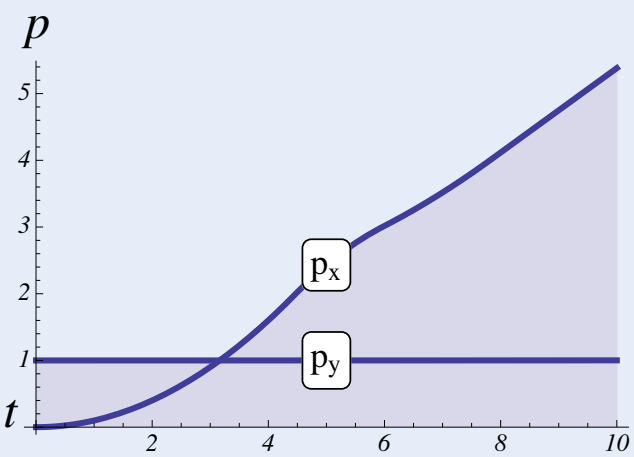
- Accelerate / brake (discrete dynamics)
- 1D motion (continuous dynamics)





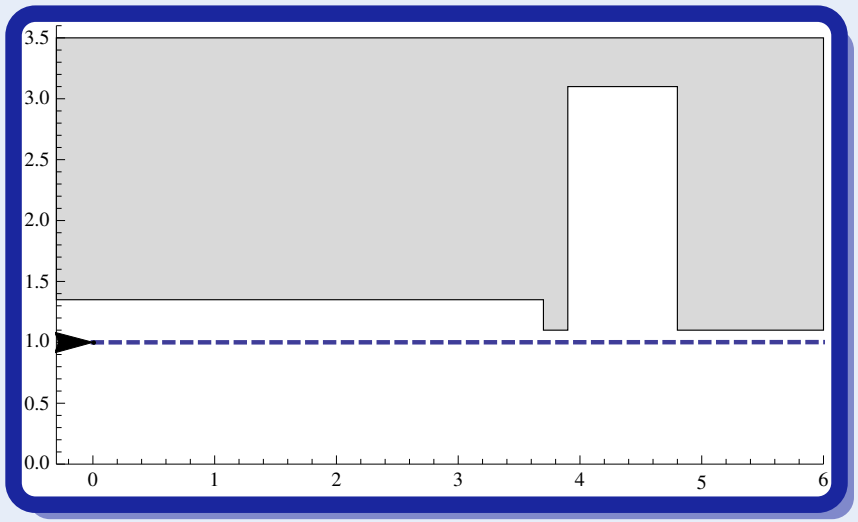



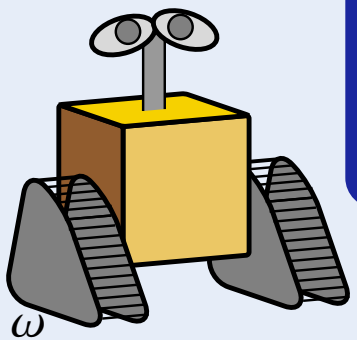



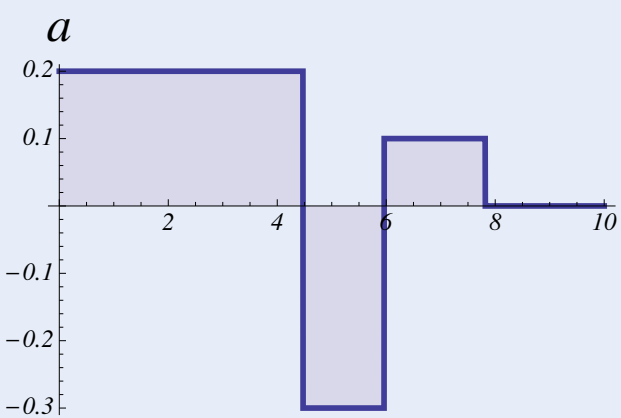
## Challenge (Hybrid Systems)

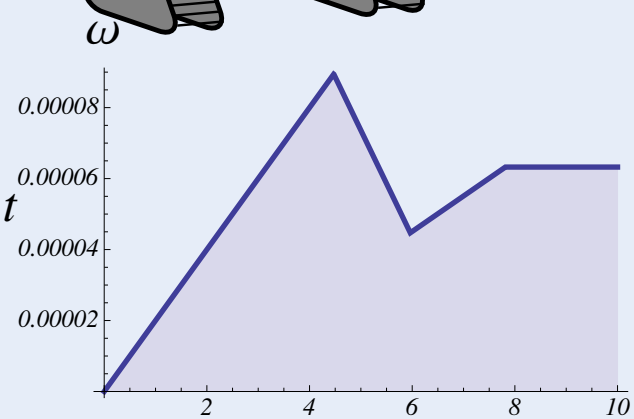
Design & verify controller for a robot avoiding obstacles

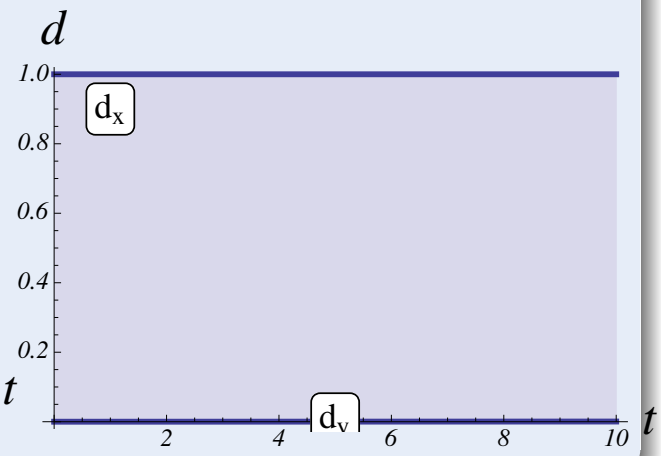
- Accelerate / brake (discrete dynamics)
- 1D motion (continuous dynamics)





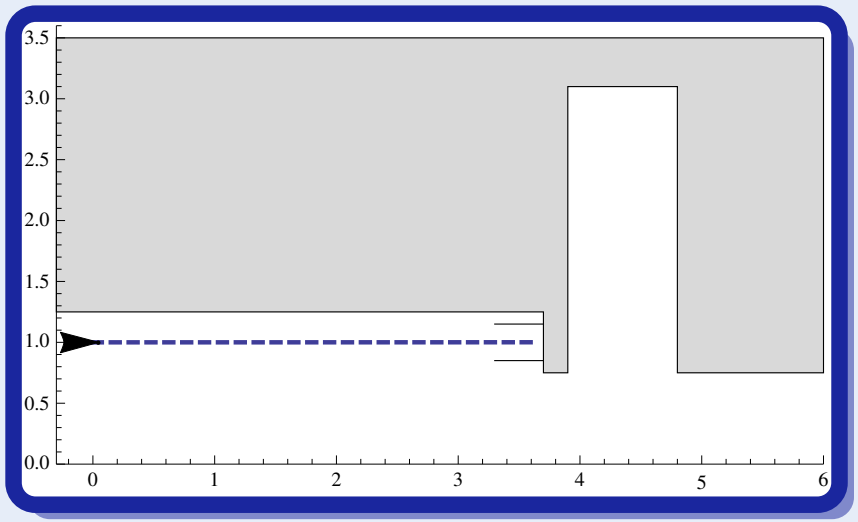



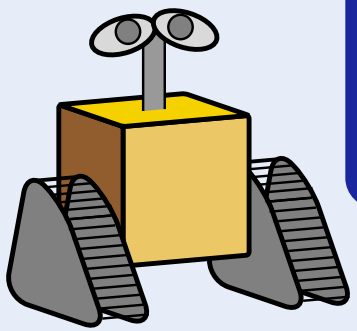



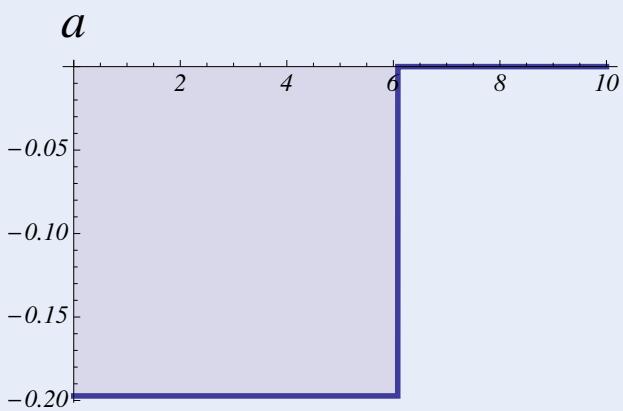
## Challenge (Hybrid Systems)

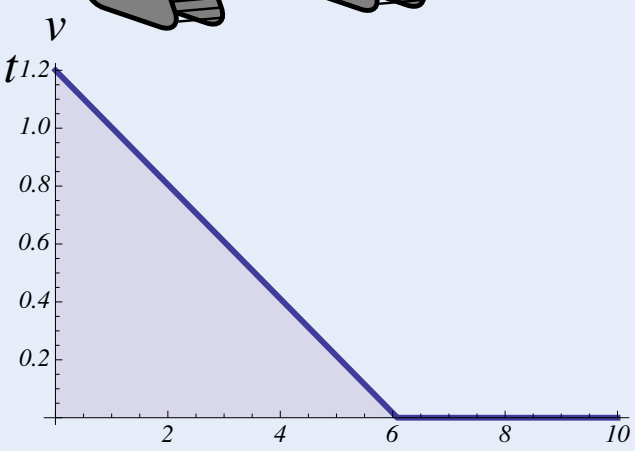
Design & verify controller for a robot avoiding obstacles

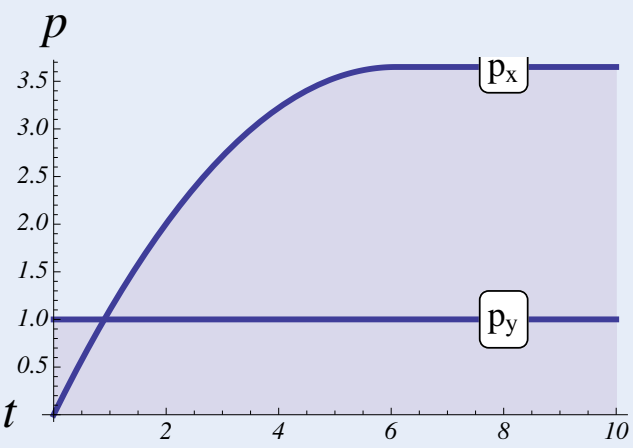
- Accelerate / brake / stop (discrete dynamics)
- 1D motion (continuous dynamics)





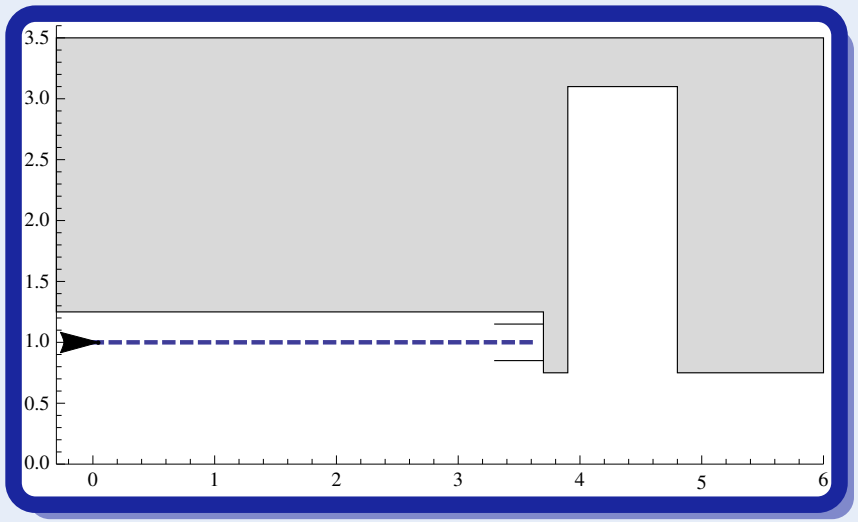



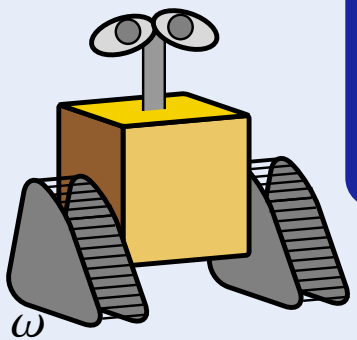



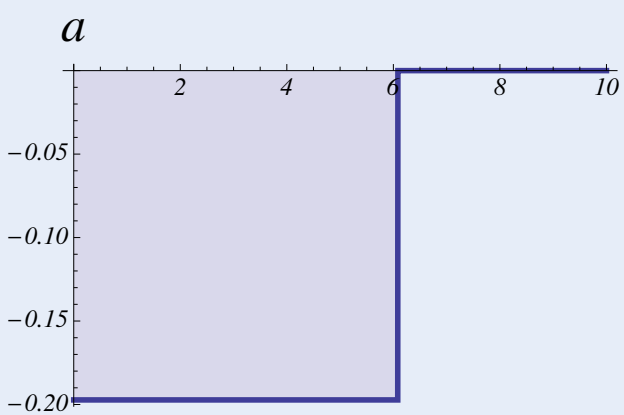
## Challenge (Hybrid Systems)

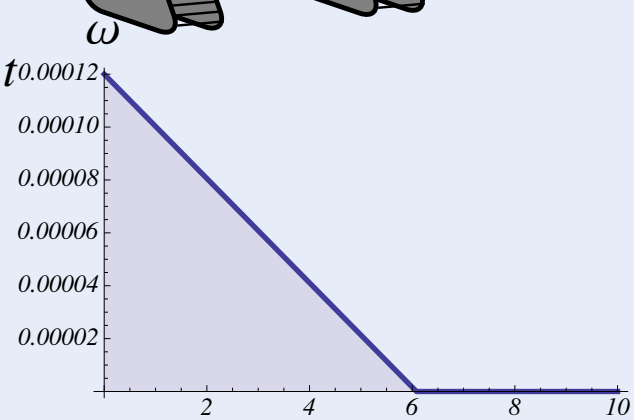
Design & verify controller for a robot avoiding obstacles

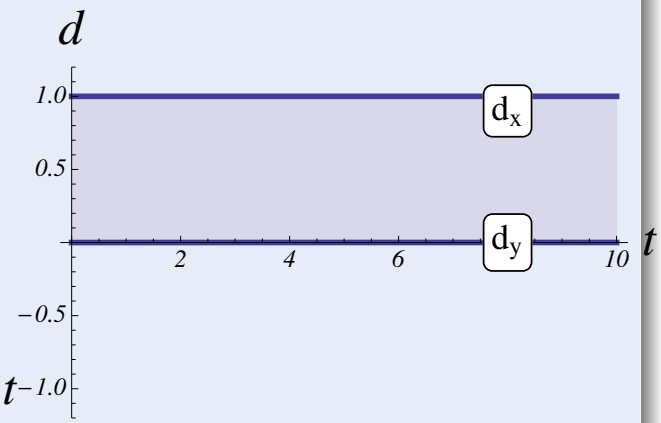
- Accelerate / brake / stop (discrete dynamics)
- 1D motion (continuous dynamics)





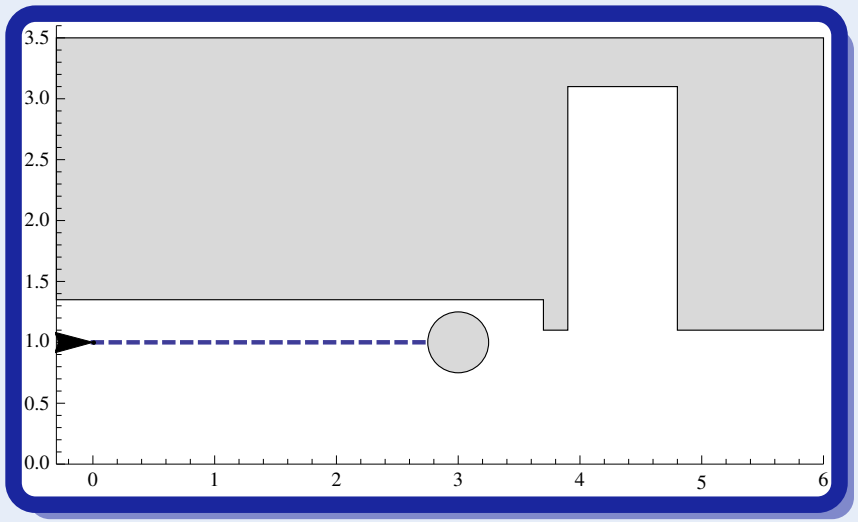



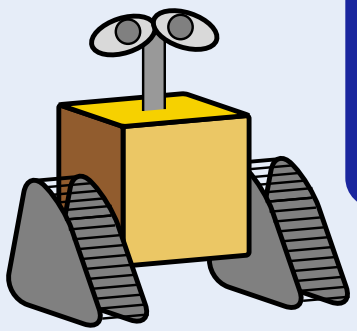



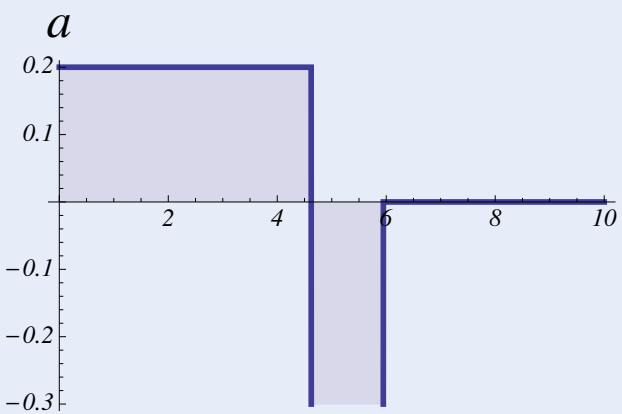
## Challenge (Hybrid Systems)

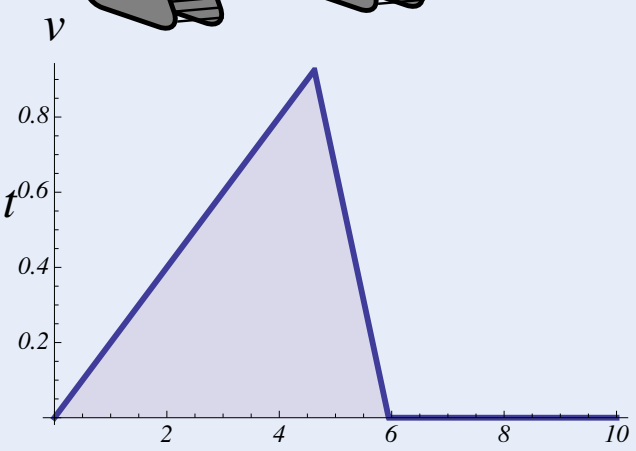
Design & verify controller for a robot avoiding obstacles

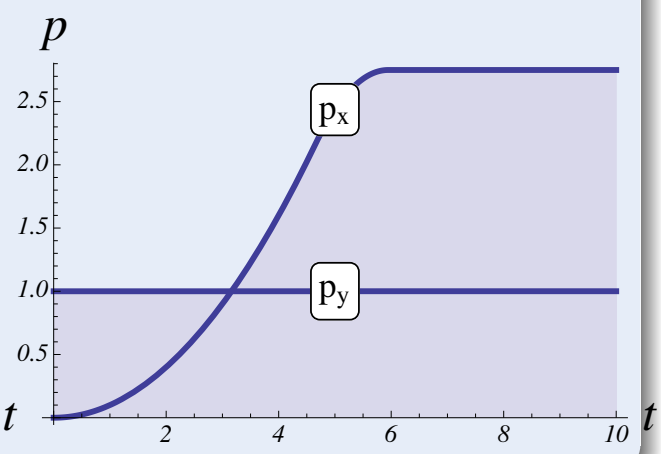
- Accelerate / brake (discrete dynamics)
- 1D motion (continuous dynamics)





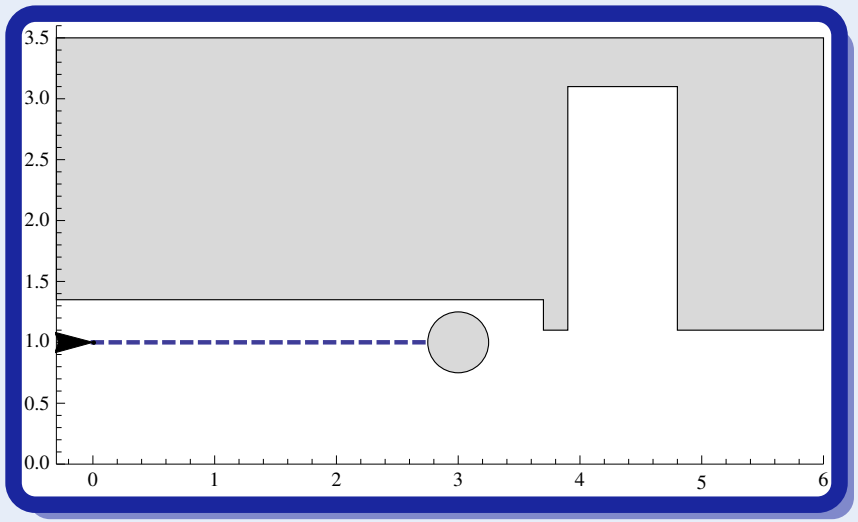



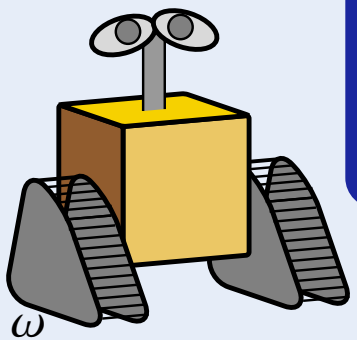



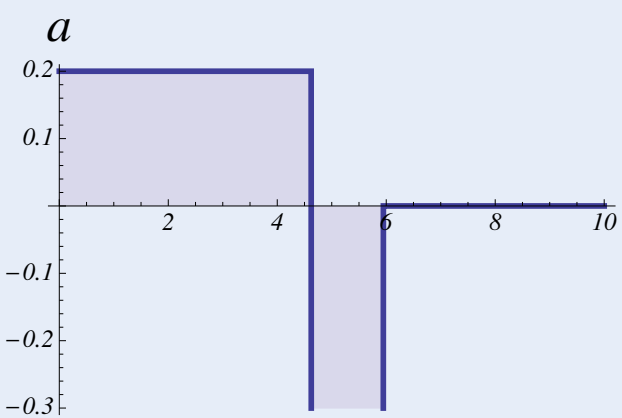
## Challenge (Hybrid Systems)

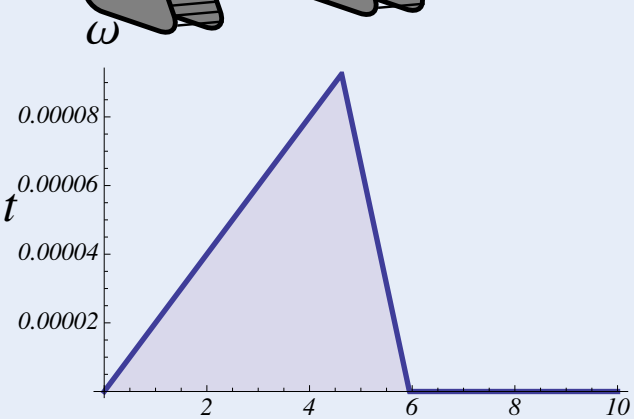
Design & verify controller for a robot avoiding obstacles

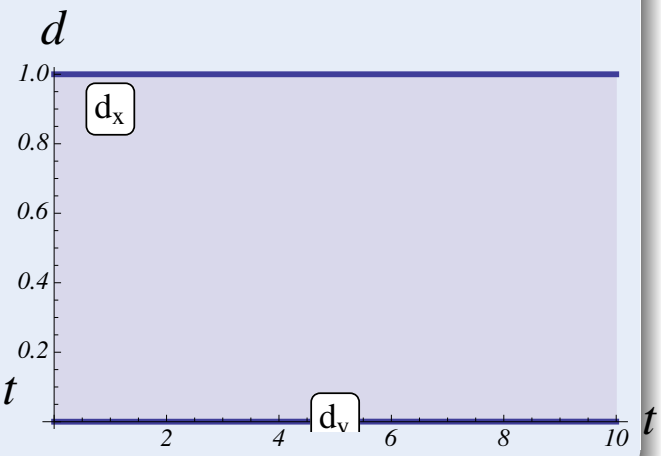
- Accelerate / brake (discrete dynamics)
- 1D motion (continuous dynamics)





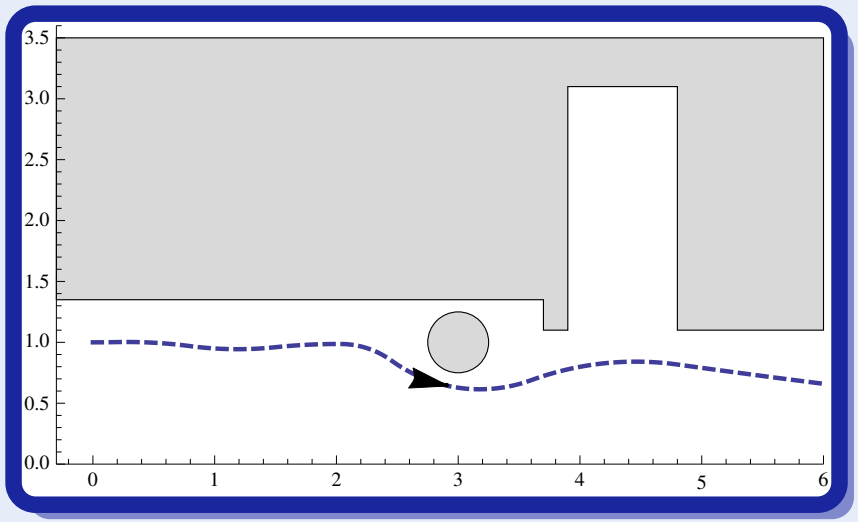



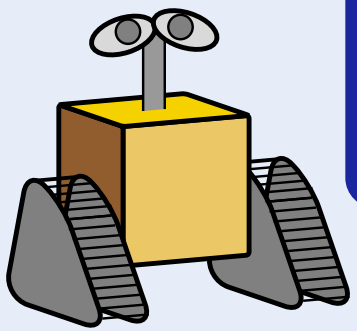



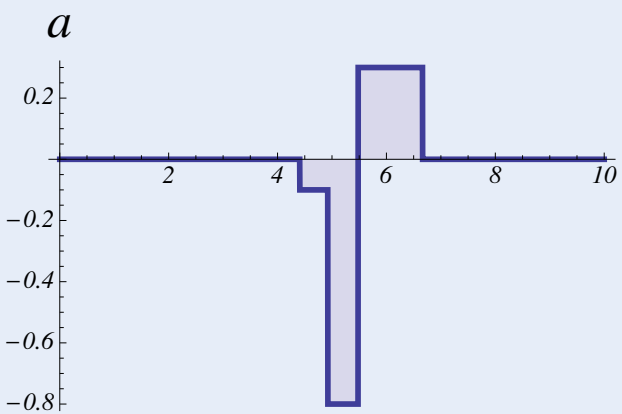
## Challenge (Hybrid Systems)

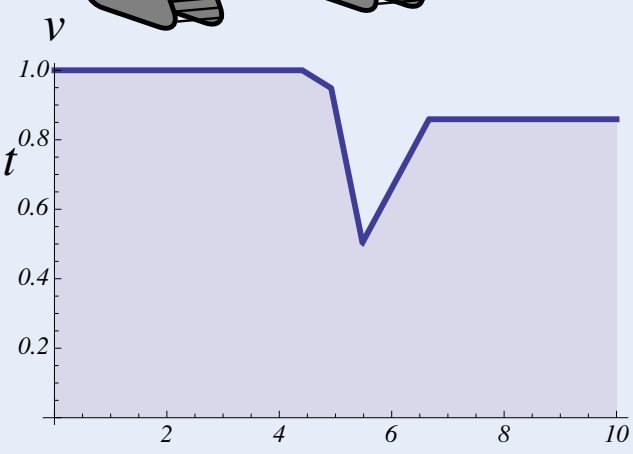
Design & verify controller for a robot avoiding obstacles

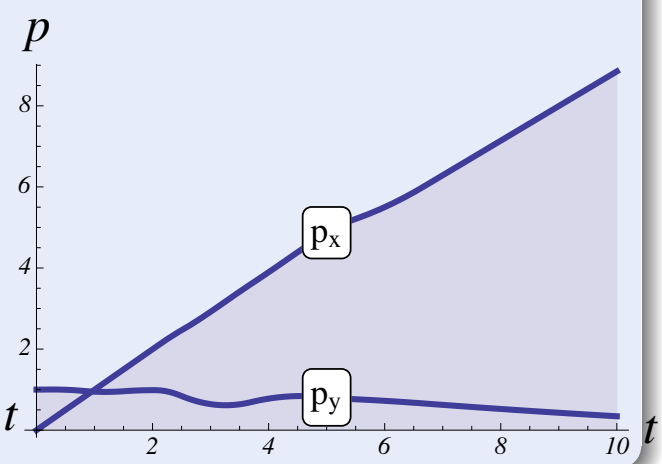
- Accel / brake / steer (discrete dynamics)
- 2D motion (continuous dynamics)





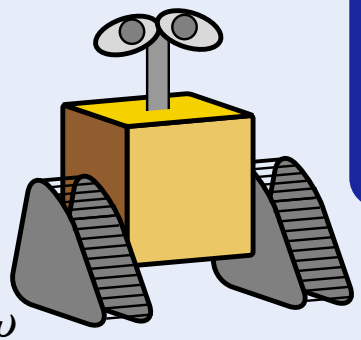



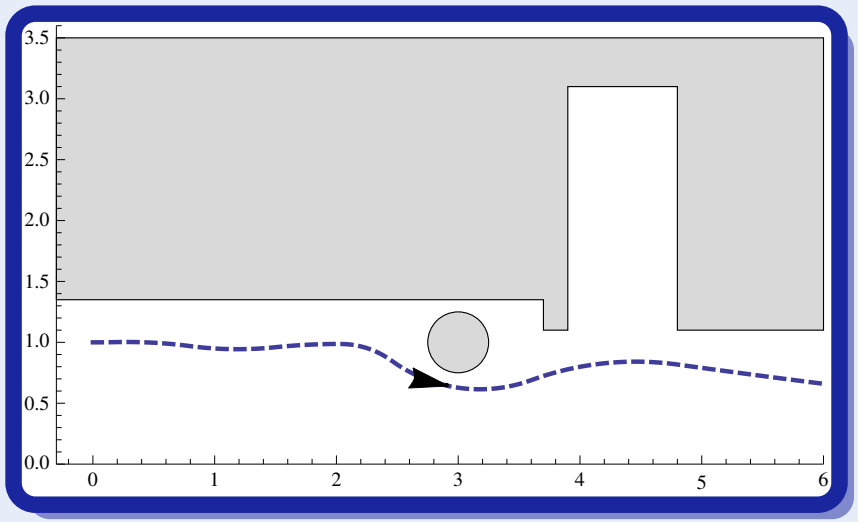



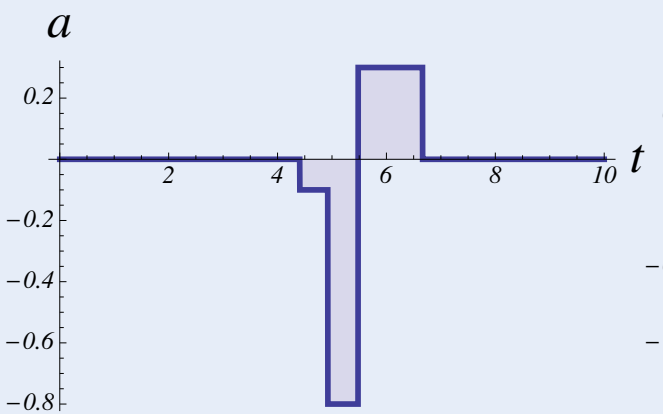
## Challenge (Hybrid Systems)

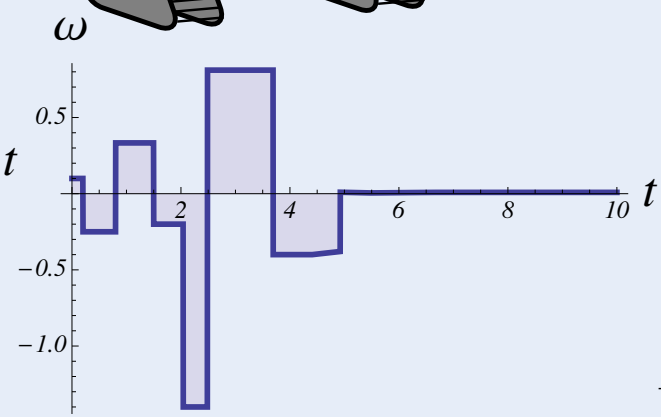
Design & verify controller for a robot avoiding obstacles

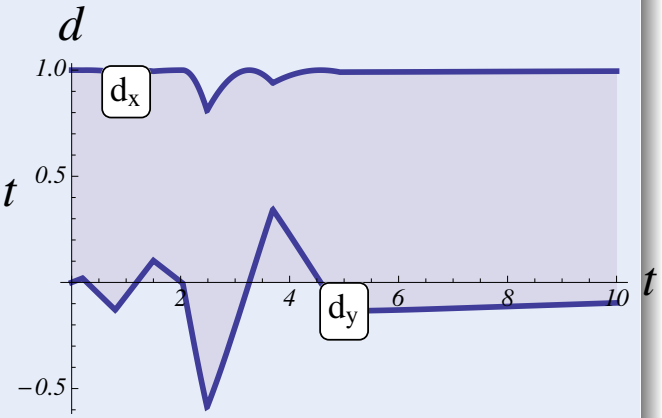
- Accel / brake / steer (discrete dynamics)
- 2D motion (continuous dynamics)





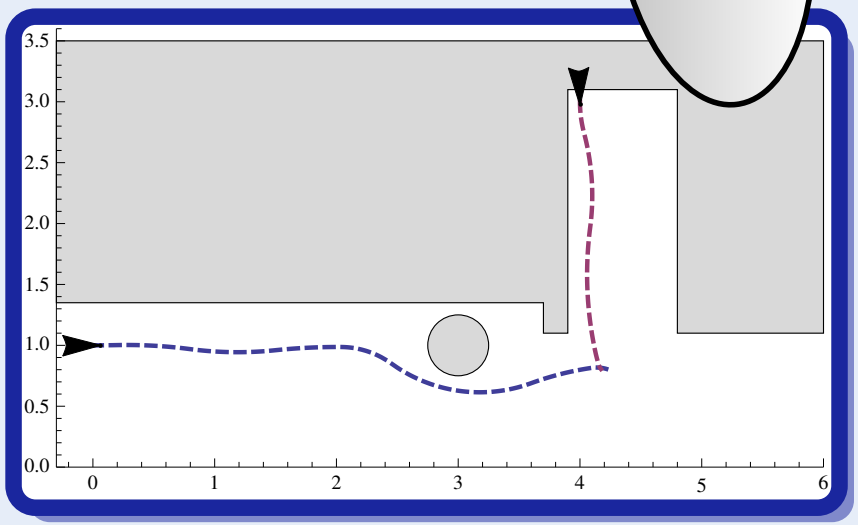



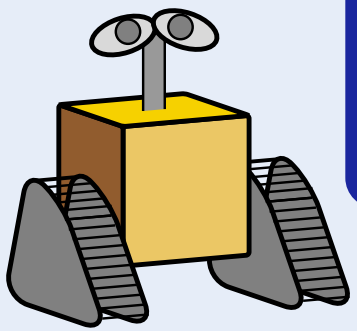



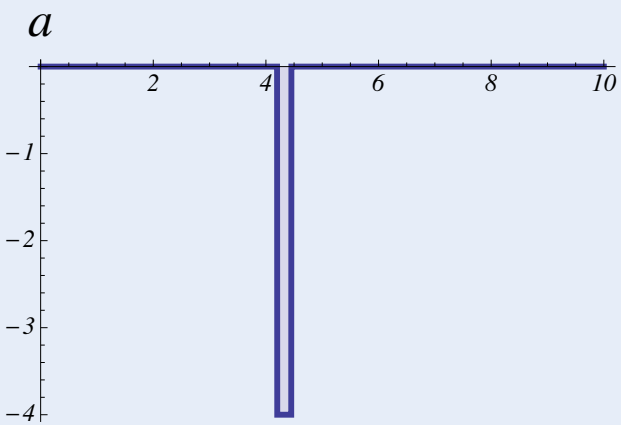
## Challenge (Hybrid Systems)

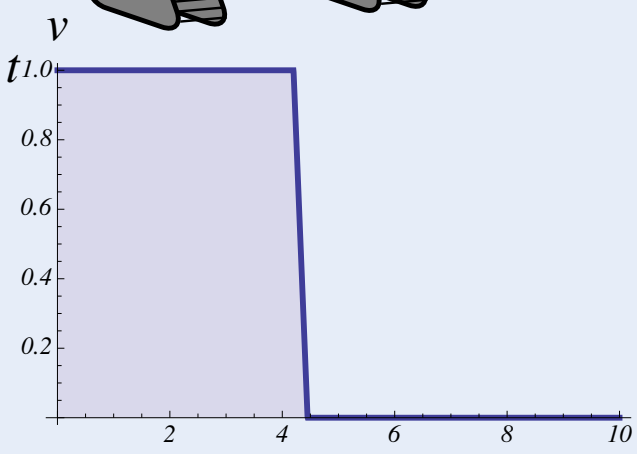
Design & verify controller for a robot avoiding obstacles

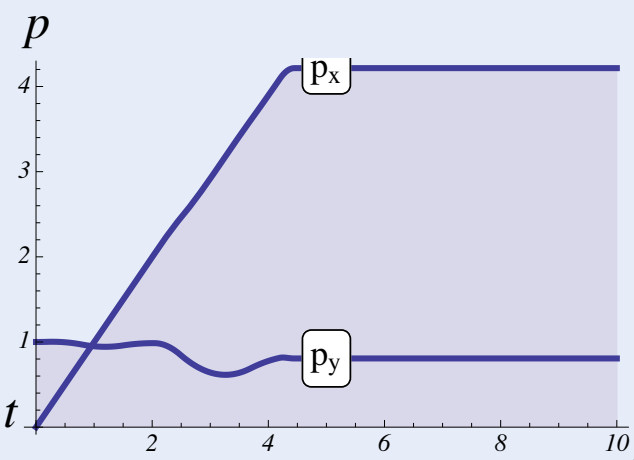
- Dynamic obstacles (other agents)
- Avoid collisions (define safety)





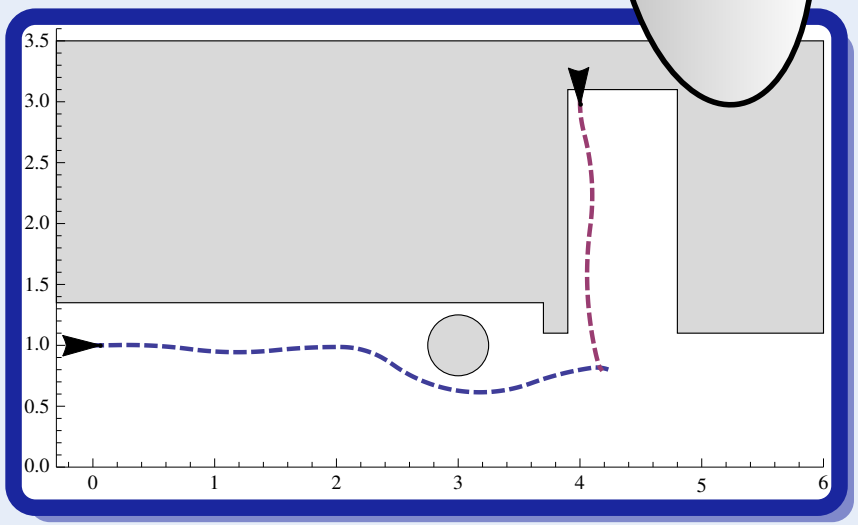



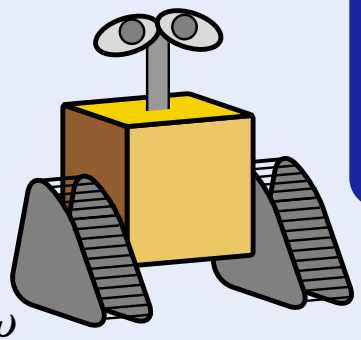



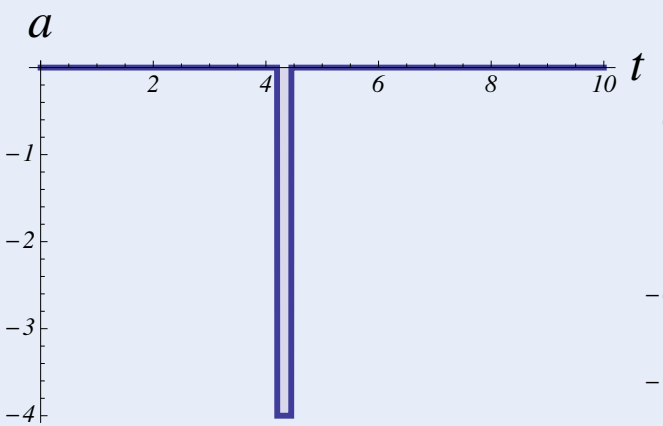
## Challenge (Hybrid Systems)

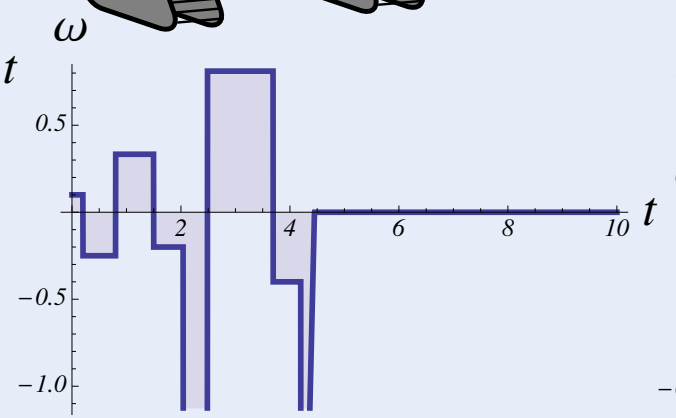
Design & verify controller for a robot avoiding obstacles

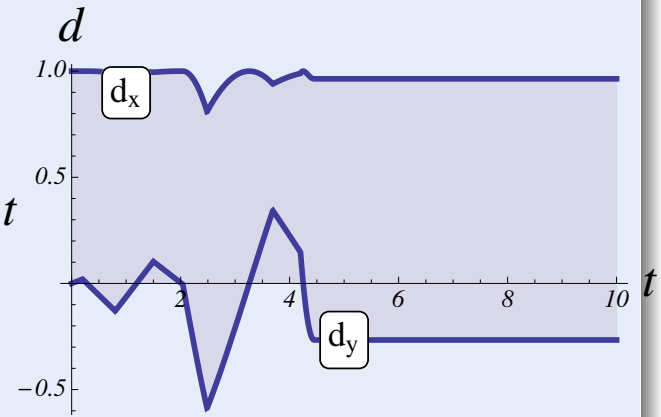
- Dynamic obstacles (other agents)
- Avoid collisions (define safety)





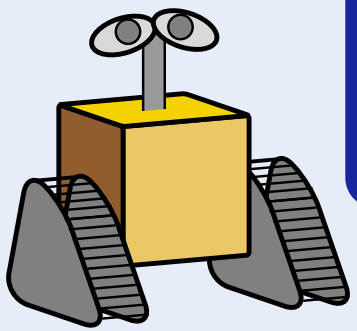



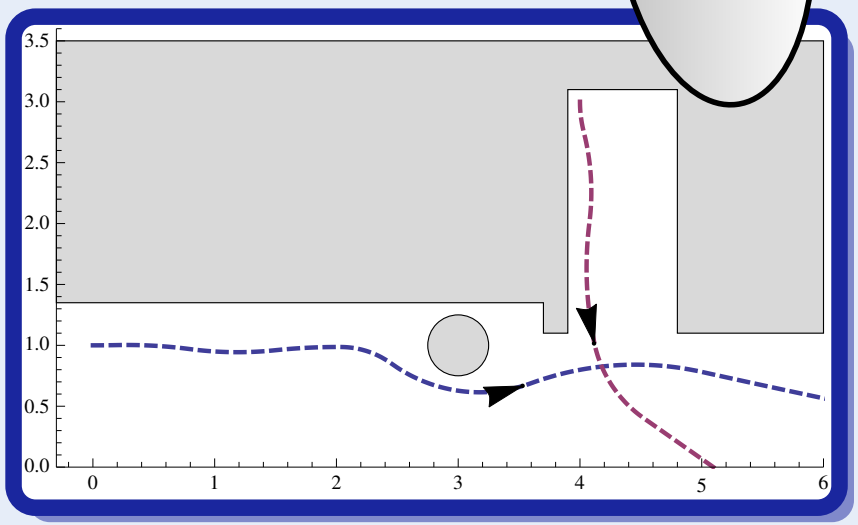



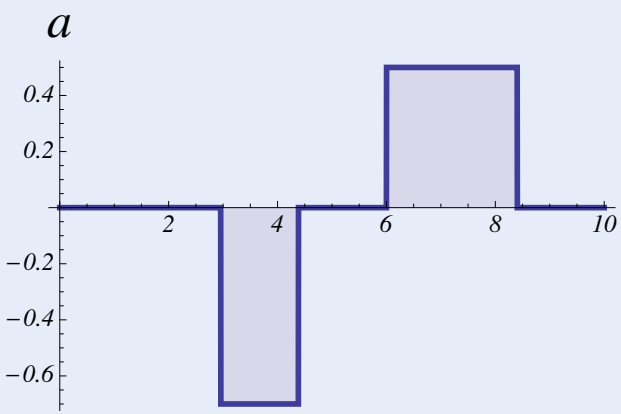
## Challenge (Hybrid Systems)

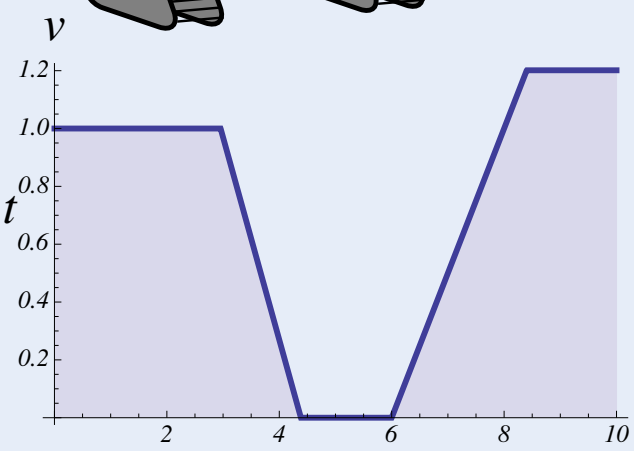
Design & verify controller for a robot avoiding obstacles

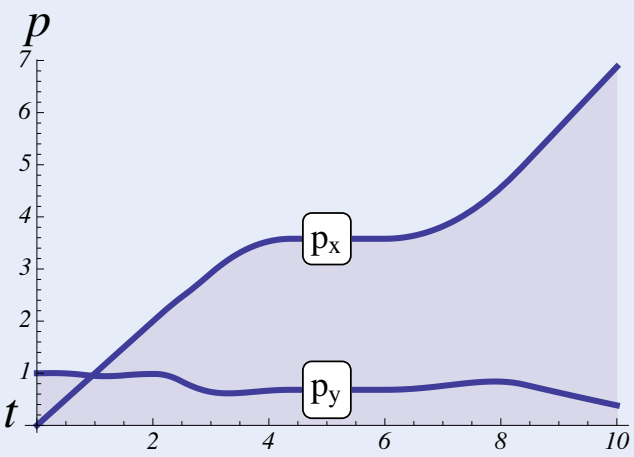
- Control robot (respect delays)
- Environment interaction (obstacles, agents, uncertainty)





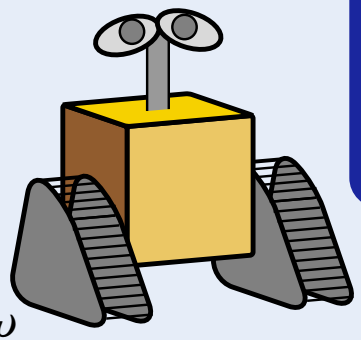



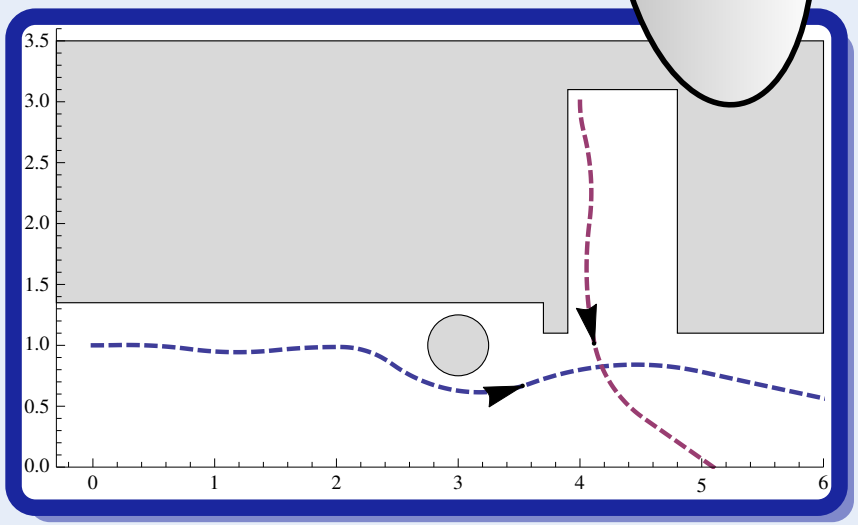



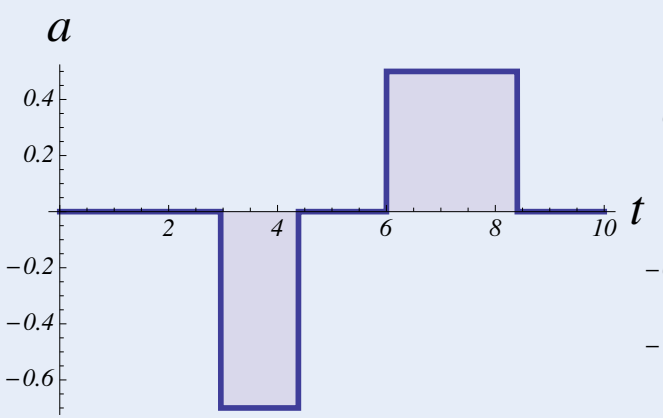
## Challenge (Hybrid Systems)

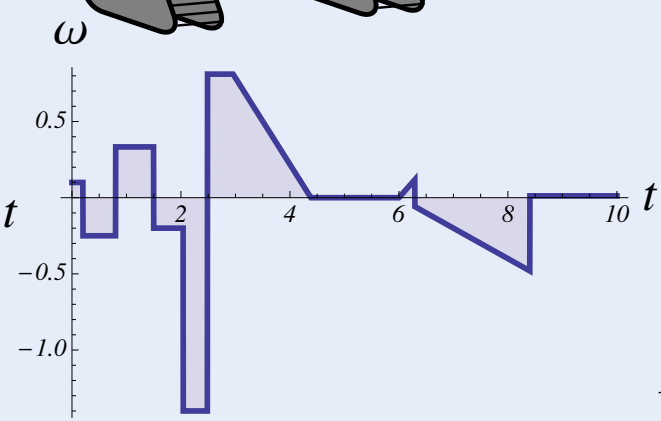
Design & verify controller for a robot avoiding obstacles

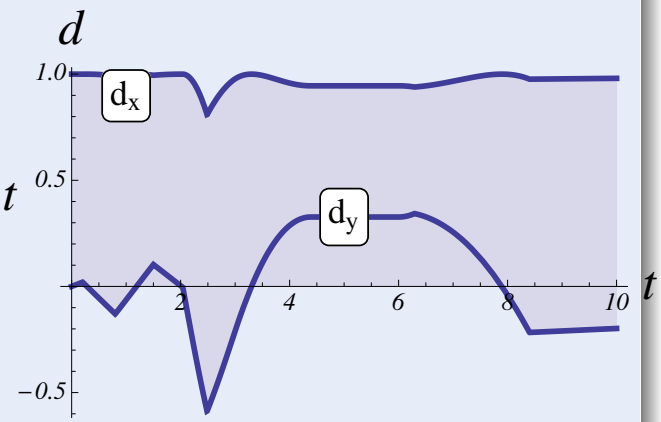
- Control robot (respect delays)
- Environment interaction (obstacles, agents, uncertainty)





# Active Learning Quizzes

- Autograded learning by doing
- Think through subtle aspects at student's own pace
- Ranging from subskill and skill to synthetic skill
- Get feedback what to review & practice before misunderstanding stacks
- Multiple-choice easy to design & grade but **educational P-NP problem**
- Checking answers easier than producing correct answers (used twice!)

What is the indefinite integral of $\sin x$?

# Active Learning Quizzes

- Autograded learning by doing
- Think through subtle aspects at student's own pace
- Ranging from subskill and skill to synthetic skill
- Get feedback what to review & practice before misunderstanding stacks
- Multiple-choice easy to design & grade but **educational P-NP problem**
- Checking answers easier than producing correct answers (used twice!)

What is the indefinite integral of $\sin x$?

ⓐ $\ln x$ ⓑ $\cos x$ ⓒ $\sin x$ ⓓ $-\cos x$ ⓔ $-\sin x$ ⓕ $1/x$

# Active Learning Quizzes

- Autograded learning by doing
- Think through subtle aspects at student's own pace
- Ranging from subskill and skill to synthetic skill
- Get feedback what to review & practice before misunderstanding stacks
- Multiple-choice easy to design & grade but **educational P-NP problem**
- Checking answers easier than producing correct answers (used twice!)

What is the indefinite integral of $\sin x$? Multiple-choice teaches differentiation!

ⓐ $\ln x$ ⓑ $\cos x$ ⓒ $\sin x$ ⓓ $-\cos x$ ⓔ $-\sin x$ ⓕ $1/x$

# Active Learning Quizzes

- Autograded learning by doing
- Think through subtle aspects at student's own pace
- Ranging from subskill and skill to synthetic skill
- Get feedback what to review & practice before misunderstanding stacks
- Multiple-choice easy to design & grade but **educational P-NP problem**
- Checking answers easier than producing correct answers (used twice!)

What is the indefinite integral of $\sin x$? Multiple-choice teaches differentiation!

ⓐ $\ln x$ ⓑ $\cos x$ ⓒ $\sin x$ ⓓ $-\cos x$ ⓔ $-\sin x$ ⓕ $1/x$

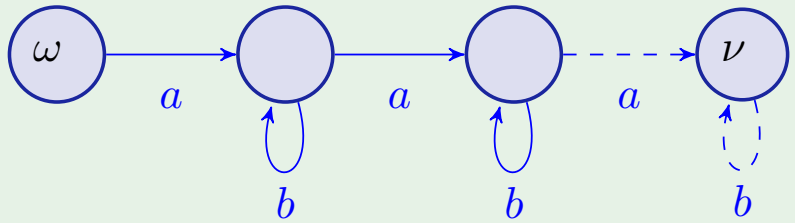
- Instead: face blank page syndrome on every question
- Free text easy but cannot autograde accurately
- Free program and formula input, autograded by KeYmaera X proof
- Succinctly convey a lot of understanding

# Example Quiz

## Example (Quiz: Program shapes)

Objective (*programming languages for CPS, semantics, models, operational effects*): It is crucial to obtain an intuitive reading of the respective transitions in a hybrid program. This question gives you an opportunity to practice the mapping between a transition structure and the hybrid programs they correspond to.

What hybrid program fits to the following transition structure?



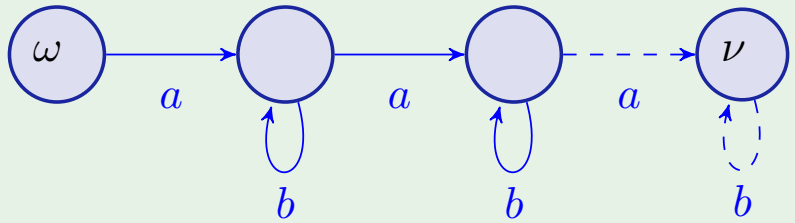

**Answer:**

# Example Quiz

## Example (Quiz: Program shapes)

Objective (*programming languages for CPS, semantics, models, operational effects*): It is crucial to obtain an intuitive reading of the respective transitions in a hybrid program. This question gives you an opportunity to practice the mapping between a transition structure and the hybrid programs they correspond to.

What hybrid program fits to the following transition structure?




**Answer:** `{a;{b;}*}*`

Inverts definitions, requires reflection, and practices deeper understandings of the inner working principles.

# Example Quiz

## Example (Quiz: True formulas)

Objective (*model semantics, preconditions, rigorous specification*): If a formula is not valid, it is important to identify when exactly it is true. This helps identify missing preconditions to make it valid, and read off consequences when a formula is available as an assumption. Of course, knowing when exactly a formula is true is also crucial when they are used as evolution domain constraints or tests, which is why those are usually quantifier-free FOL formulas.

**Question:** When is dL formula $[x := x + 1]\, x > 5$ true?

**Answer:**

**Question:** When is dL formula $[x' = v, v' = a]\, x \leq m$ true?

**Answer:**

# Example Quiz

## Example (Quiz: True formulas)

Objective (*model semantics, preconditions, rigorous specification*): If a formula is not valid, it is important to identify when exactly it is true. This helps identify missing preconditions to make it valid, and read off consequences when a formula is available as an assumption. Of course, knowing when exactly a formula is true is also crucial when they are used as evolution domain constraints or tests, which is why those are usually quantifier-free FOL formulas.

**Question:** When is dL formula $[x := x+1]\, x > 5$ true?

**Answer:** `x>4`

**Question:** When is dL formula $[x' = v, v' = a]\, x \leq m$ true?

**Answer:**

# Example Quiz

## Example (Quiz: True formulas)

Objective (*model semantics, preconditions, rigorous specification*): If a formula is not valid, it is important to identify when exactly it is true. This helps identify missing preconditions to make it valid, and read off consequences when a formula is available as an assumption. Of course, knowing when exactly a formula is true is also crucial when they are used as evolution domain constraints or tests, which is why those are usually quantifier-free FOL formulas.

**Question:** When is dL formula $[x := x + 1]\, x > 5$ true?

**Answer:** `x>4`

**Question:** When is dL formula $[x' = v, v' = a]\, x \leq m$ true?

**Answer:**

Check physical intuition as a logical formula, prepare for safe veribot design

# Example Quiz

## Example (Quiz: Axiom usage)

Objective (*rigorous reasoning about CPS*): As one important part of rigorous reasoning about CPS, you will practice the correct application of axioms to differential dynamic logic problems. While the KeYmaera X prover correctly applies axioms for you, it is still helpful if you practice this yourself to get a better intuition for how it works and predict the outcome of a proof step before trying it. That will make you more time-efficient in your reasoning. It will also inform you how to transform parts of a proof to make useful axioms applicable later. If you properly understand reasoning principles, you are also better able to identify and check clever problem decompositions.

**Question:** What is the result of using axiom [;] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:**

**Question:** What is the result of using axiom [:=] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:**

**Question:** What is the result of using axiom [;] on $[\text{sense};\text{ctrl};\text{plant}]\, x > y$

**Answer:**

# Example Quiz

## Example (Quiz: Axiom usage)

Objective (*rigorous reasoning about CPS*): As one important part of rigorous reasoning about CPS, you will practice the correct application of axioms to differential dynamic logic problems. While the KeYmaera X prover correctly applies axioms for you, it is still helpful if you practice this yourself to get a better intuition for how it works and predict the outcome of a proof step before trying it. That will make you more time-efficient in your reasoning. It will also inform you how to transform parts of a proof to make useful axioms applicable later. If you properly understand reasoning principles, you are also better able to identify and check clever problem decompositions.

**Question:** What is the result of using axiom [;] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:** `[ctrl;][plant;]x>y`

**Question:** What is the result of using axiom [:=] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:**

**Question:** What is the result of using axiom [;] on $[\text{sense};\text{ctrl};\text{plant}]\, x > y$

**Answer:**

# Example Quiz

## Example (Quiz: Axiom usage)

Objective (*rigorous reasoning about CPS*): As one important part of rigorous reasoning about CPS, you will practice the correct application of axioms to differential dynamic logic problems. While the KeYmaera X prover correctly applies axioms for you, it is still helpful if you practice this yourself to get a better intuition for how it works and predict the outcome of a proof step before trying it. That will make you more time-efficient in your reasoning. It will also inform you how to transform parts of a proof to make useful axioms applicable later. If you properly understand reasoning principles, you are also better able to identify and check clever problem decompositions.

**Question:** What is the result of using axiom [;] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:** `[ctrl;][plant;]x>y`

**Question:** What is the result of using axiom [:=] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:** `n/a`

**Question:** What is the result of using axiom [;] on $[\text{sense};\text{ctrl};\text{plant}]\, x > y$

**Answer:**

# Example Quiz

## Example (Quiz: Axiom usage)

Objective (*rigorous reasoning about CPS*): As one important part of rigorous reasoning about CPS, you will practice the correct application of axioms to differential dynamic logic problems. While the KeYmaera X prover correctly applies axioms for you, it is still helpful if you practice this yourself to get a better intuition for how it works and predict the outcome of a proof step before trying it. That will make you more time-efficient in your reasoning. It will also inform you how to transform parts of a proof to make useful axioms applicable later. If you properly understand reasoning principles, you are also better able to identify and check clever problem decompositions.

**Question:** What is the result of using axiom [;] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:** `[ctrl;][plant;]x>y`

**Question:** What is the result of using axiom [:=] on $[\text{ctrl};\text{plant}]\, x > y$

**Answer:** `n/a`

**Question:** What is the result of using axiom [;] on $[\text{sense};\text{ctrl};\text{plant}]\, x > y$

**Answer:** `[sense;][ctrl;plant;]x>y`

# Example Quiz

## Example (Quiz: Axiom development)

Objective (*operational CPS effects,* dL *as a verification language*): The axioms of differential dynamic logic are complete, so you do not need any more for its operators. But whenever you add new syntax to the language, then you give that operator a semantics, and also need to add new axioms for reasoning about the new syntactic features. These questions give you an opportunity to practice the extension of syntax, semantics, and axiomatics that fit together in harmony and properly decompose hybrid programs into logic. Recall that it is imperative that only sound axioms be adopted. ...

**Question:** Develop an axiom for $[\text{if}(Q)\,a\,\text{else}\,b]P$ that decomposes the effect of the if-then-else statement in logic with simpler logical connectives.

**Answer:**

# Example Quiz

## Example (Quiz: Axiom development)

Objective (*operational CPS effects,* dL *as a verification language*): The axioms of differential dynamic logic are complete, so you do not need any more for its operators. But whenever you add new syntax to the language, then you give that operator a semantics, and also need to add new axioms for reasoning about the new syntactic features. These questions give you an opportunity to practice the extension of syntax, semantics, and axiomatics that fit together in harmony and properly decompose hybrid programs into logic. Recall that it is imperative that only sound axioms be adopted. ...

**Question:** Develop an axiom for $[\text{if}(Q)\,a\,\text{else}\,b]P$ that decomposes the effect of the if-then-else statement in logic with simpler logical connectives.

**Answer:** $[\text{if}(Q)\,a\,\text{else}\,b]P \leftrightarrow (Q \rightarrow [a]P) \wedge (\neg Q \rightarrow [b]P)$

Create rather than use axioms for reasoning about CPS, which enables students to develop higher metacritical analytic skills

# Example Quiz

## Example (Quiz: Loop invariants)

Objective (*identifying and expressing invariants*): The most important ingredient of a CPS is its invariant, because an invariant tells you what you always know about your system, no matter how long it operates. This question allows you to practice the important but challenging task of identifying loop invariants for hybrid systems.

Identify a loop invariant $J$ proving the following dL formulas with exactly the following version of the loop invariant proof rule:

$$\text{loop}\ \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Write `n/a` when no loop invariant exists that proves the given dL formula.

$x \geq 1 \land v > 0 \land A > 0 \rightarrow [((a := 0 \cup a := A); \{x' = v, v' = a\})^*] x \geq 0$

**Answer:**

# Example Quiz

## Example (Quiz: Loop invariants)

Objective (*identifying and expressing invariants*): The most important ingredient of a CPS is its invariant, because an invariant tells you what you always know about your system, no matter how long it operates. This question allows you to practice the important but challenging task of identifying loop invariants for hybrid systems.
Identify a loop invariant $J$ proving the following dL formulas with exactly the following version of the loop invariant proof rule:

$$\text{loop}\ \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Write `n/a` when no loop invariant exists that proves the given dL formula.
$x \geq 1 \land v > 0 \land A > 0 \rightarrow [((a := 0 \cup a := A); \{x' = v, v' = a\})^*] x \geq 0$
**Answer:** $x \geq 0 \land v \geq 0 \land A \geq 0$

Figure out why a control loop works correctly

# Carnegie Mellon University
# May 5th, 2016

# Course Evaluation

- Essentially perfect course evaluations
- Students stay despite huge workload (≈20h/week with all parts)
- Resources: Lectures, videos, textbook, slides, quiz, KeYmaera X prover
- Active learning quiz introduction correlates with significant learning and deep understanding advances. But also at start of pandemic . . .
- Active learning quiz scores strongest predictor of course grade
- CPS V&V Grand Prix independent way to become verification rockstar
- Experimenting with different setups to reduce workload

# Outline

# Course: Programming Language Semantics

- Programs reach or influence important decisions
- Guarantees for program computations begin with program semantics
- Justify correctness of understanding / analysis / compilation of programs
- Characterize prog. languages syntactically, semantically, axiomatically
- Denotational semantics, operational semantics, axiomatic semantics
- Soundness & completeness
- Program specification and proof
- Interactive and game programming
- Concurrent programming
- Correct compilation and interpretation
- Use logic to simplify core concepts and avoid technicalities
- Key tricks: Dynamic logic and uniform substitution

# Logic Simplifies Program Program Semantics

## Partial Semantics from Programming Langue Implementations

- State $\omega : \mathscr{V} \dashrightarrow \mathscr{D}$ is **partial** function from variables in $\mathscr{V}$ to values in $\mathscr{D}$
- Meaning $\omega[\![x+1]\!]$ of term $x+1$ in state $\omega$ only defined if $\omega$ gives value to $x$. Then $\omega[\![x+1]\!] = \omega(x)+1$
- \+ Teaches delicacy + precision, e.g., for null-pointer or array out of bounds
- \- Every step conditional as syntax & semantics connection needs guards

## Impartial Semantics from Logic

- State $\omega : \mathscr{V} \to \mathscr{D}$ is **total** function from variables in $\mathscr{V}$ to values in $\mathscr{D}$
- Meaning $\omega[\![x+1]\!] = \omega(x)+1$ of term $x+1$ in state $\omega$ always defined
- \+ Everything always has a value
- \- Wastes information until coincidence lemma proves FV() dependency

## Lemma (Coincidence lemma)

*If $\omega = \nu$ on $FV(\theta)$, then $\omega[\![\theta]\!] = \nu[\![\theta]\!]$.*

# Logic Simplifies Syntactic Program Transformations

## Syntactic Transformation $\alpha(e) \rightsquigarrow \alpha(k)$ via Special Semantics

- define syntactic transformation mechanism $\alpha(e) \rightsquigarrow \alpha(k)$
- define program context $\alpha(\_)$ and semantics $[\![\cdot]\!]_{\text{cxt}}$ for every type of hole
- prove $[\![\alpha(e)]\!]_{\text{prg}} = [\![\alpha(\_)]\!]_{\text{cxt}}([\![e]\!]_{\text{part}})$ for every type of hole

\- technical and repetitive proofs (even if they teach precision)

## Syntactic Transformation $\alpha(e) \rightsquigarrow \alpha(k)$ via Logic

$$\text{CQ}\ \frac{f = g}{p(f) \leftrightarrow p(g)} \qquad \text{CT}\ \frac{f = g}{c(f) = c(g)} \qquad \text{CE}\ \frac{P \leftrightarrow Q}{C(P) \leftrightarrow C(Q)}$$

\+ Substituting equals for equals, no special mechanisms or proofs

## Theorem (Soundness of uniform substitution)

$$\textit{US}\ \frac{\varphi}{\sigma(\varphi)} \qquad \frac{\varphi_1 \quad \dots \quad \varphi_n}{\psi}\textit{loc.sound} \overset{\textit{US}}{\Rightarrow} \frac{\sigma(\varphi_1) \quad \dots \quad \sigma(\varphi_n)}{\sigma(\psi)}\textit{loc.sound}$$

# Logic Simplifies Compiler Optimizations

Logic reduces CSE/prop to simple axiom: $[:=]\ [x := e]p(x) \leftrightarrow p(e)$

CSE
$$[\text{while}(y^2 < a^2+b)\{z := z + y^2 * (a^2+b); y := y + 2*3\}]P \leftrightarrow [x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 2*3\}]P$$

CSE
$$[\text{while}(y^2 < a^2+b)\{z := z + y^2 * (a^2+b); y := y + 2*3\}]P \not\leftrightarrow [x := y^2; \text{while}(x < a^2+b)\{z := z + x * (a^2+b); y := y + 2*3\}]P$$

Prop
$$[x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 2*3\}]P \leftrightarrow [x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * (a^2+b); y := y + 2*3\}]P$$

Logic reduces fold to simple rule: CQ $\dfrac{f = g}{p(f) \leftrightarrow p(g)}$ CP $\dfrac{\alpha = \beta}{C(\alpha) \leftrightarrow C(\beta)}$

Fold
$$[x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 2*3\}]P \leftrightarrow [x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 6\}]P$$

Unfold
$$[x := a^2+b; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 6\}]P \leftrightarrow [x := b + a^2; \text{if}(y^2 < x)\{z := z + y^2 * x; y := y + 6; \text{while}(y^2 < x)\{z := z + y^2 * x; y := y + 6\}\}]P$$

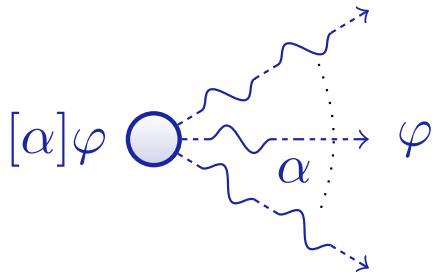
# Outline

# Significance of Symbolic Logic for Scientific Education

## Logical foundations make a big difference

**differential dynamic logic**

$$\mathsf{dL} = \mathsf{DL} + \mathsf{HP}$$




- Logic identifies core ideas
- Logic simplifies concepts
- Logic helps CS intro programming, logic courses, CPS, program semantics
- Logic helps (in)formally or with tools
- Dynamic logic
- Uniform substitution

- Active learning quizzes
- Theorem prover autogrades

1 Principles Imperative Comput.

3 Constructive Logic

3+ Compiler Design

3+ Logical Foundations of CPS

MS+ Progr. Lang. Semantics

`https://symbolaris.com`


André Platzer.
The significance of symbolic logic for scientific education.
In Emil Sekerinski and Leila Ribeiro, editors, *Formal Methods Teaching: 6th International Workshop, FMTea 2024, Milano, Italy, September 10, 2024, Proceedings*, volume 14939 of *LNCS*, pages 3–22. Springer, 2024.
doi:10.1007/978-3-031-71379-8_1.


André Platzer.
*Logical Foundations of Cyber-Physical Systems*.
Springer, Cham, 2018.
doi:10.1007/978-3-319-63588-0.


André Platzer.
*Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*.
Springer, Heidelberg, 2010.
doi:10.1007/978-3-642-14509-4.


André Platzer.
Logics of dynamical systems.

In LICS [11], pages 13–24.
doi:10.1109/LICS.2012.13.


André Platzer.
Differential dynamic logic for hybrid systems.
*J. Autom. Reas.*, 41(2):143–189, 2008.
doi:10.1007/s10817-008-9103-8.


André Platzer.
A complete uniform substitution calculus for differential dynamic logic.
*J. Autom. Reas.*, 59(2):219–265, 2017.
doi:10.1007/s10817-016-9385-1.


André Platzer.
Logic & proofs for cyber-physical systems.
In Nicola Olivetti and Ashish Tiwari, editors, *IJCAR*, volume 9706 of *LNCS*, pages 15–21, Cham, 2016. Springer.
doi:10.1007/978-3-319-40229-1_3.


André Platzer.
Differential game logic.
*ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015.

doi:10.1145/2817824.


André Platzer.
Differential hybrid games.
*ACM Trans. Comput. Log.*, 18(3):19:1–19:44, 2017.
doi:10.1145/3091123.


André Platzer.
The complete proof theory of hybrid systems.
In LICS [11], pages 541–550.
doi:10.1109/LICS.2012.64.


*Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, Los Alamitos, 2012. IEEE.