1. Learning Objectives

2. Gradual Introduction to Hybrid Programs

3. Hybrid Programs
   - Syntax
   - Semantics
   - Notational Convention

4. Examples

5. Summary
Outline

1. Learning Objectives

2. Gradual Introduction to Hybrid Programs

3. Hybrid Programs
   - Syntax
   - Semantics
   - Notational Convention

4. Examples

5. Summary
Learning Objectives

Choice & Control

- nondeterminism
- abstraction
- programming languages for CPS
- semantics
- compositionality

CT

M&C

CPS

models
core principles
discrete +
continuous

operational effect
operational precision

Andre Platzer (CMU)
1 Learning Objectives

2 Gradual Introduction to Hybrid Programs

3 Hybrid Programs
   - Syntax
   - Semantics
   - Notational Convention

4 Examples

5 Summary
Example (Speedy the point)

\[ \{ x' = v, v' = a \} \]

Purely continuous dynamics

What about the cyber?
Example (Speedy the point)

\[ a := a + 1 \]

Purely discrete dynamics

How do both meet?
Example (Speedy the point)

\[ a := a + 1; \quad \{ x' = v, v' = a \} \]

Hybrid dynamics, i.e., composition of continuous and discrete dynamics
Here: sequential composition first; second
Example (Speedy the point)

\[ a := -2; \quad \{x' = v, v' = a\}; \]
\[ a := 0.25; \quad \{x' = v, v' = a\}; \]
\[ a := -2; \quad \{x' = v, v' = a\}; \]
\[ a := -2; \quad \{x' = v, v' = a\}; \]
\[ a := 0.25; \quad \{x' = v, v' = a\}; \]
Playing with Acceleration and Braking

Example (Speedy the point)

\[ a := -2; \quad \{x' = v, v' = a\}; \]

\[ a := 0.25; \quad \{x' = v, v' = a\}; \]

\[ a := -2; \quad \{x' = v, v' = a\}; \]

\[ a := 0.25; \quad \{x' = v, v' = a\}; \]

How long to follow an ODE?
Playing with Acceleration and Braking

Example (Speedy the point)

\[ a := -2; \{x' = v, v' = a\}; \]
\[ a := 0.25; \{x' = v, v' = a\}; \]
\[ a := -2; \{x' = v, v' = a\}; \]
\[ a := 0.25; \{x' = v, v' = a\}; \]
\[ a := -2; \{x' = v, v' = a\}; \]
\[ a := 0.25; \{x' = v, v' = a\}; \]

How to check conditions before actions?
Example (Speedy the point)

\[
\text{if}(v < 4) a := a + 1 \text{ else } a := -b; \\
\{x' = v, v' = a\}
\]

Velocity-dependent control
Example (Speedy the point)

\[
\begin{align*}
\text{if}(x - m > s) & \quad a := a + 1 \quad \text{else} \quad a := -b; \\
\{x' = v, v' = a\}
\end{align*}
\]

Distance-dependent control for obstacle \( m \)
Example (Speedy the point)

\[
\text{if}(x - m > s \land v < 4) \ a := a + 1 \ \text{else} \ a := -b; \\
\{x' = v, v' = a\}
\]

Velocity and distance-dependent control

**Iterative Design**

Start as simple as possible, then add challenges once basics are correct.
### Example (Speedy the point)

\[
\text{if}(x - m > s \land v < 4 \land \text{efficiency}) \ a := a + 1 \ \text{else} \ a := -b;
\{x' = v, \ v' = a\}
\]

Also only accelerate if it's efficient to do so
Example (Speedy the point)

\[
\text{if}(x - m > s \land v < 4 \land \text{efficiency}) \ a := a + 1 \ \text{else} \ a := -b;
\{x' = v, v' = a\}
\]

Exact models are unnecessarily complex. Not all features are safety-critical.
Example (Speedy the point)

\[(a := a + 1 \cup a := -b); \{x' = v, v' = a\}\]

Nondeterministic choice \(\cup\) allows either side to be run, arbitrarily.

**Power of Abstraction**

Only include relevant aspects, elide irrelevant detail. The model and its analysis become simpler. And apply to more systems.
Example (Speedy the point)

\[ (a := a + 1 \cup a := -b); \]
\[ \{ x' = v, v' = a \} \]

Nondeterministic choice \( \cup \) allows either side to be run, arbitrarily.

Oops, now it got too simple! Not every choice is always acceptable.
Example (Speedy the point)

(?v < 4; a := a + 1 ∪ a := −b);
{x′ = v, v′ = a}

Test ?Q checks if formula Q is true in current state
Example (Speedy the point)

\[
(?v < 4; a := a + 1 \cup a := -b);
\{x' = v, v' = a\}
\]

Test \(Q\) checks if formula \(Q\) is true in current state, otherwise run fails.

**Discarding failed runs and backtracking**

System runs that fail tests are discarded and not considered further.

\(?v < 4; v := v + 1\) only runs if \(v < 4\) initially true
\(v := v + 1; ?v < 4\) only runs if \(v < 4\) initially true

**Broader significance of nondeterminism**

Nondeterminism is a tool for abstraction to focus on critical aspects. Nondeterminism is essential to describe imperfectly known environment.
Example (Speedy the point)

$(?v < 4; a := a + 1 \cup a := -b);$
$
\{x' = v, v' = a\}$

Test $?Q$ checks if formula $Q$ is true in current state, otherwise run fails.

**Discarding failed runs and backtracking**

System runs that fail tests are discarded and not considered further.

$?v < 4; v := v + 1$ only runs if $v < 4$ initially true

$v := v + 1; ?v < 4$ only runs if $v < 3$ initially true

**Broader significance of nondeterminism**

Nondeterminism is a tool for abstraction to focus on critical aspects. Nondeterminism is essential to describe imperfectly known environment.
Example (Speedy the point)

\[
\begin{align*}
(?v < 4; a &:= a + 1 \cup a := -b); \\
\{x' = v, v' = a\};
\end{align*}
\]

Repeated control needs longer programs, e.g., by copy&paste
Example (Speedy the point)

\[
\left( (?v < 4; a := a + 1 \cup a := -b); \\
\{x' = v, v' = a\}\right)^*
\]

Nondeterministic repetition * repeats any arbitrary number of times
Outline

1 Learning Objectives

2 Gradual Introduction to Hybrid Programs

3 Hybrid Programs
   • Syntax
   • Semantics
   • Notational Convention

4 Examples

5 Summary
Hybrid Programs: Syntax

Definition (Syntax of hybrid program $\alpha$)

$$\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha ; \beta \mid \alpha^*$$
Definition (Syntax of hybrid program $\alpha$)

$$
\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*
$$

- Discrete Assign
- Test Condition
- Differential Equation
- Nondet. Choice
- Seq. Compose
- Nondet. Repeat
Hybrid Programs: Syntax

Definition (Syntax of hybrid program $\alpha$)

$\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$

- Discrete Assign
- Test Condition
- Differential Equation
- Nondet. Choice
- Seq. Compose
- Nondet. Repeat

Like regular expressions. Everything nondeterministic.
Hybrid Programs: Semantics

\[
\begin{align*}
\omega & \xrightarrow{x := e} \nu \\
\omega' & = f(x) \land Q \\
\omega & \xrightarrow{?Q} \nu
\end{align*}
\]
Hybrid Programs: Semantics

\[ x := e \]

\[ x' = f(x) \& Q \]

\[ ?Q \]
Hybrid Programs: Semantics

\[
\begin{align*}
\omega & \xrightarrow{x := e} \nu \\
\omega & \xrightarrow{x' = f(x) \& Q} \nu \\
\omega & \xrightarrow{?Q} \nu
\end{align*}
\]

\[
\begin{align*}
\nu & \text{ if } \nu(x) = \omega[e] \\
\omega & \text{ and } \nu(z) = \omega(z) \text{ for } z \neq x
\end{align*}
\]

\[
\begin{align*}
x' &= f(x) \& Q \\
x &= \omega \quad \text{if } \omega| = Q \\
&\quad \text{and } \omega| / = Q \\
&\quad \text{otherwise no transition}
\end{align*}
\]
Hybrid Programs: Semantics

\[ x := e \]

\[ x' = f(x) \land Q \]

\[ ?Q \]
Hybrid Programs: Semantics

\[ x := e \]

\[ x' = f(x) \land Q \]

\[ \omega \quad \nu \]

\[ \omega \quad ?Q \quad \nu \]

[Diagram showing transition from \( \omega \) to \( \nu \) with conditions for each transition, including assignments and guards.]
Hybrid Programs: Semantics

\[ x := e \]

\[ x' = f(x) \land Q \]

\[ ?Q \]

\[ \text{if } \nu(x) = \omega[e] \text{ and } \nu(z) = \omega(z) \text{ for } z \neq x \]

\[ x' = f(x) \]

\[ \text{Q} \]
Hybrid Programs: Semantics

\[ x := e \]

\[ x' = f(x) \& Q \]

\[ ? Q \]

\[ \text{if } \omega \models Q \]

\[ \nu \text{ if } \nu(x) = \omega[e] \]

\[ \omega \text{ and } \nu(z) = \omega(z) \text{ for } z \neq x \]

\[ x' = f(x) \& Q \]

\[ \omega \text{ no change if } \omega \models Q \]
Hybrid Programs: Semantics

\[ x := e, \quad \nu(x) = \omega[e] \]

\[ x' = f(x) & Q, \quad \nu(x') = \omega(f(x)) & Q \]

\[ ?Q, \quad \text{if } \nu(x) = \omega[e] \]

\[ x' = f(x) & Q, \quad \text{otherwise no transition} \]

\[ \nu(x) = \omega(x) \text{ for } z \neq x \]

\[ \nu(x) = \omega(x) \text{ for } z \neq x \]

\[ \text{if } \omega \models Q \]

\[ \text{if } \omega \not\models Q \]

\[ \text{no change if } \omega \models Q \]

\[ \text{otherwise no transition} \]
Hybrid Programs: Semantics

$\omega$ \quad $\alpha \cup \beta$ \quad $\nu$

$\omega$ \quad $\alpha ; \beta$ \quad $\nu$

$\omega$ \quad $\alpha^*$ \quad $\nu$
Hybrid Programs: Semantics

André Platzer (CMU)
Hybrid Programs: Semantics

\[ \omega \xrightarrow{\alpha} \nu_1 \]
\[ \omega \xrightarrow{\alpha \cup \beta} \nu_2 \]
\[ \alpha ; \beta \]

\[ \omega \xrightarrow{\alpha} \mu \xrightarrow{\beta} \nu \]

\[ \omega \xrightarrow{\alpha} \nu_1 \]
\[ \nu_2 \]

\[ \omega \xrightarrow{\alpha} \omega_1 \xrightarrow{\alpha} \omega_2 \xrightarrow{\alpha} \nu \]

\[ X \]
\[ t \]

\[ \omega \xrightarrow{\mu} \nu \]

\[ X \]
\[ t \]

\[ X \]
\[ t \]

\[ \omega \xrightarrow{\nu} \nu \]

\[ X \]
\[ t \]
Plug-in for Semantics of Composed Hybrid Programs

\[ \alpha ; \beta \]

\[ \omega \rightarrow \mu \rightarrow \nu \]

\[ \alpha, \beta \]

\[ \alpha^* \]

\[ \omega \rightarrow \omega_1 \rightarrow \omega_2 \rightarrow \nu \]

\[ \alpha, \alpha, \alpha \]

\[ \omega \rightarrow \nu_1 \rightarrow \nu_2 \]

\[ \alpha, \alpha \cup \beta, \beta \]

\[ \omega \rightarrow \nu_1 \rightarrow \nu_2 \]
Plug-in for Semantics of Composed Hybrid Programs

\[ (\alpha; \beta)^* \]

\[ (\alpha \cup \beta)^* \]

\[ (\alpha; \beta; \gamma)^* \]

\[ (\alpha \cup \beta)^* \]

\[ (\alpha \cup \beta)^* \]

\[ (\alpha \cup \beta)^* \]

\[ (\alpha \cup \beta)^* \]
Hybrid Programs: Syntax & Semantics

Definition (Syntax of hybrid program $\alpha$)

$$\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha ; \beta \mid \alpha^*$$

Definition (Semantics of hybrid programs) $(\llbracket \cdot \rrbracket : \text{HP} \to \wp(S \times S))$

$$\llbracket x := e \rrbracket = \{ (\omega, \nu) : \nu = \omega \text{ except } \nu[x] = \omega[e] \}$$
$$\llbracket ?Q \rrbracket = \{ (\omega, \omega) : \omega \models Q \}$$
$$\llbracket x' = f(x) \rrbracket = \{ (\varphi(0), \varphi(r)) : \varphi \models x' = f(x) \text{ for some duration } r \geq 0 \}$$
$$\llbracket \alpha \cup \beta \rrbracket = \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket$$
$$\llbracket \alpha ; \beta \rrbracket = \llbracket \alpha \rrbracket \circ \llbracket \beta \rrbracket = \{ (\omega, \nu) : (\omega, \mu) \in \llbracket \alpha \rrbracket \text{ and } (\mu, \nu) \in \llbracket \beta \rrbracket \}$$
$$\llbracket \alpha^* \rrbracket = \llbracket \alpha \rrbracket^* = \bigcup_{n \in \mathbb{N}} \llbracket \alpha^n \rrbracket \quad \alpha^n \equiv \alpha; \alpha; \alpha; \ldots; \alpha \quad \text{compositional}$$

Andrè Platzer (CMU)
Hybrid Programs: Syntax & Semantics

Definition (Syntax of hybrid program $\alpha$)

\[\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*\]

Definition (Semantics of hybrid programs) \((\llbracket \cdot \rrbracket : \text{HP} \to \wp(S \times S))\)

\[
\begin{align*}
\llbracket x := e \rrbracket & = \{ (\omega, \nu) : \nu = \omega \text{ except } \nu[x] = \omega[e] \} \\
\llbracket ?Q \rrbracket & = \{ (\omega, \omega) : \omega \models Q \} \\
\llbracket x' = f(x) \rrbracket & = \{ (\varphi(0), \varphi(r)) : \varphi \models x' = f(x) \text{ for some duration } r \geq 0 \} \\
\llbracket \alpha \cup \beta \rrbracket & = \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \\
\llbracket \alpha; \beta \rrbracket & = \llbracket \alpha \rrbracket \circ \llbracket \beta \rrbracket \\
\llbracket \alpha^* \rrbracket & = \llbracket \alpha \rrbracket^* = \bigcup_{n \in \mathbb{N}} \llbracket \alpha^n \rrbracket 
\end{align*}
\]

1. $\varphi(z)(x') = \frac{d\varphi(t)(x)}{dt}(z)$ exists at all times $0 \leq z \leq r$
2. $\varphi(z) \models x' = f(x) \& Q$ for all times $0 \leq z \leq r$
3. $\varphi(z) = \varphi(0)$ except at $x, x'$

André Platzer (CMU)
**Example (Naming Conventions)**

<table>
<thead>
<tr>
<th>Letters</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z$</td>
<td>variables</td>
</tr>
<tr>
<td>$e, \tilde{e}$</td>
<td>terms</td>
</tr>
<tr>
<td>$P, Q$</td>
<td>formulas</td>
</tr>
<tr>
<td>$\alpha, \beta$</td>
<td>programs</td>
</tr>
<tr>
<td>$c$</td>
<td>constant symbols</td>
</tr>
<tr>
<td>$f, g, h$</td>
<td>function symbols</td>
</tr>
<tr>
<td>$p, q, r$</td>
<td>predicate symbols</td>
</tr>
</tbody>
</table>

In CPS applications, all bets are off because names follow application: $x$ position, $v$ velocity, and $a$ acceleration variables.
Notational Conventions: Precedence

Convention (Operator Precedence)

1. Unary operators (including \(*\), \(\neg\) and \(\forall x, \exists x\)) bind stronger than binary.
2. \(\land\) binds stronger than \(\lor\), which binds stronger than \(\rightarrow, \leftrightarrow\).
3. \(\;\) binds stronger than \(\cup\).
4. Arithmetic operators \(+, −, \cdot\) associate to the left.
5. Logical and program operators associate to the right.

Example (Operator Precedence)

\(\forall x P \land Q \equiv (\forall x P) \land Q\)
\(\forall x P \rightarrow Q \equiv (\forall x P) \rightarrow Q.\)
\(α; β \cup γ \equiv (α; β) \cup γ\)
\(α \cup β; γ \equiv α \cup (β; γ)\)
\(P \rightarrow Q \rightarrow R \equiv P \rightarrow (Q \rightarrow R).\)

But \(\rightarrow, \leftrightarrow\) expect explicit parentheses. Illegal: \(P \rightarrow Q \leftrightarrow R \quad P \leftrightarrow Q \rightarrow R\).
Outline

1. Learning Objectives

2. Gradual Introduction to Hybrid Programs

3. Hybrid Programs
   - Syntax
   - Semantics
   - Notational Convention

4. Examples

5. Summary
Robot ≡ (ctrl ; drive)^* 

ctrl ≡ (Q_A; a := A) 
∪ (Q_b; a := −b) 

drive ≡ t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon\}
Branching Transition Structure in Hybrid Programs

Robot ≡ (ctrl ; drive)∗

ctrl ≡ (?QA; a := A)
  ∪ (?Qb; a := −b)

drive ≡ t := 0; {x′ = v, v′ = a, t′ = 1 & v ≥ 0 ∧ t ≤ ε}
Branching Transition Structure in Hybrid Programs

Robot \equiv (\text{ctrl} ; \text{drive})^*

\text{ctrl} \equiv (?Q_A; a := A) \cup (?Q_b; a := -b)

\text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \land v \geq 0 \land t \leq \varepsilon\}
Robot ≡ (ctrl ; drive)*

ctrl ≡ (?QA; a := A)

U (?Qb; a := −b)

drive ≡ t := 0; \{ x' = v, v' = a, t' = 1 \& v \geq 0 \& t \leq \varepsilon \}
Branching Transition Structure in Hybrid Programs

Robot ≡ (ctrl ; drive)*
ctrl ≡ (?Q_A; a := A)
∪ (?Q_b; a := −b)
drive ≡ t := 0; \{x' = v, v' = a, t' = 1 & v ≥ 0 ∧ t ≤ ε\}
Branching Transition Structure in Hybrid Programs

Robot ≡ (ctrl ; drive)*
ctrl ≡ (?QA; a := A)
∪ (?Qb; a := \(-b\))
drive ≡ t := 0; \{x' = v, v' = a, t' = 1 & v ≥ 0 \land t ≤ \varepsilon\}
Robot $\equiv (\text{ctrl} ; \text{drive})^*$

\[
\begin{align*}
\text{ctrl} &\equiv (?Q_A; a := A) \\
&\quad \cup (?Q_b; a := -b) \\
\text{drive} &\equiv t := 0; \{x' = v, v' = a, t' = 1 \land v \geq 0 \land t \leq \varepsilon\}
\end{align*}
\]
Branching Transition Structure in Hybrid Programs

Robot \equiv (ctrl; drive)^*

ctrl \equiv (?Q_A; a := A)
\quad \cup (\; ?Q_b; a := -b \; )

drive \equiv t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon \}
Branching Transition Structure in Hybrid Programs

Robot ≡ (ctrl ; drive)*

ctrl ≡ (?Q_A; a := A)
∪ (?Q_b; a := −b)

drive ≡ t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon\}
Robot $\equiv (\text{ctrl} ; \text{drive})^*$

\[\text{ctrl} \equiv (?Q_A; a := A) \cup (?Q_b; a := -b)\]

\[\text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \& t \leq \varepsilon\}\]
Branching Transition Structure in Hybrid Programs

Robot \equiv (\text{ctrl} ; \text{drive})^*
\[
\begin{align*}
\text{ctrl} & \equiv (?Q_A; a := A) \\
& \quad \cup (?Q_b; a := -b)
\end{align*}
\]
\[
\begin{align*}
\text{drive} & \equiv t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon\}
\end{align*}
\]
Branching Transition Structure in Hybrid Programs

Robot \equiv (\text{ctrl} ; \text{drive})^*

\text{ctrl} \equiv (?Q_A; a := A)
\quad \cup (?Q_b; a := -b)

\text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \land v \geq 0 \land t \leq \varepsilon\}
Robot \equiv (\text{ctrl} ; \text{drive})^*

\text{ctrl} \equiv (\mathcal{Q}_A; a := A)
\quad \cup (\mathcal{Q}_b; a := -b)

\text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \land v \geq 0 \land t \leq \varepsilon\}
Branching Transition Structure in Hybrid Programs

\[
\begin{align*}
?Q_A & \quad a := A \\
\cup & \\
?Q_b & \quad a := -b
\end{align*}
\]

Robot \equiv (ctrl ; drive)^* \quad \\
ctrl \equiv (?Q_A; a := A) \cup (?Q_b; a := -b) \quad \\
drive \equiv t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon\}
Branching Transition Structure in Hybrid Programs

Robot ≡ (ctrl ; drive)*

ctrl ≡ (?Q_A; a := A)
   ∪ (?Q_b; a := −b)

drive ≡ t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \& t \leq \epsilon\}
Branching Transition Structure in Hybrid Programs

Robot \equiv (\text{ctrl} ; \text{drive})^*

\text{ctrl} \equiv (\Diamond Q_A; a := A)
\quad \cup (\Diamond Q_b; a := -b)

\text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \land v \geq 0 \land t \leq \varepsilon\}

if(\Diamond) \alpha \text{ else } \beta \equiv

while(\Diamond) \alpha \equiv
Branching Transition Structure in Hybrid Programs

\[
\text{if}(Q) \alpha \text{ else } \beta \equiv (?Q; \alpha) \cup (?\neg Q; \beta)
\]

\[
\text{while}(Q) \alpha \equiv
\]

Robot \equiv (ctrl ; drive)^*

ctrl \equiv (?Q_A; a := A)

∪ (?Q_b; a := -b)

drive \equiv t := 0; \{x' = v, v' = a, t' = 1 & v \geq 0 \land t \leq \varepsilon\}
Branching Transition Structure in Hybrid Programs

if\( (Q) \alpha \) else \( \beta \equiv (?Q; \alpha) \cup (?\neg Q; \beta) \)

while\( (Q) \alpha \equiv (?Q; \alpha)^*; ?\neg Q \)

Robot \( \equiv (\text{ctrl} ; \text{drive})^* \)

\( \text{ctrl} \equiv (?Q_A; a := A) \)

\( \cup (?Q_b; a := -b) \)

\( \text{drive} \equiv t := 0; \{x' = v, v' = a, t' = 1 \& v \geq 0 \land t \leq \varepsilon\} \)
Runaround Robot with Dubins Paths

\[
Q \equiv (x' + w\omega - o) + (y' - v\omega - o) \neq v^2 + w^2
\]

\[
Q_0 \equiv (o - x)w \neq (o - y)v
\]

Obstacle not on tangential circle
Obstacle not on ray \((x, y) + R(v, w)\)

Example (Runaround Robot)

\[
(x', y', v', w') = \begin{cases} 
\omega w, & \text{if } \omega = -1 \cup \omega = 1 \cup \omega = 0; \\
\omega v, & \text{otherwise.}
\end{cases}
\]
Example (Runaround Robot)

\[
((\omega := -1 \cup \omega := 1 \cup \omega := 0); \\
\{x' = v, y' = w, v' = \omega w, w' = -\omega v}\)^* 
\]
Example (Runaround Robot)

\[
((\textcolor{red}{?Q}_{-1}; \omega := -1 \cup ?Q_1; \omega := 1 \cup ?Q_0; \omega := 0); \\
\{x' = v, y' = w, v' = \omega w, w' = -\omega v\})^*
\]
Example (Speedy the point)

\[
(\forall v < 4; a := a + 1 \cup a := -b); \\
\{x' = v, v' = a\}; \\
(\forall v < 4; a := a + 1 \cup a := -b); \\
\{x' = v, v' = a\}; \\
(\forall v < 4; a := a + 1 \cup a := -b); \\
\{x' = v, v' = a\}
\]
Example (Speedy the point)

\[ \begin{align*}
?v &< 4; \ a := a + 1; \\
\{x' = v, \ v' = a\}; \\
?v &< 4; \ a := a + 1; \\
\{x' = v, \ v' = a\}; \\
?v &< 4; \ a := a + 1; \\
\{x' = v, \ v' = a\}
\end{align*} \]
A Matter of Choice: Recall Speedy, Remove Stuff

Example (Speedy the point)

\[
\begin{align*}
? & v < 4; a := a + 1; \\
\{ & x' = v, v' = a\}; \\
? & v < 4; a := a + 1; \\
\{ & x' = v, v' = a\}; \\
? & v < 4; a := a + 1; \\
\{ & x' = v, v' = a\}
\end{align*}
\]

No wait, now it’s a bad model! The HP assumes the test \( v < 4 \) passes after each ODE. No other choices are available.

Don’t let your controller discard important cases!
Outline

1. Learning Objectives

2. Gradual Introduction to Hybrid Programs

3. Hybrid Programs
   - Syntax
   - Semantics
   - Notational Convention

4. Examples

5. Summary
Hybrid Programs: Syntax & Semantics

Definition (Syntax of hybrid program $\alpha$)

$$\alpha, \beta ::= x := e \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

Definition (Semantics of hybrid programs) \([\cdot] : \text{HP} \rightarrow \wp(S \times S)\)

\[
\begin{align*}
[x := e] &= \{(\omega, \nu) : \nu = \omega \text{ except } \nu[x] = \omega[e]\} \\
[?Q] &= \{(\omega, \omega) : \omega \models Q\} \\
[x' = f(x)] &= \{(\varphi(0), \varphi(r)) : \varphi \models x' = f(x) \text{ for some duration } r \geq 0\} \\
[\alpha \cup \beta] &= [\alpha] \cup [\beta] \\
[\alpha; \beta] &= [\alpha] \circ [\beta] \\
[\alpha^*] &= [\alpha]^* = \bigcup_{n \in \mathbb{N}} [\alpha^n]
\end{align*}
\]

Andr´e Platzer (CMU)
André Platzer.
*Logical Foundations of Cyber-Physical Systems.*
URL: http://www.springer.com/978-3-319-63587-3,
doi:10.1007/978-3-319-63588-0.

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

André Platzer.
Logics of dynamical systems.

André Platzer.
Differential dynamic logic for hybrid systems.
André Platzer.
A complete uniform substitution calculus for differential dynamic logic.