# Constructing and Verifying Cyber Physical Systems

## Differential Dynamic Logic and KeYmaera X

Marcus Völp

# Overview

| | |
|---|---|
| Introduction<br>Mathematical Foundations (Differential Equations and Laplace Transformation) | Math |
| Control and Feedback | Physics |
| Transfer Functions and State Space Models<br>Poles, Zeros / PID Control<br>Stability, Root Locust Method, Digital Control | Feedback Control |
| Mixed-Criticality Scheduling and Real-Time Operating Systems (RTOS) | RTOS |
| Coordinating Networked Cyber-Physical Systems | CPS |
| Program Verification<br>**Differential Dynamic Logic and KeYmaera X**<br>Differential Invariants | Verification |

# Overview

TECHNISCHE UNIVERSITÄT DRESDEN

**Why Hybrid Systems Verification**

↓

**Differential Dynamic Logic**

→ **Hybrid Programs**

↓

**Transition Semantics of Hybrid Programs**

↓

**Proof Rules**

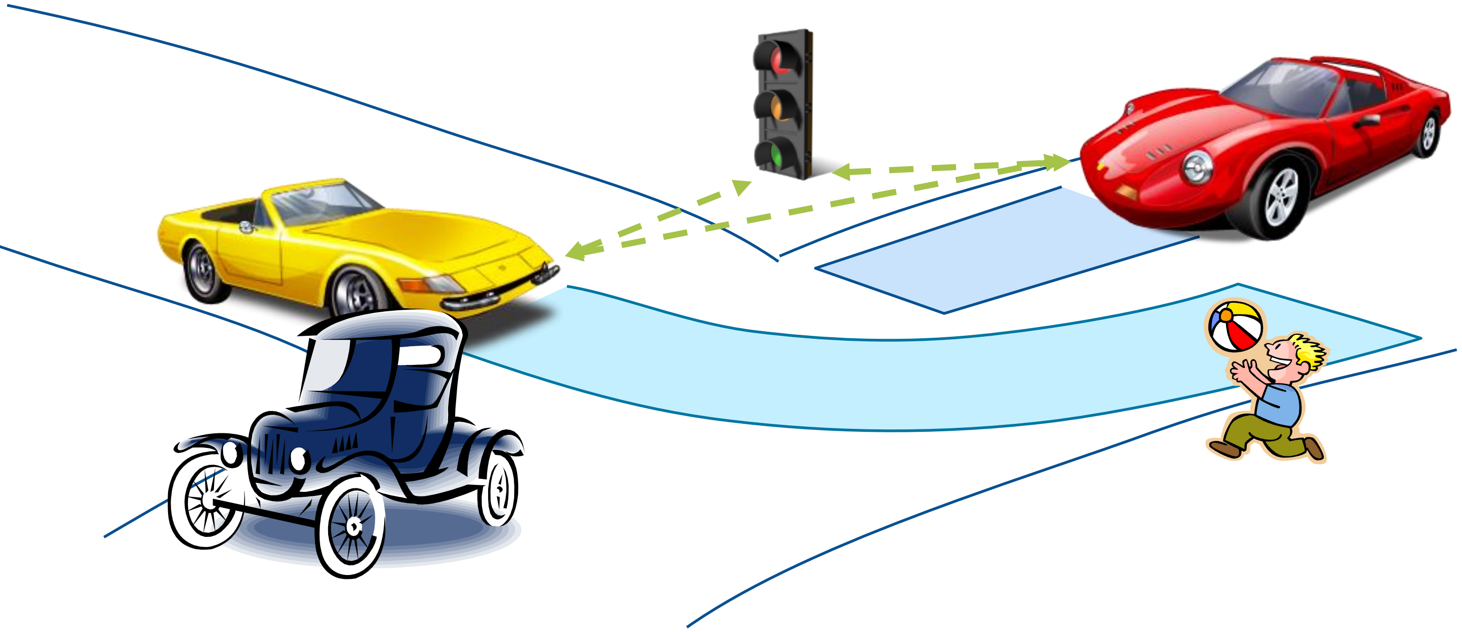
→ **KeYmaeraX**

symbolaris.com

Andre Platzer:
Logical Analysis of
Hybrid-Systems

# Why Hybrid Systems Verification

Source: Marcus Grundmann



**security** and **dependability** are inevitable

**late results** and **erroneous behavior** immediately affect reality

**formal verification**

# Why Hybrid Systems Verification

```
float PID::get_pid(float error, float scaler)
{
    uint32_t tnow = hal.scheduler->millis();
    uint32_t dt = tnow - _last_t;
    float output          = 0;
    float delta_time;

    if (_last_t == 0 || dt > 1000) { // reset integrator if inactive for a second
        dt = 0;
        reset_I();
    }
    _last_t = tnow;
    delta_time = (float)dt / 1000.0f;

    // Compute proportional component
    output += error * _kp;

    // Compute derivative component if time has elapsed
    if ((fabsf(_kd) > 0) && (dt > 0)) {
        float derivative;

            if (isnan(_last_derivative)) {
                        derivative = 0;
                        _last_derivative = 0;
            } else {
                        derivative = (error - _last_error) / delta_time;
            }

      // discrete low pass filter, cuts out the
        // high frequency noise that can drive the controller crazy
        float RC = 1/(2*PI*_fCut);
        derivative = _last_derivative +
                    ((delta_time / (RC + delta_time)) *
                     (derivative - _last_derivative));

        // update state
        _last_error          = error;
        _last_derivative    = derivative;

        // add in derivative component
        output += _kd * derivative;
    }

    // scale the P and D components
    output *= scaler;

    // Compute integral component if time has elapsed
    if ((fabsf(_ki) > 0) && (dt > 0)) {
        _integrator          += (error * _ki) * scaler * delta_time;
        if (_integrator < -_imax) {
            _integrator = -_imax;
        } else if (_integrator > _imax) {
            _integrator = _imax;
        }
        output += _integrator;
    }
    return output;
}
```

# Why Hybrid Systems Verification

```
float PID::get_pid(float error, float scaler)
{
   uint32_t tnow = hal.scheduler->millis();
   uint32_t dt = tnow - _last_t;
   float output         = 0;
   float delta_time;

   if (_last_t == 0 || dt > 1000) { // reset integrator if inactive for a second
      dt = 0;
      reset_I();
   }
   _last_t = tnow;
   delta_time = (float)dt / 1000.0f;

   // Compute proportional component
   output += error * _kp;

   // Compute derivative component if time
   if ((fabsf(_kd) > 0) && (dt > 0)) {
      float derivative;

         if (isnan(_last_derivative)) {
                  derivative = 0;
                  _last_derivative = 0;
         } else {
                  derivative = (error - _last_error) / delta_time;
         }
```

```
   // discrete low pass filter, cuts out the
   // high frequency noise that can drive the controller crazy
   float RC = 1/(2*PI*_fCut);
   derivative = _last_derivative +
            ((delta_time / (RC + delta_time)) *
             (derivative - _last_derivative));

   // update state
   _last_error        = error;
```

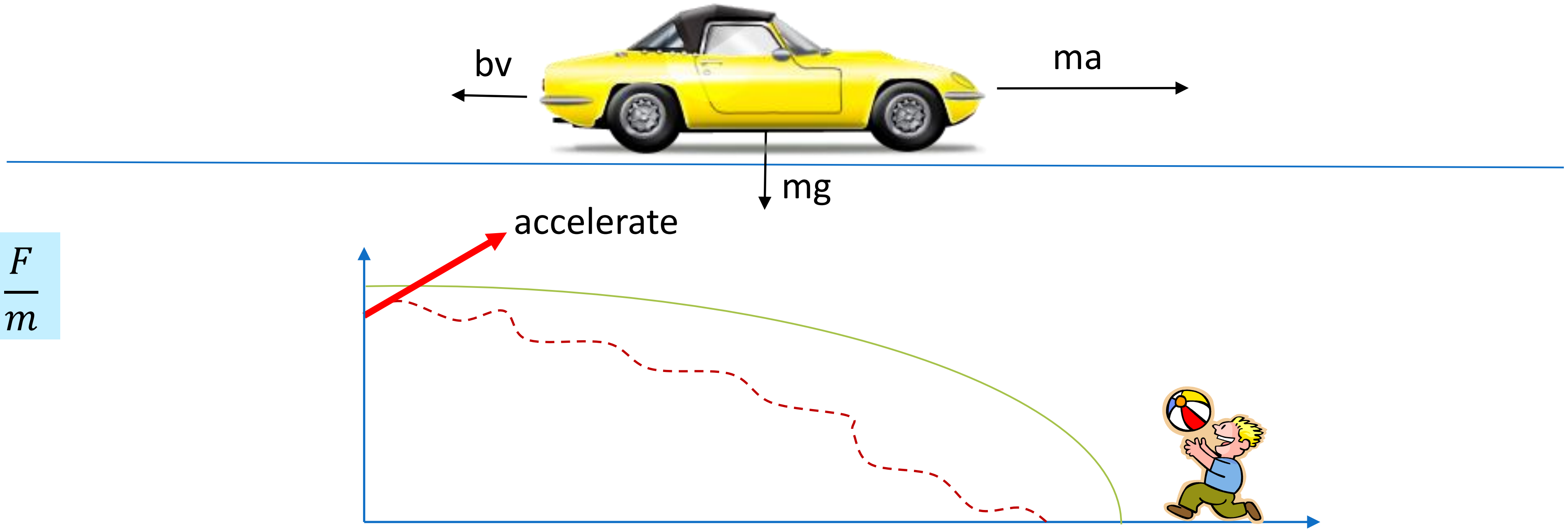
```
   if ((fabsf(_ki) > 0) && (dt > 0)) {
      _integrator          += (error * _ki) * scaler * delta_time;
      if (_integrator < -_imax) {
         _integrator = -_imax;
      } else if (_integrator > _imax) {
         _integrator = _imax;
      }
      output += _integrator;
   }
   return output;
}
```

Program verification only reveals errors in the code.

Did we use the right dynamics?
Does the controller match the dynamics (linearization, …)?

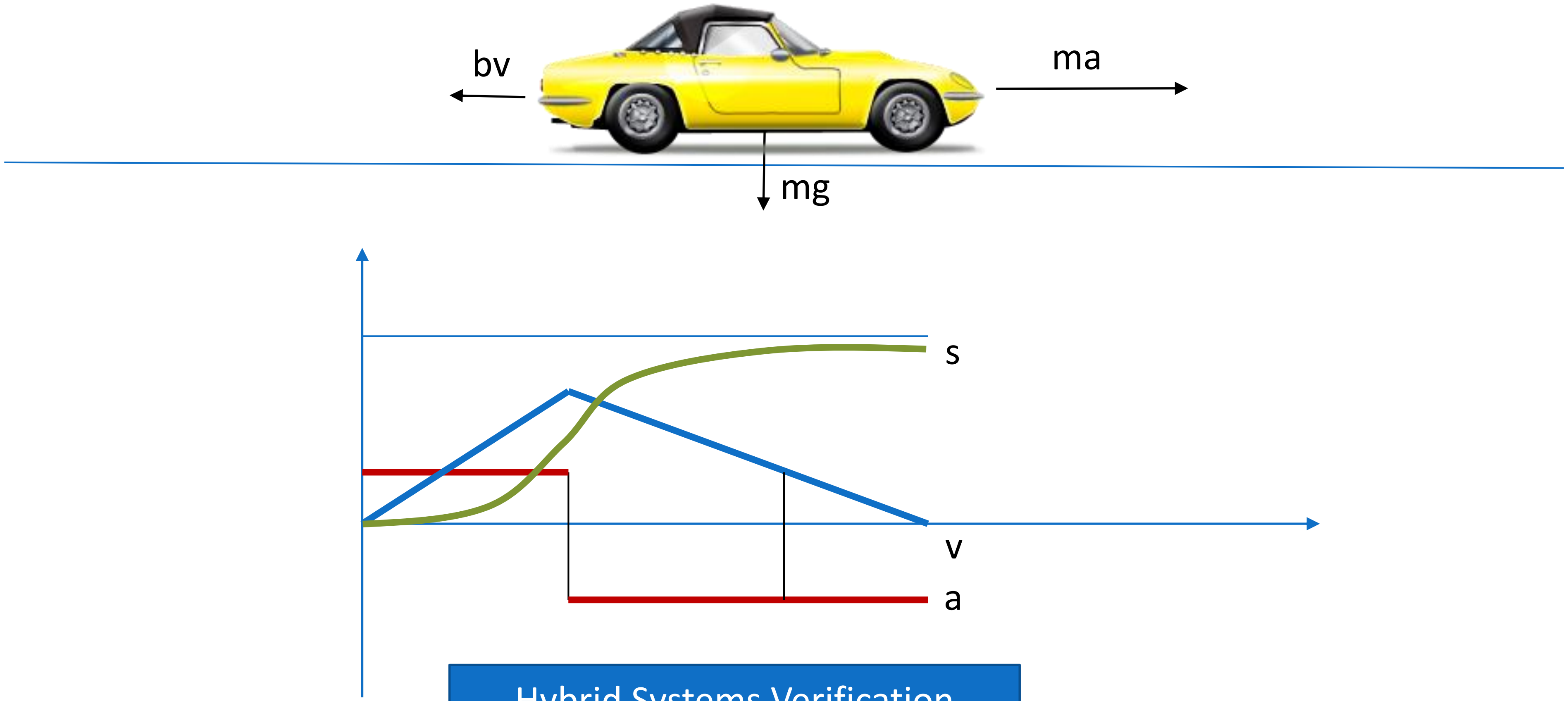
# Why Hybrid Systems Verification

$$v' + \frac{b}{m} v = \frac{F}{m}$$




Only looking at the continuous side neglects errors due to digital control decisions!

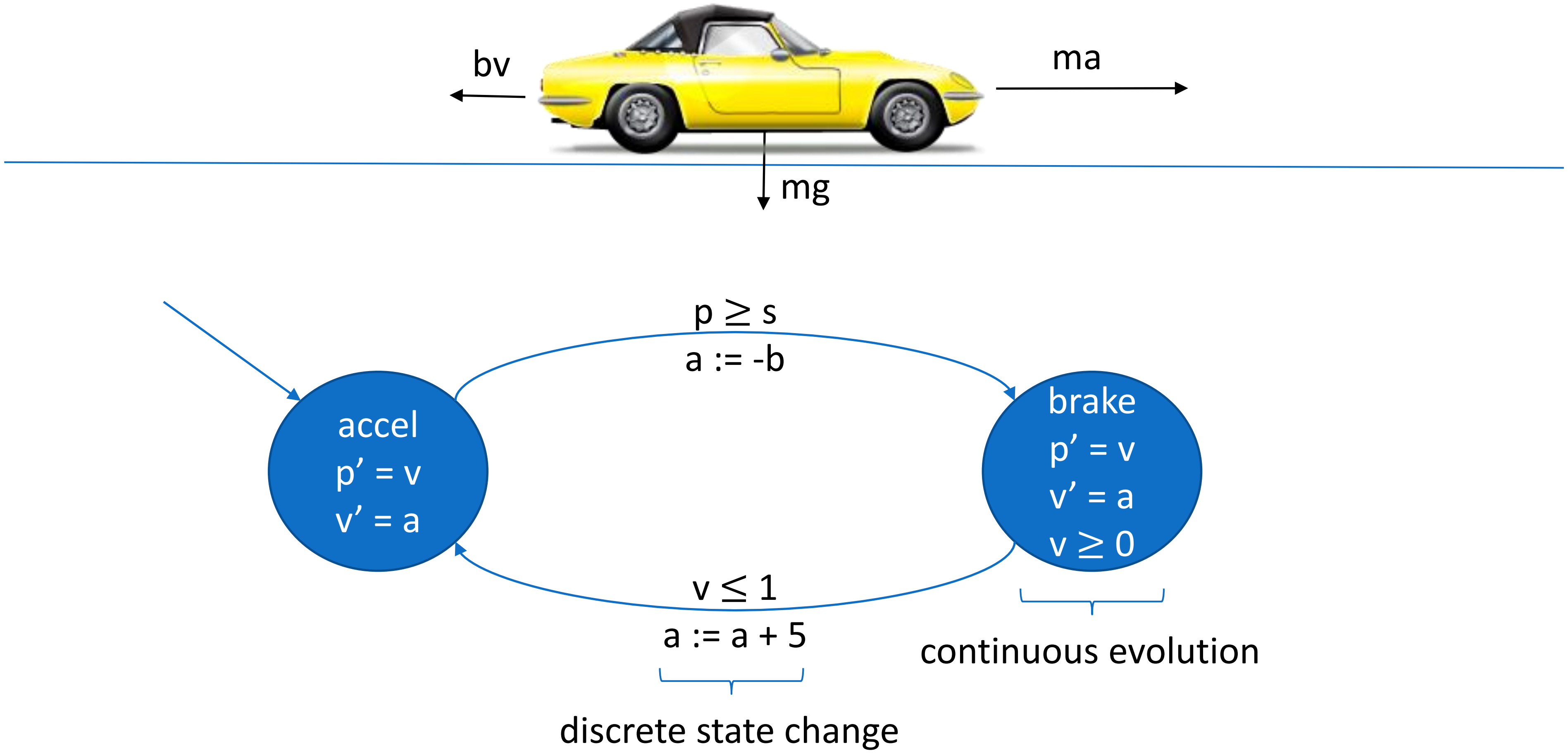
# Why Hybrid Systems Verification



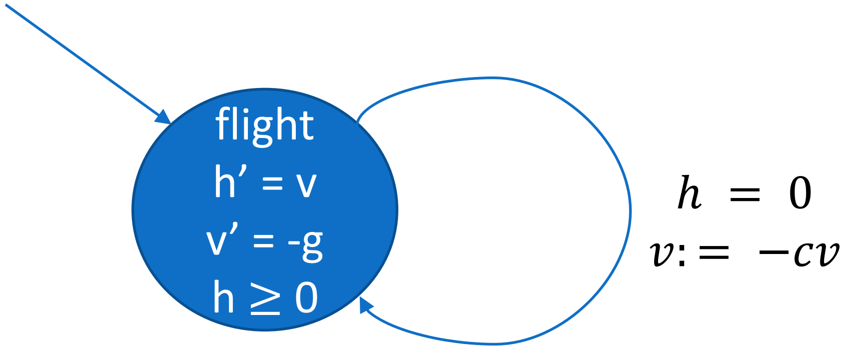

Hybrid Systems Verification

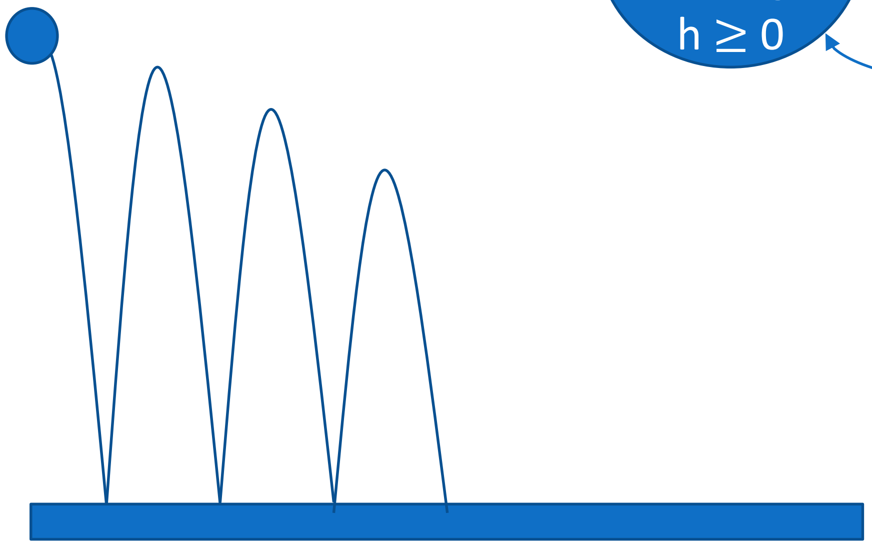
**Hybrid system:** dynamical systems where the system state evolves over time according to interacting laws of discrete and continuous dynamics.

# Hybrid Automaton



# Hybrid Automaton

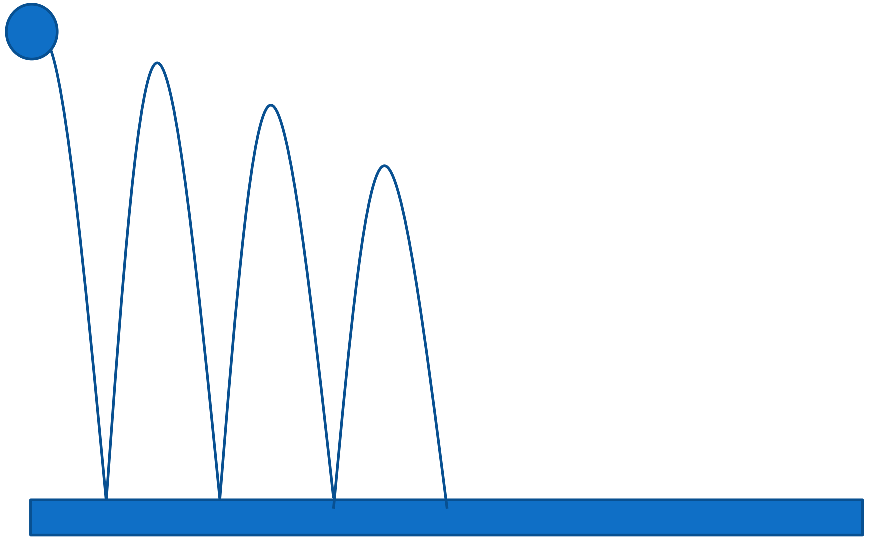


$$(\{h' = v, v' = -g \;\&\; h \geq 0\};$$
$$if\ (h = 0)\ then$$
$$v := -cv$$
$$fi)^*$$

hybrid program

Syntax



$(\{h' = v, v' = -g \;\&\; h \geq 0\};$ — differential equation system
$if\ (h = 0)\ then$ — control structure
$v := -cv$ — discrete jump
$fi)^*$

hybrid program

$$\alpha \ ::= \ \alpha; \beta \mid \alpha \cup \beta \mid \alpha^* \mid x := 0 \mid x := * \mid \{x'_1 = \theta_1, \dots, x'_n = \theta_n \;\&\; F\} \mid ?F$$

$$\Phi \ ::= \ \theta_1 \sim \theta_2 | \neg\Phi | \Phi \wedge \psi | \Phi \vee \psi | \Phi \rightarrow \psi | \Phi \leftrightarrow \psi | \forall x. \Phi | \exists x. \Phi | [\alpha]\Phi | \langle\alpha\rangle\Phi$$

# Differential Dynamic Logic

TECHNISCHE UNIVERSITÄT DRESDEN

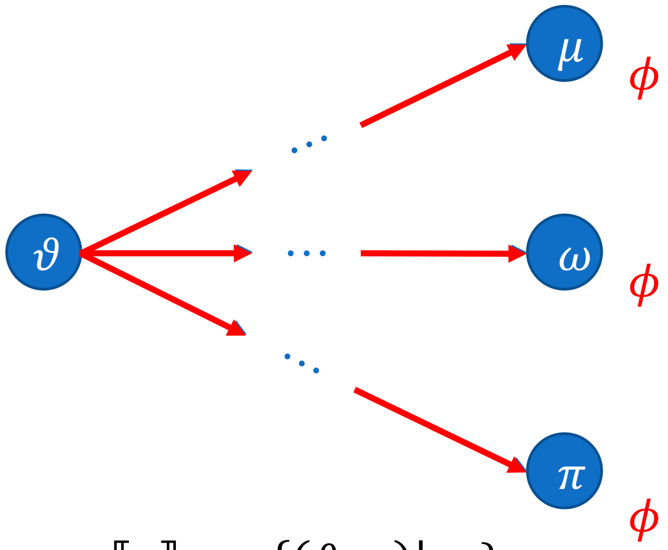
## Syntax

| | |
|---|---|
| $\alpha; \beta$ | sequential composition ($\alpha$ before $\beta$) |
| $\alpha \cup \beta$ | nondeterministic choice ($\alpha$ or $\beta$) |
| $\alpha^*$ | nondeterministic repetition ($\alpha$ some number of times, incl. 0) |
| $x \coloneqq 0$ | assignment |
| $x \coloneqq *$ | random assignment (with arbitrary value) |
| $\{x'_1 = \theta_1, \dots, x'_n = \theta_n \,\&\, F\}$ | continuous evolution, $F$ must hold the entire time |
| $?\,F$ | deadlock if $F$ is false |
| $[\alpha]\Phi$ | modality box: true if $\Phi$ holds after all runs of $\alpha$ |
| $\langle\alpha\rangle\Phi$ | modality diamond: true if $\Phi$ holds after at least one run of $\alpha$ |

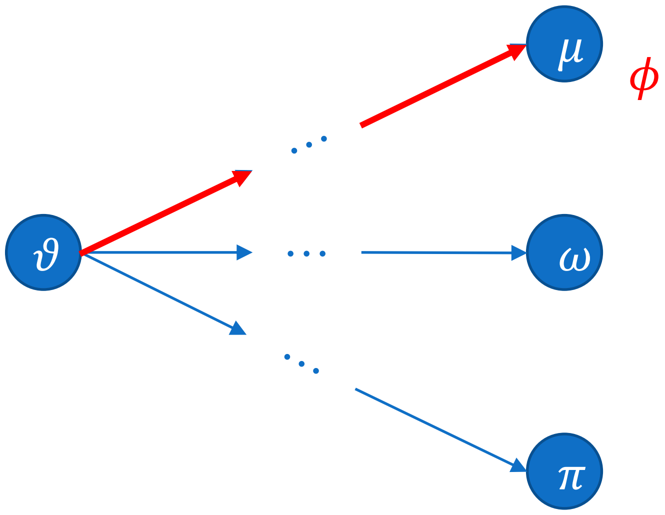
## Semantics

$$\alpha \ ::= \ \alpha;\beta \mid \alpha \ \cup \ \beta \mid \alpha^* \mid x := 0 \mid x := * \mid \{x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ F\} \mid ?\,F$$

$$\Phi \ ::= \ \theta_1 \sim \theta_2 | \neg\Phi | \Phi \wedge \psi | \Phi \vee \psi | \Phi \rightarrow \psi | \Phi \leftrightarrow \psi | \forall x.\, \Phi | \exists x.\, \Phi | [\alpha]\Phi | \langle\alpha\rangle\Phi$$




$[\alpha]\Phi$




$\langle\alpha\rangle\Phi$

$[\![\alpha]\!] ::= \{(\vartheta, \omega) | \dots\}$

TECHNISCHE UNIVERSITÄT DRESDEN

## Semantics

$$\alpha \;::=\; \alpha;\beta \mid \alpha \cup \beta \mid \alpha^* \mid x \coloneqq 0 \mid x \coloneqq * \mid \{x'_1 = \theta_1, \dots, x'_n = \theta_n \;\&\; F\} \mid ?\,F$$

$$\Phi \;::=\; \theta_1 \sim \theta_2 | \neg\Phi | \Phi \wedge \psi | \Phi \vee \psi | \Phi \rightarrow \psi | \Phi \leftrightarrow \psi | \forall x.\, \Phi | \exists x.\, \Phi | [\alpha]\Phi | \langle\alpha\rangle\Phi$$


$Var \rightarrow \mathbb{R}$

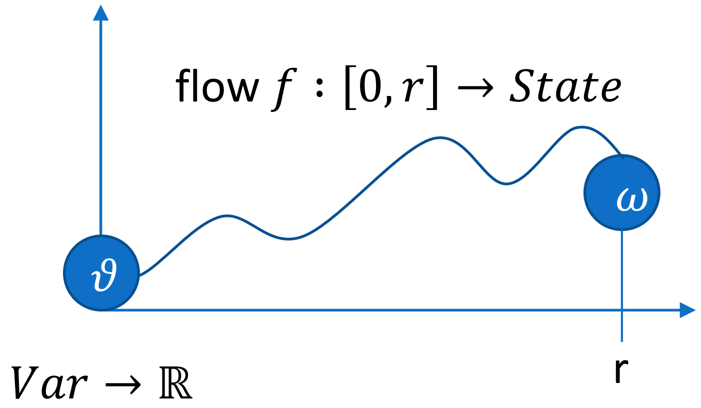
$$[\![x \coloneqq 0]\!] ::= \{(\vartheta, \omega) \mid \omega = \vartheta[x \longmapsto 0]\}$$
$$[\![x \coloneqq *]\!] ::= \{(\vartheta, \omega) \mid \omega = \vartheta[x \longmapsto r], r \in \mathbb{R}\}$$
$$[\![?\,F]\!] ::= \{(\vartheta, \vartheta) \mid F(\vartheta) = true\}$$
$$[\![\alpha;\beta]\!] ::= \{(\vartheta, \omega) \mid (\vartheta, \mu) \in [\![\alpha]\!], (\mu, \omega) \in [\![\beta]\!]\}$$
$$[\![\alpha \cup \beta]\!] ::= [\![\alpha]\!] \cup [\![\beta]\!]$$
$$[\![\alpha^*]\!] ::= \{(\vartheta, \omega) \mid \exists\, n \in \mathbb{N}.\, \vartheta = \vartheta_0, \vartheta_n = \omega, (\vartheta_i, \vartheta_{i+1}) \in [\![\alpha]\!], 0 \leq i < n\}$$

# Differential Dynamic Logic

## Semantics

$$\alpha \ ::= \ \alpha;\beta \mid \alpha \ \cup \ \beta \mid \alpha^* \mid x \coloneqq 0 \mid x \coloneqq * \mid \underline{\{x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ F\}} \mid ?\,F$$

$$\Phi \ ::= \ \theta_1 \sim \theta_2 | \neg\Phi | \Phi \wedge \psi | \Phi \vee \psi | \Phi \rightarrow \psi | \Phi \leftrightarrow \psi | \forall x.\, \Phi | \exists x.\, \Phi | [\alpha]\Phi | \langle\alpha\rangle\Phi$$

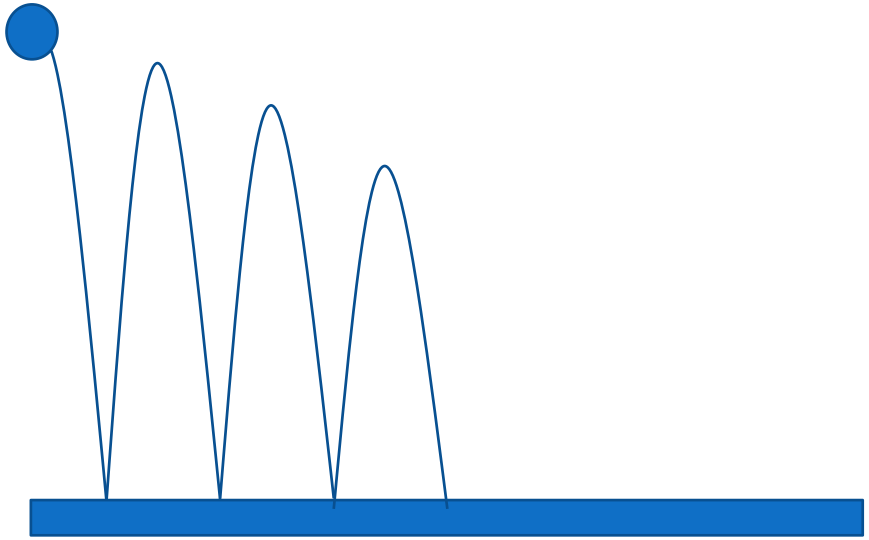



$[\![\{x'_1 = \theta_1, \dots, x'_n = \theta_n \ \& \ F\}]\!] ::= \{(\vartheta, \omega) | \dots$

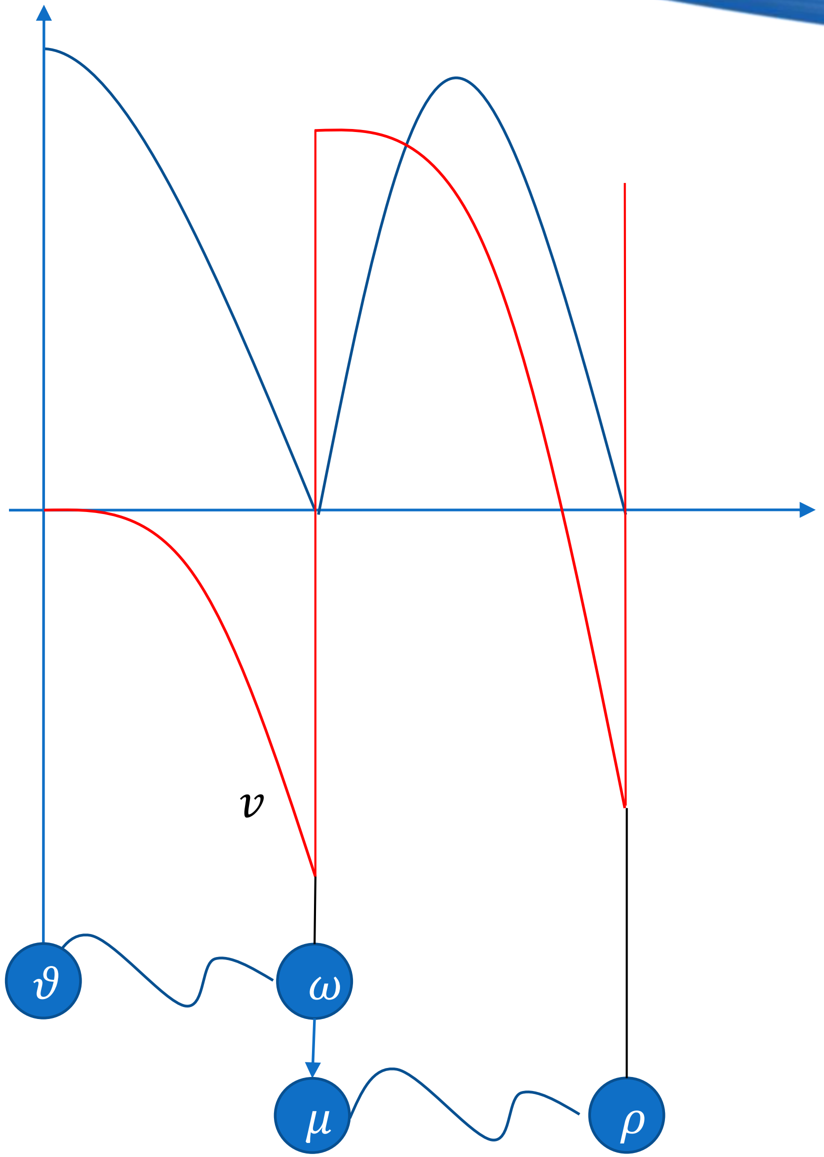
- $f(0) = \vartheta, f(r) = \omega$
- $f$ respects the differential equation:
  - $val_{I,\eta}(f(\zeta), x_i) = f(\zeta)(x_i)$ is continuous in $\zeta \in [0, r]$
  - $f$ is differentiable and has value $val_{I,\eta}(f'(\zeta), \theta x_i)$ in $\zeta \in (0, r)$
- $f$ respects the invariant $val_{I,\eta}(f(\zeta), F) = true$ for $\zeta \in [0, r]$
- (assume $y' = 0$ for all other variables)

# Differential Dynamic Logic

## Semantics



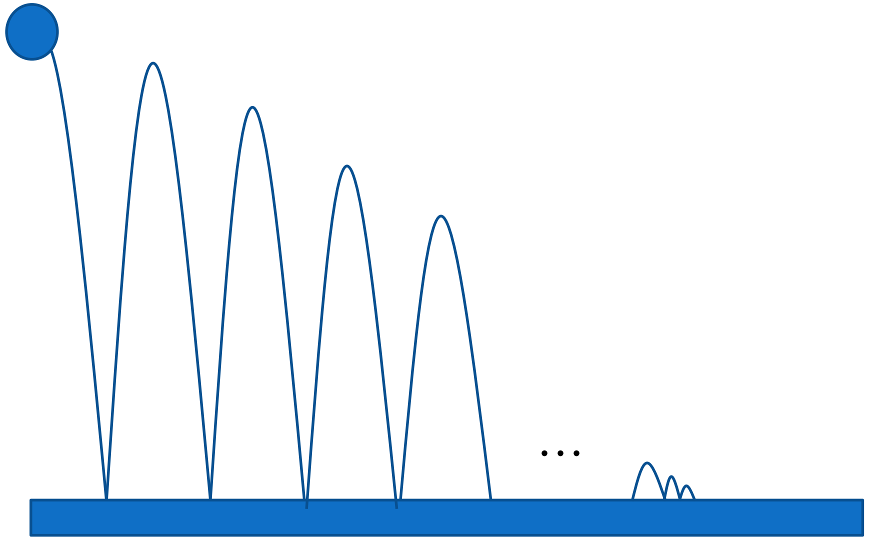
$(\{h' = v, v' = -g \ \& \ h \geq 0\};$
$\ if \ (h = 0) \ then$
$\qquad v := -cv$
$\ fi)^*$



# Differential Dynamic Logic

**Super Dense Time and Zeno Behavior:**

System behavior is zeno if infinitely many discrete transitions happen in finite time.



Hybrid programs do not define discrete actions in parallel to continuous evolutions. There is no differential equation to describe how reality evolves!

But! It is possible to emulate all desired behavior.

$$\underbrace{\{x'_i = \theta_i, t' = 1 \,\&\, F \wedge t \leq t_{max}\}; ?\,t \geq t_{min};}_{\text{wait between } t_{min} \text{ and } t_{max}} \underbrace{y := 42}_{\text{before update becomes effective}}$$

# Differential Dynamic Logic

## Sequent Calculus

$\Gamma, \phi \vdash \psi, \Delta$

$\phi_1,$
$\ldots,$
$\phi_n$
$\vdash$
$\psi_1,$
$\ldots,$
$\psi_n$

$$\frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta}$$

$$\frac{\Gamma \vdash \phi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}$$

$$\frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta}$$

# Differential Dynamic Logic

## Sequent Calculus

$$
\frac{\dfrac{\begin{array}{c} H \geq 0, \\ c \geq 0 \\ c < 1 \\ g > 0 \\ \frac{1}{2} m v^2 \leq m g (H - h) \\ \vdash \\ [(\{h' = v, v' = -g \,\&\, h \geq 0\}; h \geq 0 \cup (?\,h = 0; v := -cv))^*](h \geq 0 \wedge h \leq H) \end{array}}{\begin{array}{c} H \geq 0 \wedge c \geq 0 \wedge c < 1 \wedge g > 0 \wedge \frac{1}{2} m v^2 \leq m g (H - h) \\ \vdash \\ [(\{h' = v, v' = -g \,\&\, h \geq 0\}; h \geq 0 \cup (?\,h = 0; v := -cv))^*](h \geq 0 \wedge h \leq H) \end{array}}}{\begin{array}{c} H \geq 0 \wedge c \geq 0 \wedge c < 1 \wedge g > 0 \wedge \frac{1}{2} m v^2 \leq m g (H - h) \rightarrow \\ [(\{h' = v, v' = -g \,\&\, h \geq 0\}; h \geq 0 \cup (?\,h = 0; v := -cv))^*](h \geq 0 \wedge h \leq H) \end{array}}
$$

$$
\frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta}
$$

$$
\frac{\Gamma \vdash \phi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}
$$

$$
\frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta}
$$

# Differential Dynamic Logic

TECHNISCHE UNIVERSITÄT DRESDEN

KeYmaera Cheat Sheet http://symbolaris.com/info/KeYmaera.html 2

$$(\neg r \text{ not right}) \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \neg\phi, \Delta} \quad (\vee r \text{ or right}) \frac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \vee \psi, \Delta} \quad (\wedge r \text{ and right}) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \quad (\rightarrow r \text{ imply right}) \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta}$$

$$(\neg l \text{ not left}) \frac{\Gamma \vdash \phi, \Delta}{\Gamma, \neg\phi \vdash \Delta} \quad (\vee l \text{ or left}) \frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \vee \psi \vdash \Delta} \quad (\wedge l \text{ and left}) \frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta} \quad (\rightarrow l \text{ imply left}) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \rightarrow \psi \vdash \Delta}$$

$$(ax \text{ close}) \frac{}{\Gamma, \phi \vdash \phi, \Delta} \quad (cut) \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta}$$

$$(\langle;\rangle \text{ compose}) \frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \quad (\langle{*^n}\rangle \text{ unwind}) \frac{\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}{\langle\alpha^*\rangle\phi} \quad (\langle:=\rangle \text{ assign}) \frac{\phi_{x_1}^{\theta_1} \cdots {}_{x_n}^{\theta_n}}{\langle x_1 := \theta_1, \ldots, x_n := \theta_n\rangle\phi}$$

$$([;] \text{ compose}) \frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi} \quad ([{*^n}] \text{ unwind}) \frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi} \quad ([:=] \text{ assign}) \frac{\langle x_1 := \theta_1, \ldots, x_n := \theta_n\rangle\phi}{[x_1 := \theta_1, \ldots, x_n := \theta_n]\phi}$$

$$(\langle\cup\rangle \text{ choice}) \frac{\langle\alpha\rangle\phi \vee \langle\beta\rangle\phi}{\langle\alpha \cup \beta\rangle\phi} \quad (\langle?\rangle \text{ test}) \frac{H \wedge \psi}{\langle ?H\rangle\psi} \quad (\langle'\rangle \text{ ODE solve}) \frac{\exists t{\geq}0\, \big((\forall 0{\leq}\tilde{t}{\leq}t\, \langle\mathcal{S}(\tilde{t})\rangle H) \wedge \langle\mathcal{S}(t)\rangle\phi\big)}{\langle x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H\rangle\phi}\ {}^1$$

$$([\cup] \text{ choice}) \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} \quad ([?] \text{ test}) \frac{H \rightarrow \psi}{[?H]\psi} \quad ([']\text{ ODE solve}) \frac{\forall t{\geq}0\, \big((\forall 0{\leq}\tilde{t}{\leq}t\, \langle\mathcal{S}(\tilde{t})\rangle H) \rightarrow \langle\mathcal{S}(t)\rangle\phi\big)}{[x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H]\phi}\ {}^1$$

$$(\forall r \text{ all right}) \frac{\Gamma \vdash \phi(s(X_1, \ldots, X_n)), \Delta}{\Gamma \vdash \forall x\, \phi(x), \Delta}\ {}^2 \quad (\exists r \text{ exists right}) \frac{\Gamma \vdash \phi(X), \Delta}{\Gamma \vdash \exists x\, \phi(x), \Delta}$$

$$(\exists l \text{ exists left}) \frac{\Gamma, \phi(s(X_1, \ldots, X_n)) \vdash \Delta}{\Gamma, \exists x\, \phi(x) \vdash \Delta}\ {}^2 \quad (\forall l \text{ all left}) \frac{\Gamma, \phi(X) \vdash \Delta}{\Gamma, \forall x\, \phi(x) \vdash \Delta}$$

# Differential Dynamic Logic

(i∀ quantifier elimination) $$\frac{\Gamma \vdash \mathrm{QE}(\forall X\,(\Phi(X) \vdash \Psi(X))), \Delta}{\Gamma, \Phi(s(X_1,\dots,X_n)) \vdash \Psi(s(X_1,\dots,X_n)), \Delta}$$ 3

(i∃ eliminate existential) $$\frac{\Gamma \vdash \mathrm{QE}(\exists X\, \bigwedge_i (\Phi_i \vdash \Psi_i)), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \dots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}$$ 4

([] generalization) $$\frac{\Gamma \vdash [\alpha]\phi, \Delta \qquad \Gamma \vdash \forall^\alpha(\phi \to \psi), \Delta}{\Gamma \vdash [\alpha]\psi, \Delta}$$

(⟨⟩ generalization) $$\frac{\Gamma \vdash \langle\alpha\rangle\phi, \Delta \qquad \Gamma \vdash \forall^\alpha(\phi \to \psi), \Delta}{\Gamma \vdash \langle\alpha\rangle\psi, \Delta}$$

(*ind* loop invariant) $$\frac{\Gamma \vdash \phi, \Delta \qquad \Gamma \vdash \forall^\alpha(\phi \to [\alpha]\phi), \Delta \qquad \Gamma \vdash \forall^\alpha(\phi \to \psi), \Delta}{\Gamma \vdash [\alpha^*]\psi, \Delta}$$

(*con* loop convergence) $$\frac{\Gamma \vdash \exists v\, \varphi(v), \Delta \qquad \Gamma \vdash \forall^\alpha \forall v{>}0\, (\varphi(v) \to \langle\alpha\rangle\varphi(v-1)), \Delta \qquad \Gamma \vdash \forall^\alpha(\exists v{\le}0\, \varphi(v) \to \psi), \Delta}{\Gamma \vdash \langle\alpha^*\rangle\psi, \Delta}$$

(*DI* differential invariant) $$\frac{\Gamma, H \vdash F, \Delta \qquad \Gamma \vdash \forall^\alpha(H \to F'^{\theta_1}_{x_1'}\cdots^{\theta_n}_{x_n'}), \Delta}{\Gamma \vdash [x_1' = \theta_1, \dots, x_n' = \theta_n \,\&\, H]F, \Delta}$$

(*DV* differential variant) $$\frac{\Gamma \vdash [x_1' = \theta_1, \dots, x_n' = \theta_n \,\&\, \sim F]H, \Delta \qquad \Gamma \vdash \exists \varepsilon{>}0\, \forall^\alpha(\neg F \wedge H \to (F' \ge \varepsilon)^{\theta_1}_{x_1'}\cdots^{\theta_n}_{x_n'}), \Delta}{\Gamma \vdash \langle x_1' = \theta_1, \dots, x_n' = \theta_n \,\&\, H\rangle F, \Delta}$$ 5

(*DW* differential weaken) $$\frac{\Gamma \vdash \forall^\alpha(H \to \phi), \Delta}{\Gamma \vdash [x' = \theta \,\&\, H]\phi, \Delta}$$

(*DC* differential cut) $$\frac{\Gamma \vdash [x' = \theta \,\&\, H]C, \Delta \qquad \Gamma \vdash [x' = \theta \,\&\, (H \wedge C)]\phi, \Delta}{\Gamma \vdash [x' = \theta \,\&\, H]\phi, \Delta}$$

(*DA* differential auxiliaries) $$\frac{\phi \leftrightarrow \exists y\, \psi \qquad \Gamma \vdash [x' = \theta, y' = \vartheta \,\&\, H]\psi, \Delta}{\Gamma \vdash [x' = \theta \,\&\, H]\phi, \Delta}$$ 6

(*IA* auxiliary variable) $$\frac{\Gamma \vdash [y := \theta]\phi, \Delta}{\Gamma \vdash \phi, \Delta}$$ 7

(⟨:∗⟩ random) $$\frac{\exists X\, \langle x := X\rangle\phi}{\langle x := *\rangle\phi}$$ 8

([:∗] random) $$\frac{\forall X\, [x := X]\phi}{[x := *]\phi}$$ 8

# Overview

TECHNISCHE UNIVERSITÄT DRESDEN

symbolaris.com

Andre Platzer:
Logical Analysis of
Hybrid-Systems

**Why Hybrid Systems Verification**

**Differential Dynamic Logic**

**Hybrid Programs**

**Transition Semantics of Hybrid Programs**

**Proof Rules**

**KeYmaeraX**

# Exercise

Model a car that can either accelerate or brake.

Introduce a controller that keeps a safe braking distance.

Proof that the car will brake within this safe distance.

See Simple Car Examples in KeYmaeraX!

# Overview

TECHNISCHE UNIVERSITÄT DRESDEN

| | |
|---|---|
| Introduction<br>Mathematical Foundations (Differential Equations and Laplace Transformation) | Math |
| Control and Feedback<br>Transfer Functions and State Space Models | Physics |
| Poles, Zeros / PID Control<br>Stability, Root Locust Method, Digital Control | Feedback Control |
| Mixed-Criticality Scheduling and Real-Time Operating Systems (RTOS) | RTOS |
| Coordinating Networked Cyber-Physical Systems | CPS |
| Program Verification<br>Differential Dynamic Logic and KeYmaera X<br>**Differential Invariants** | Verification |