Towards a Refinement Type System for Hybrid Synchronous Program Verification

Jiawei Chen, José Luiz Vargas de Mendonça, Bereket Shimels Ayele, Bereket Ngussie Bekele, Shayan Jalili, Pranjal Sharma, Nicholas Wolhfeil, Yicheng Zhang and Jean-Baptiste Jeannin

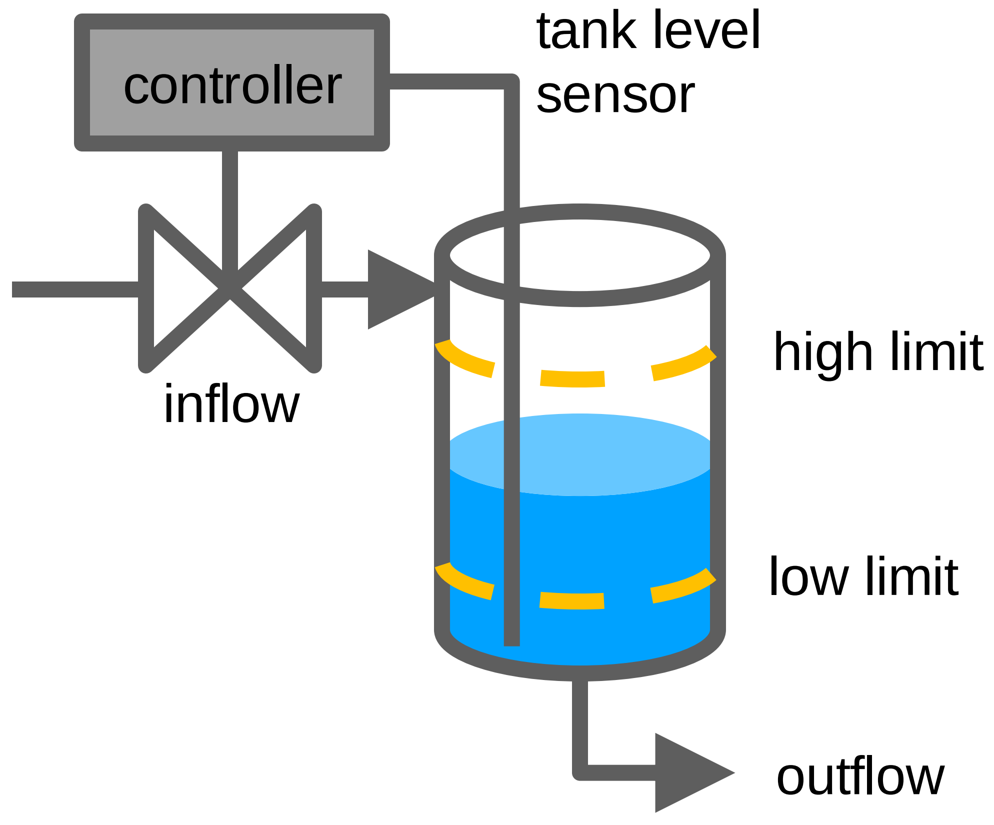
MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

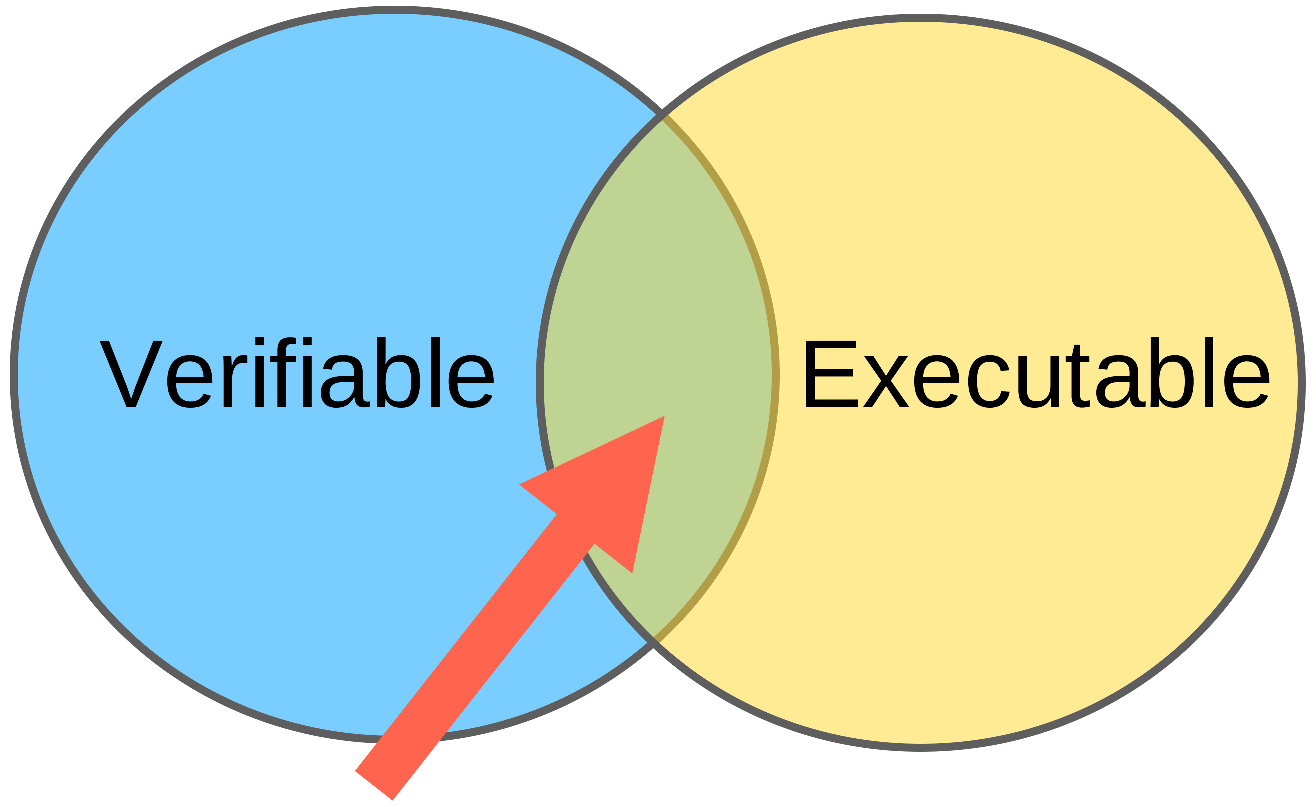
Context

- OCaml-based robotics platform with verification and real-time execution
- Preliminary work published at 2022 FTSCS workshop
- Full paper at ICFP 2024
  - Available, Reusable artifact

Background

- Cyber-Physical Systems (CPS): Software that **interacts with the physical environment**
  - Stringent safety requirements
- **CPS Verification**: Implementation?
- **CPS Implementation**: Verification?
- CPS designers **shouldn't need to choose** between **verification** and **implementation**
- Language with **both**?

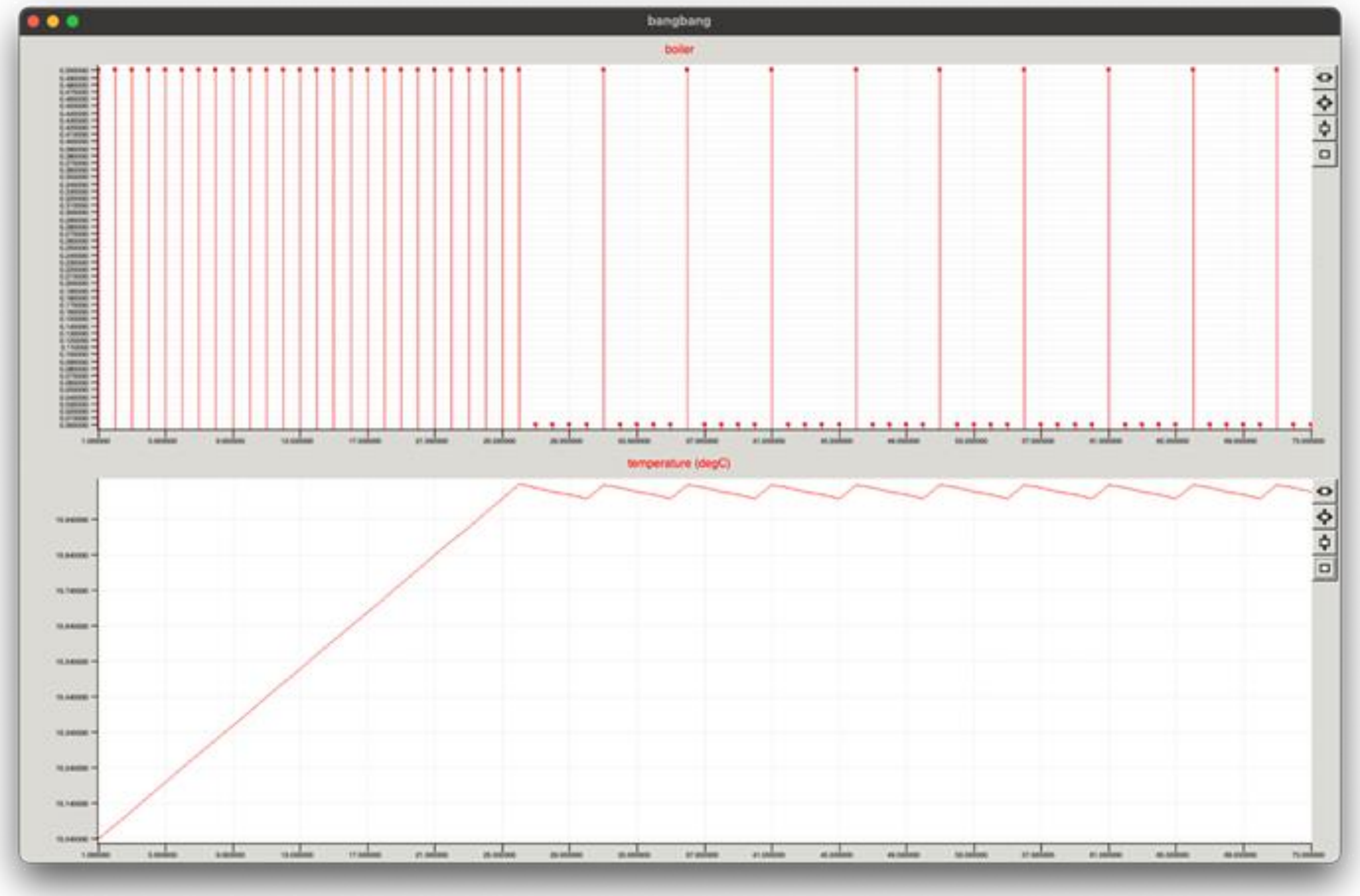


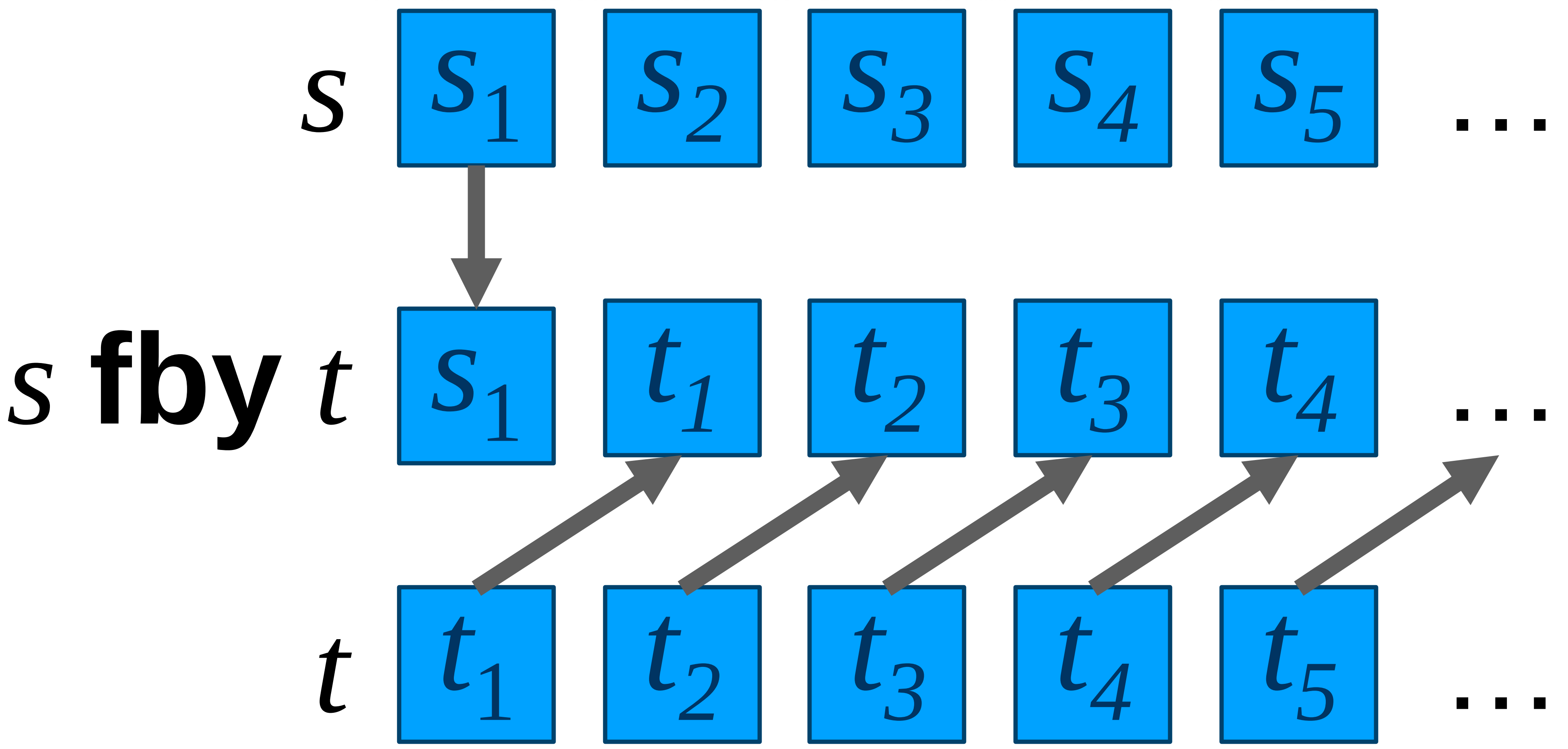


Synchronous Programming

[Caspi et al., POPL ’87; Bourke et al., HSCC ’13; Colaço et al., TASE ’17]

- **Proven track record** in industry
  - Lustre, SCADE, Esterel, Signal, etc.
- Data as **streams** (over time), programs as **stream manipulations**
- Our work is based on Zélus
  - Hybrid program **modeling and simulation**
  - Streams built using **unit delay** and **recursion**
  - Eventually: **hybrid systems verification**





Refinement Types

[Freeman and Pfenning, PLDI ’91; Rondon et al., PLDI ’08; Vazou et al., ICFP ’14; Jhala and Vazou, FTPL ’21]

- Inspired by Liquid Haskell
- Decidable **SMT-based** type checking and subtyping
- Type refinements on streams = **temporal properties**
- Support a **subset of LTL**
  - Interested in **safety properties**

Refinement Predicate

```
let x:{v:float| v >= 0.} = 3.14
```

Base Type

Term Variable

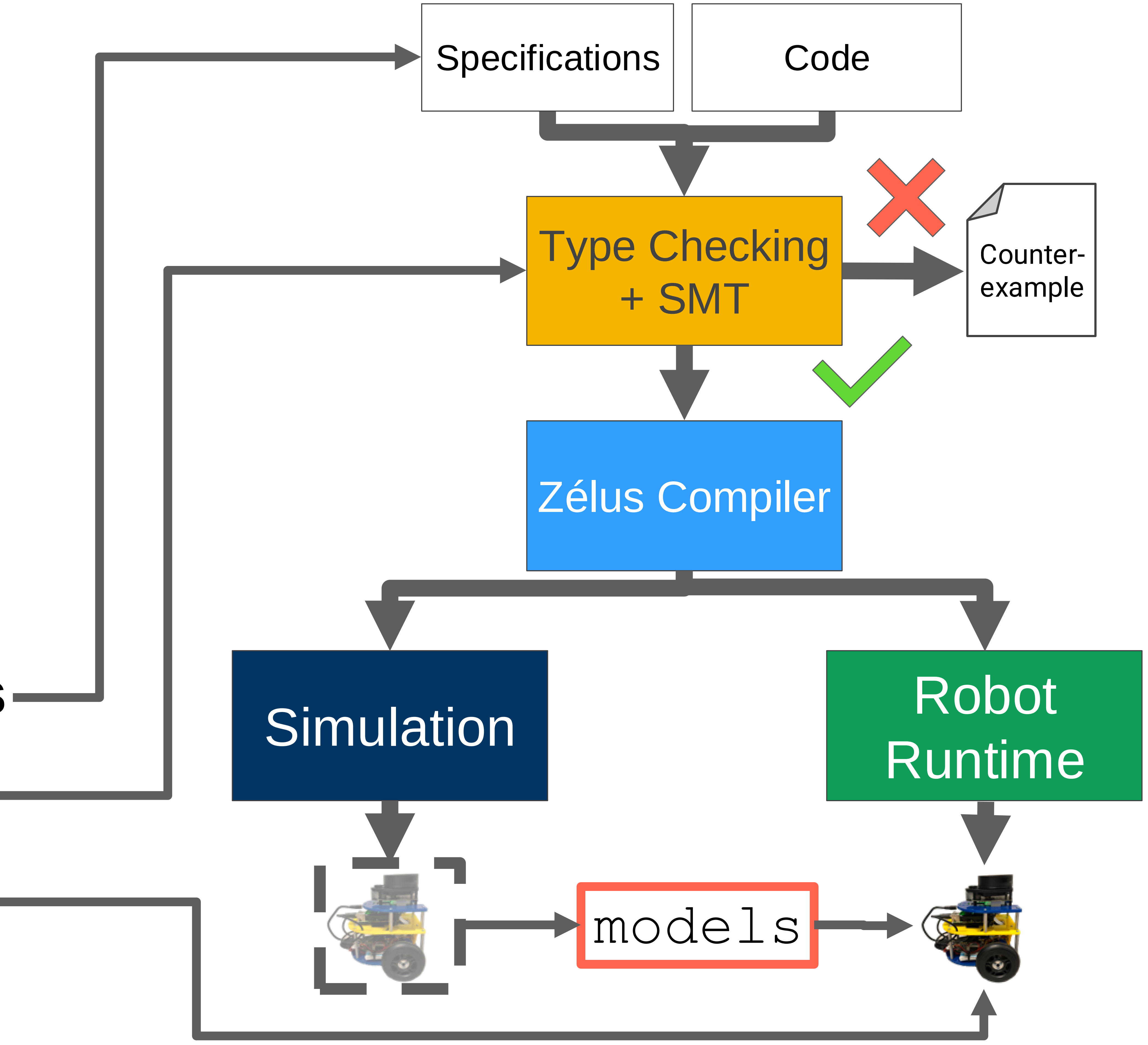
$$p, q ::= \text{true} \mid \text{false} \mid x \mid e_1 = e_2 \mid e_1 > e_2 \mid p \wedge q \mid \neg p$$

$$\varphi, \psi ::= p \mid \boxed{\Box\varphi \mid \bigcirc\varphi} \mid \varphi \wedge \psi$$

We formalize **refinement types** for a **synchronous** language, prove **type safety**, and implement verified programs on **physical robots**.

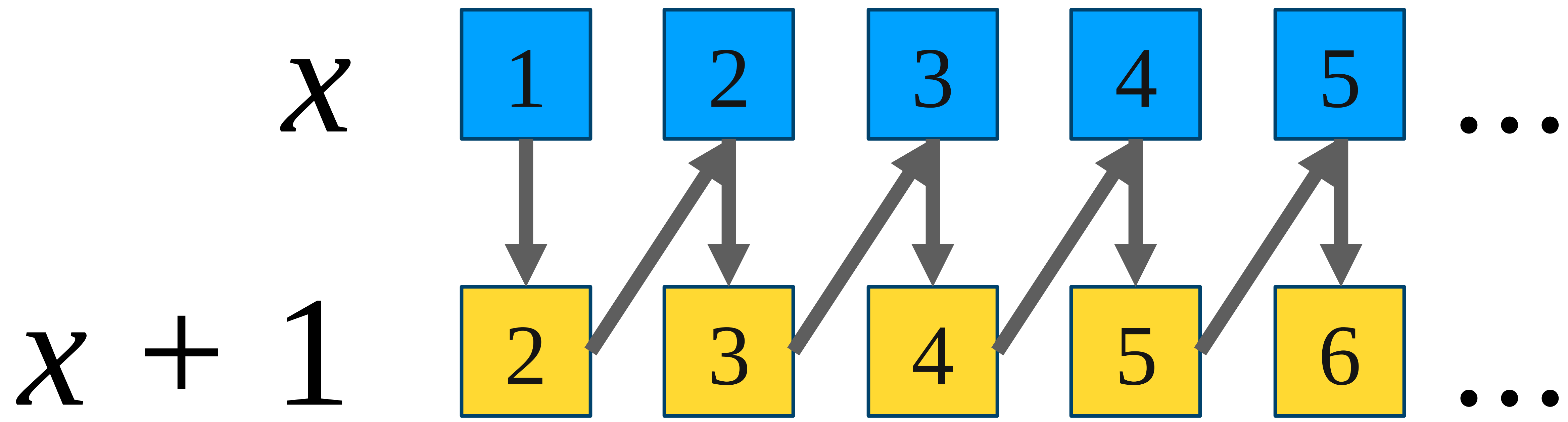
MARVeLus

- Method for Automated Refinement-Type Verification of Lustre
- **Separate compilation paths** for **simulation** and **execution**
- Verify a **discrete-time** subset of Zélus
- Our contributions:
  - Formal refinement **type system** and **semantics**
  - Type checker inside Zélus (+ artifact)
  - Demonstration of real-time execution

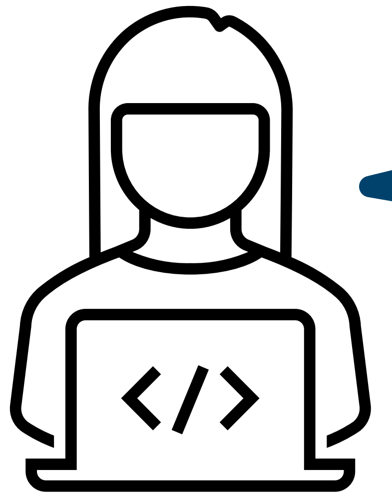


A Simple Example

```
let rec x
  = 1 fby (x + 1) in x
```

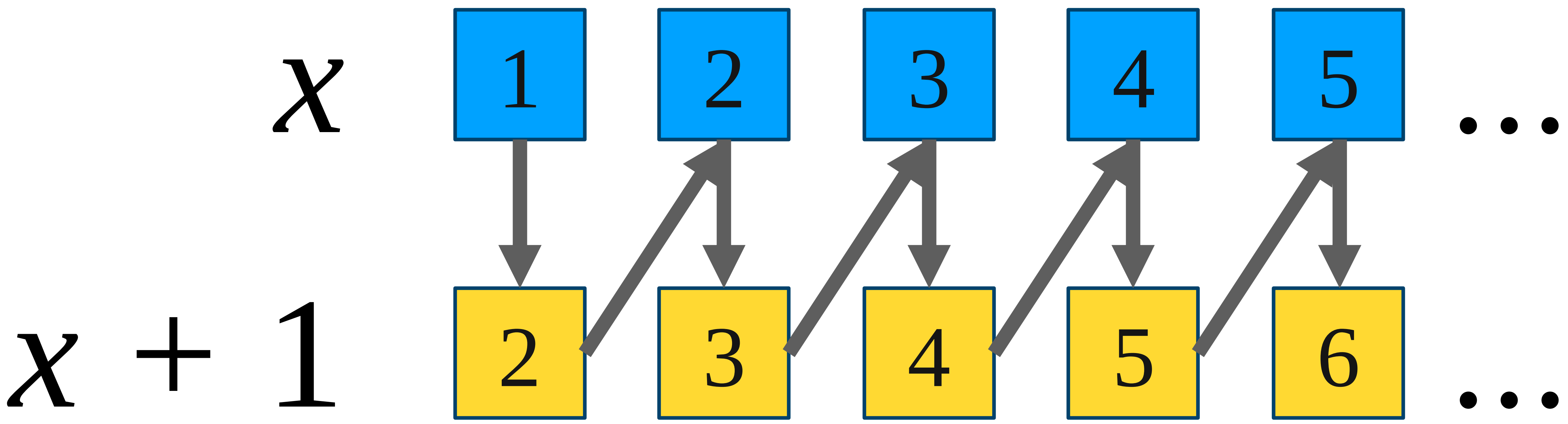


A Simple Example



"x should always be positive"

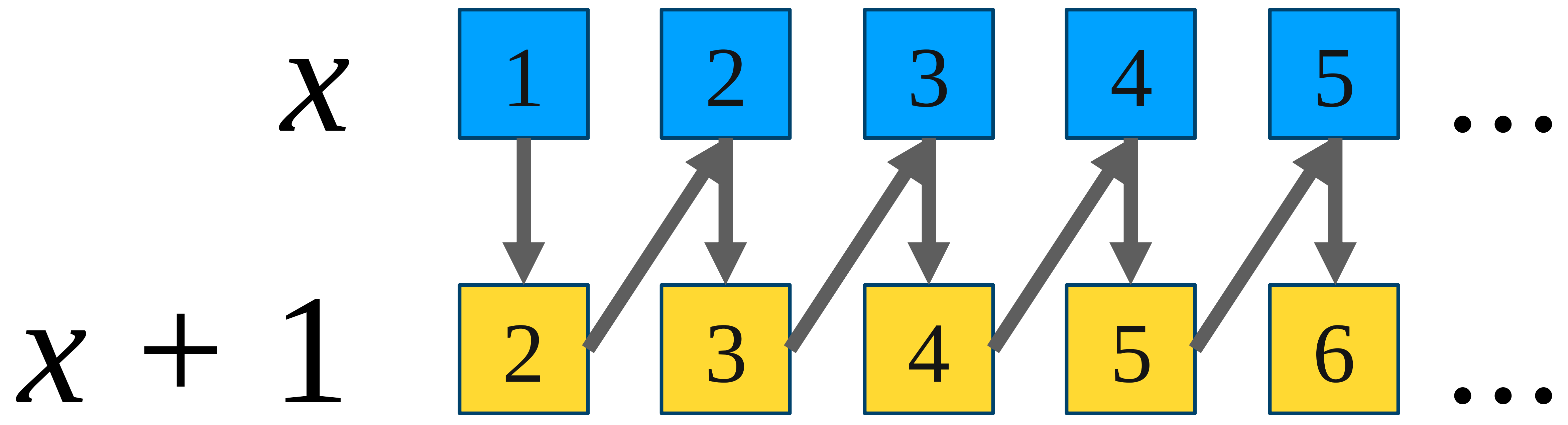
```
let rec x
  = 1 fby (x + 1) in x
```



A Simple Example

"x should always be positive"

```
let rec x:{v:int | □(x>0)}
  = 1 fby (x + 1) in x
```



A Simple Example

"x should always be positive"

```
let rec x:{v:int | □(x>0)}
  = 1 fby (x + 1) in x
```

$$1 > 0 \quad \land \quad \forall\, x{:}\mathrm{int}\,.\; x > 0 \Rightarrow (x+1) > 0$$

$$\forall\, x{:}\mathrm{int}\,.\; x > 0$$

SMT

Syntax-guided

MARVeLus Semantics

[Caspi and Pouzet, ICFP ’96; Caspi and Pouzet, CMCS ’98]

- Like Lustre semantics…
  - But adapted for type safety proofs with refinements
- Terms emit a **value** and **rewrite**
- One step **per unit time**

Environment **Term** **Value** **New Term**

$$\boxed{\boxed{S;\ \sigma} \vdash \boxed{e} \xhookrightarrow{\boxed{v}} \boxed{e'}}$$

Functions Terms

$$\frac{S;\ \sigma \vdash \boxed{e_1 \xhookrightarrow{v_1} e_1'}}{S;\ \sigma \vdash \boxed{e_1 \text{ fby } e_2} \xhookrightarrow{\boxed{v_1}} \boxed{\text{delay}(e_2)}}\ \text{(S-FBY)}$$

Semantics Rules

$$\boxed{S;\ \sigma \vdash e \xhookrightarrow{v} e'}$$

$$\frac{}{S;\ \sigma \vdash c \xhookrightarrow{c} c}\ \text{(S-CONST)} \qquad \frac{}{S;\ \sigma, x = h :: v \vdash x \xhookrightarrow{v} x}\ \text{(S-VAR)}$$

$$\frac{S;\ \sigma \vdash e_1 \xhookrightarrow{v_1} e_1'}{S;\ \sigma \vdash e_1 \text{ fby } e_2 \xhookrightarrow{v_1} \text{delay}(e_2)}\ \text{(S-FBY)} \qquad \frac{S,\ \text{prev}(\sigma) \vdash e \xhookrightarrow{v} e'}{S,\ \sigma \vdash \text{delay}(e) \xhookrightarrow{v} \text{delay}(e')}\ \text{(S-DELAY)}$$

$$\frac{S;\ \sigma, x = h :: \text{true} \vdash e_t \xhookrightarrow{v_t} e_t' \qquad S;\ \sigma, x = h :: \text{true} \vdash e_f \xhookrightarrow{v_f} e_f'}{S;\ \sigma, x = h :: \text{true} \vdash (\text{if } x \text{ then } e_t \text{ else } e_f) \xhookrightarrow{v_t} (\text{if } x \text{ then } e_t' \text{ else } e_f')}\ \text{(S-IF-T)}$$

$$\frac{S;\ \sigma, x = h :: \text{false} \vdash e_t \xhookrightarrow{v_t} e_t' \qquad S;\ \sigma, x = h :: \text{false} \vdash e_f \xhookrightarrow{v_f} e_f'}{S;\ \sigma, x = h :: \text{false} \vdash (\text{if } x \text{ then } e_t \text{ else } e_f) \xhookrightarrow{v_f} (\text{if } x \text{ then } e_t' \text{ else } e_f')}\ \text{(S-IF-F)}$$

$$\frac{S;\ \sigma, x = h :: \text{nil} \vdash e_1 \xhookrightarrow{v} e_1' \qquad S;\ \sigma, x = h :: v \vdash e_2 \xhookrightarrow{w} e_2'}{S;\ \sigma \vdash \text{let rec}_h\ x : \tau = e_1 \text{ in } e_2 \xhookrightarrow{w} \text{let rec}_{h::v}\ x : \text{tl}(\tau) = e_1' \text{ in } e_2'}\ \text{(S-LETREC)}$$

$$\frac{S;\ \sigma \vdash e_1 \xhookrightarrow{v} e_1' \qquad S;\ \sigma, x = h :: v \vdash e_2 \xhookrightarrow{w} e_2'}{S;\ \sigma \vdash \text{let}_h\ x : \tau = e_1 \text{ in } e_2 \xhookrightarrow{w} \text{let}_{h::v}\ x : \text{tl}(\tau) = e_1' \text{ in } e_2'}\ \text{(S-LET)}$$

$$\frac{}{S, f(x) = e; \sigma, y = h :: v \vdash f(y) \xhookrightarrow{e[x \mapsto v]} f(y)}\ \text{(S-APP)}$$

$$\frac{S, \sigma \vdash e \xhookrightarrow{v} e' \qquad \neg \text{ robot mode}}{S, \sigma \vdash e \text{ models } r \xhookrightarrow{v} e' \text{ models } r}\ \text{(S-MODELS)}$$

MARVeLus Types

- Like **refinement types**…
  - But with streams and temporal predicates
- **Syntax-guided** type safety…
  - But must also account for streams
  - Modified **progress and preservation**

Environment **Term** **Type**

$$\underbrace{\boxed{\underbrace{G}_{\text{Functions}};\ \underbrace{H}_{\text{Terms}}}}_{\text{Environment}} \vdash \underbrace{\boxed{e}}_{\text{Term}} : \underbrace{\boxed{\tau}}_{\text{Type}}$$

$$\frac{G;\ H \vdash e_1 : \{w : b \mid \boxed{\mathrm{hd}(\psi_1)}\} \qquad G;\ H \vdash e_2 : \{w : b \mid \boxed{\psi_2}\}}{G;\ H \vdash e_1 \text{ fby } e_2 : \{w : b \mid \underbrace{\boxed{\mathrm{hd}(\psi_1)}}_{\text{Beginning of } \psi_1} \wedge \underbrace{\boxed{\bigcirc \psi_2}}_{\text{All of } \psi_2 \text{ later}}\}}\ \text{(T-FBY)}$$

Selected Typing Rules

$$\boxed{G;\ H \vdash e : \tau}$$

$$\frac{}{G;\ H \vdash c : \{w : b \mid \square(w = c)\}}\ (\text{T-CONST}) \qquad \frac{G;\ H(x) = \{w : b \mid \psi\}}{G;\ H \vdash x : \{w : b \mid \psi \wedge \square(w = x)\}}\ (\text{T-VAR})$$

$$\frac{G;\ H \vdash e_1 : \{w : b \mid \mathsf{hd}(\psi_1)\} \qquad G;\ H \vdash e_2 : \{w : b \mid \psi_2\}}{G;\ H \vdash e_1 \text{ fby } e_2 : \{w : b \mid \mathsf{hd}(\psi_1) \wedge \bigcirc\psi_2\}}\ (\text{T-FBY})$$

$$\frac{G;\ \mathsf{prev}(H) \vdash e : \tau}{G;\ H \vdash \mathsf{delay}(e) : \tau}\ (\text{T-DELAY})$$

$$\frac{G;\ H \vdash \tau_1 \qquad G;\ H \vdash e_1 : \tau_1 \qquad G;\ H, x : \tau_1 \vdash e_2 : \tau_2}{G;\ H \vdash \mathsf{let}_h\ x : \tau_1 = e_1 \text{ in } e_2 : \tau_2}\ (\text{T-LET})$$

$$\frac{G;\ H \vdash \tau_1 \qquad G;\ H, x : \tau_1 \vdash e_1 : \tau_1 \qquad G;\ H, x : \tau_1 \vdash e_2 : \tau_2}{G;\ H \vdash \mathsf{let}_h \text{ rec } x : \tau_1 = e_1 \text{ in } e_2 : \tau_2}\ (\text{T-LETREC})$$

$$\frac{G;\ H \vdash x_c : \mathsf{bool} \qquad \begin{array}{c} G;\ H \vdash e_t : \{w : b \mid \mathsf{impl}(x_c, \psi)\} \\ G;\ H \vdash e_f : \{w : b \mid \mathsf{impl}(\neg x_c, \psi)\} \end{array}}{G;\ H \vdash \text{if } x_c \text{ then } e_t \text{ else } e_f : \{w : b \mid \psi\}}\ (\text{T-IF})$$

Typing Rules, Continued

$$\frac{G, f : (x : \{w_1 : b_1 \mid \varphi_1\} \to \{w_2 : b_2 \mid \varphi_2\});\ H \vdash x : \{w_1 : b_1 \mid \Box\varphi_1\}}{G, f : (x : \{w_1 : b_1 \mid \varphi_1\} \to \{w_2 : b_2 \mid \varphi_2\});\ H \vdash f\ y : \{w_2 : b_2 \mid \Box\varphi_2[x \mapsto y]\}}\ \text{(T-APP)}$$

$$\frac{G;\ H \vdash e : \tau}{G;\ H \vdash e \text{ models } r : \tau}\ \text{(T-MODELS)} \qquad \frac{}{G; H \vdash \text{nil} : \tau}\ \text{(T-NIL)}$$

$$\frac{G; H \vdash e : \tau \qquad G; H \vdash \tau \leq \tau' \qquad G; H \vdash \tau'}{G; H \vdash e : \tau'}\ \text{(T-SUB)}$$

Type Safety

Under some assumptions about the environment, **well-typed terms**

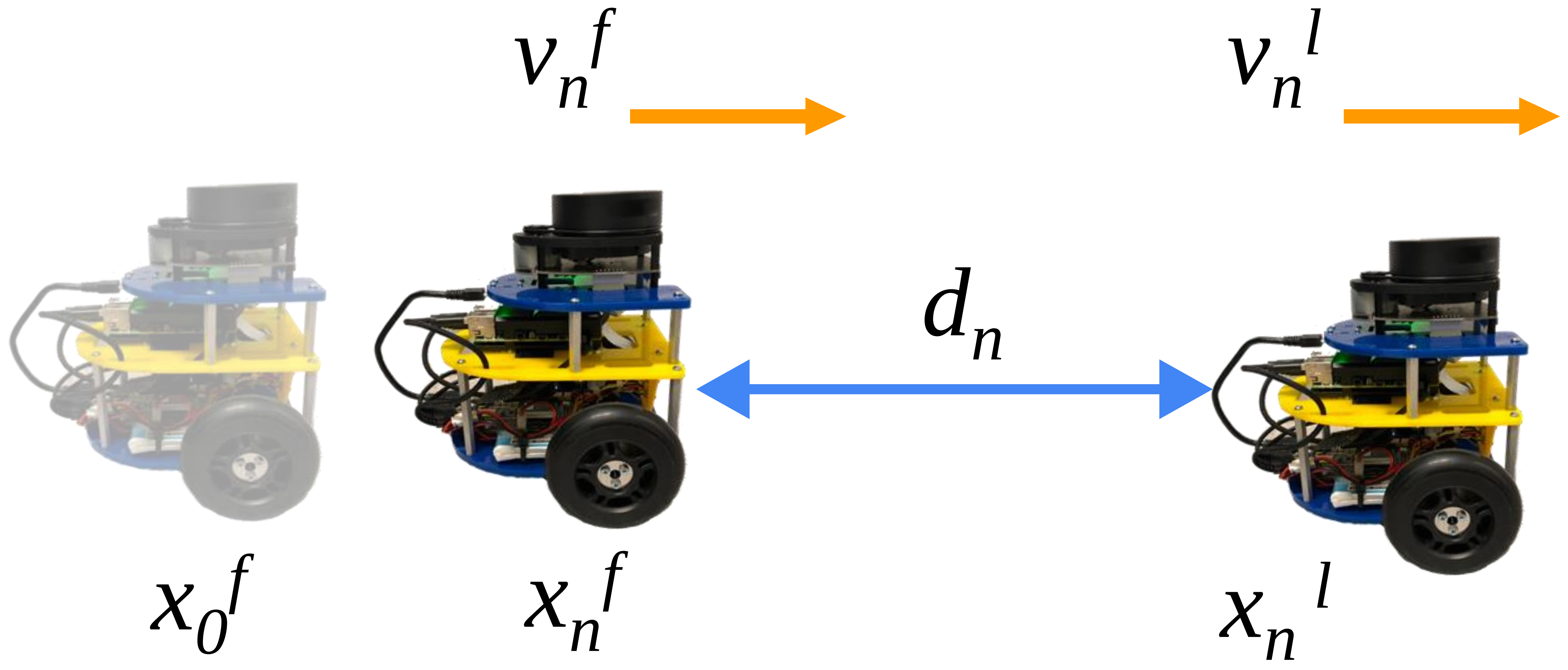
THEOREM 1 (TYPE SAFETY). $\boxed{\textit{If } H \textit{ corresponds with } \sigma \textit{ and } G \textit{ corresponds with } S,}$ *and* $\boxed{\textit{if } G; H \vdash e : \tau,}$ *then* $\boxed{\exists e' \textit{ such that } S; \sigma \vdash e \overset{v}{\hookrightarrow} e' \textit{ where } v \textit{ is a value,}}$ $\boxed{G, H \vdash v : \mathsf{hd}(\tau), \textit{ and } G, \mathsf{tl}(H) \vdash e' : \mathsf{tl}(\tau)}$

always step to terms **which are well-typed in the <u>next time step</u>**

Verified Adaptive Cruise Control

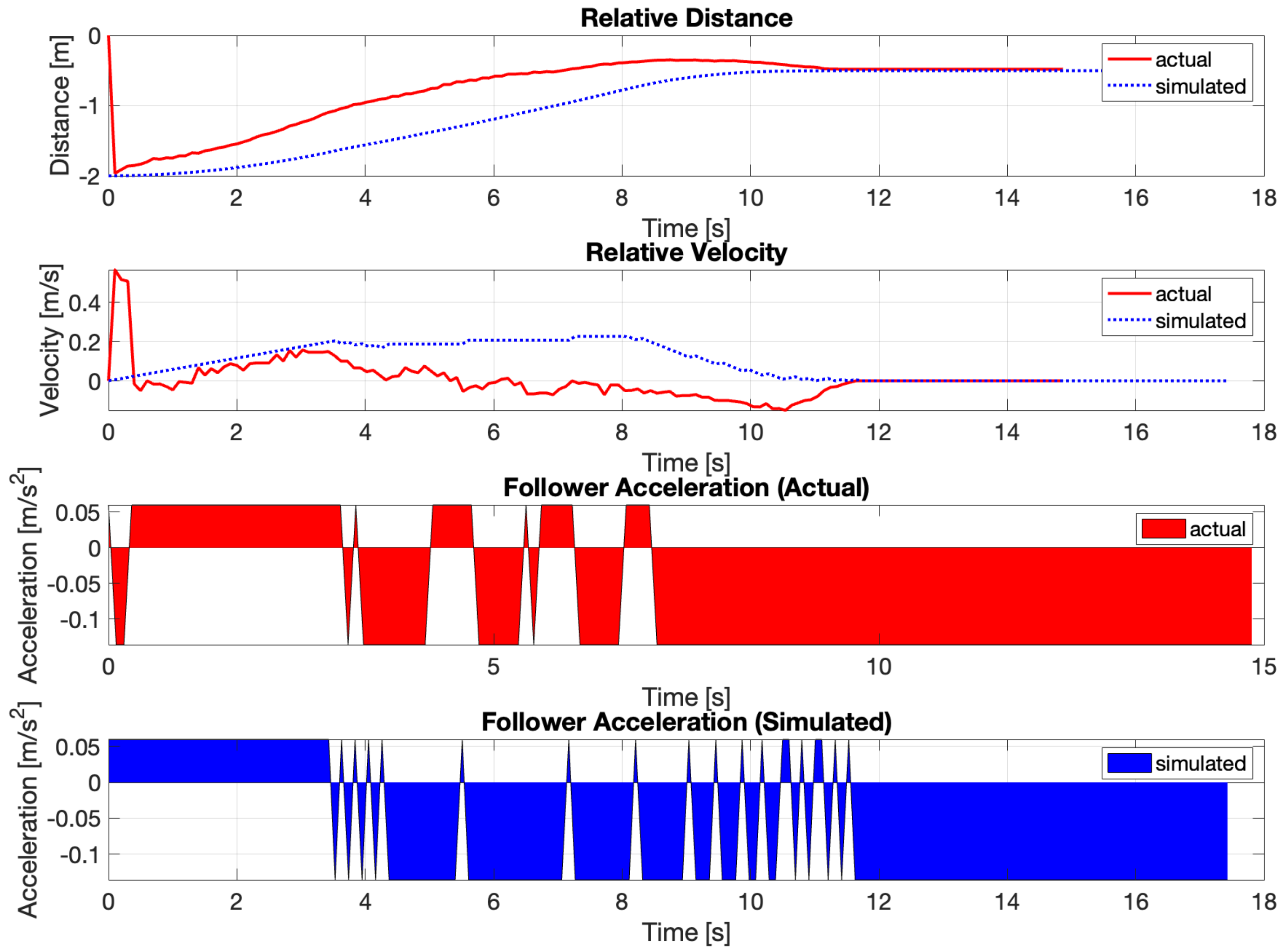
[Loos et al., FM ’11]

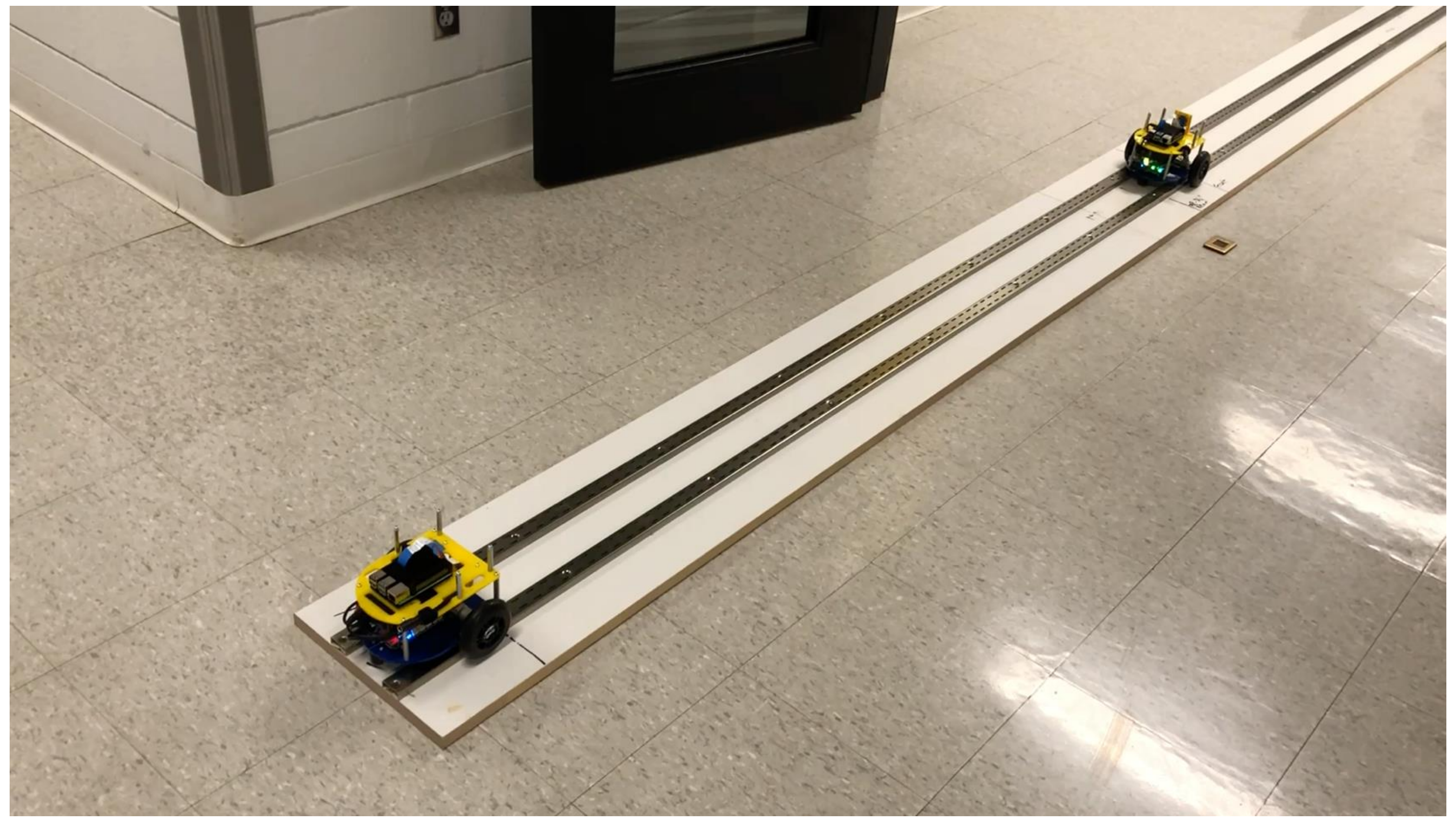
- Verified **autonomous braking controller** for Adaptive Cruise Control
- **Safety Property**: Never crash into the obstacle: $\square(d > 0)$
- Hardware abstractions and sensors trusted



Experiment

- Verified autonomous braking demonstrated on **physical robots**
- Can follow **moving vehicles**
- Collision-free in **all runs**





Future Work

Specification

- Hybrid programs
- Full LTL specifications

Verification / Compilation

- Invariant generation
- Verified compiler

Execution

- More hardware support
- Runtime monitoring

Future Work: Specifications

- Currently we support a limited subset of LTL
  - Necessary for “predicate splitting”
  - May be tricky to extend to hybrid
- Must have:
  - Type safety proof
  - Predictable compile-time verification
- Possible ideas:
  - Synchronous Observers

$$p, q ::= \mathsf{true} \mid \mathsf{false} \mid x \mid e_1 = e_2 \mid e_1 > e_2 \mid p \wedge q \mid \neg p$$

$$\varphi, \psi ::= p \mid \boxed{\Box\varphi \mid \bigcirc \varphi} \mid \varphi \wedge \psi$$

$$\frac{G;\ H \vdash e_1 : \{w : b \mid \boxed{\mathsf{hd}(\psi_1)}\} \qquad G;\ H \vdash e_2 : \{w : b \mid \boxed{\psi_2}\}}{G;\ H \vdash e_1\ \mathsf{fby}\ e_2 : \{w : b \mid \boxed{\mathsf{hd}(\psi_1)} \wedge \boxed{\bigcirc\psi_2}\}}\ \text{(T-FBY)}$$

Beginning of ψ_1 **All of** ψ_2 **later**

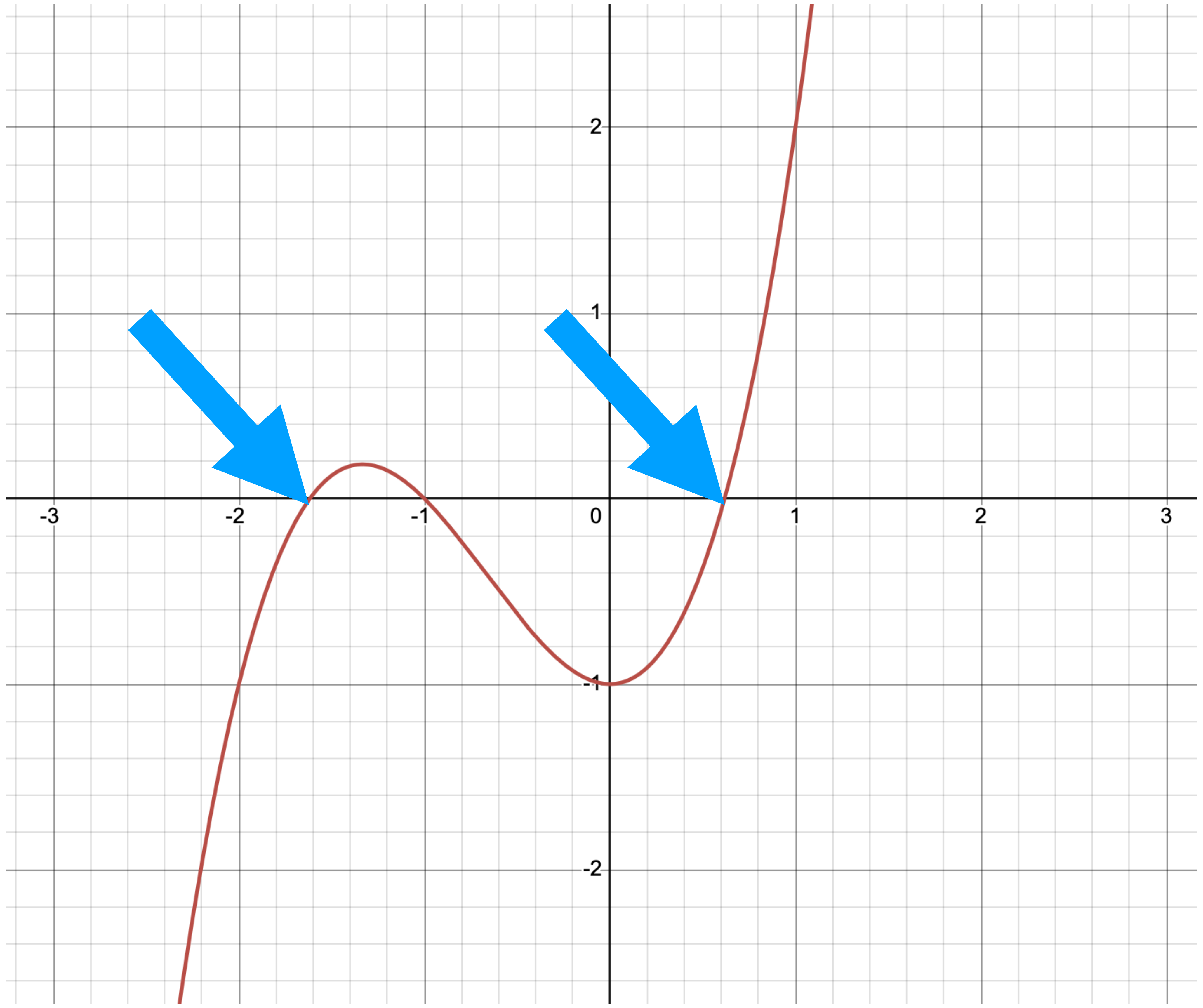
$\psi[0]$	$\psi[1]$	$\psi[2]$	$\psi[3]$	$\psi[4]$	…

$\equiv$

$\psi_1[0]$	$\psi_2[0]$	$\psi_2[1]$	$\psi_2[2]$	$\psi_2[3]$	…

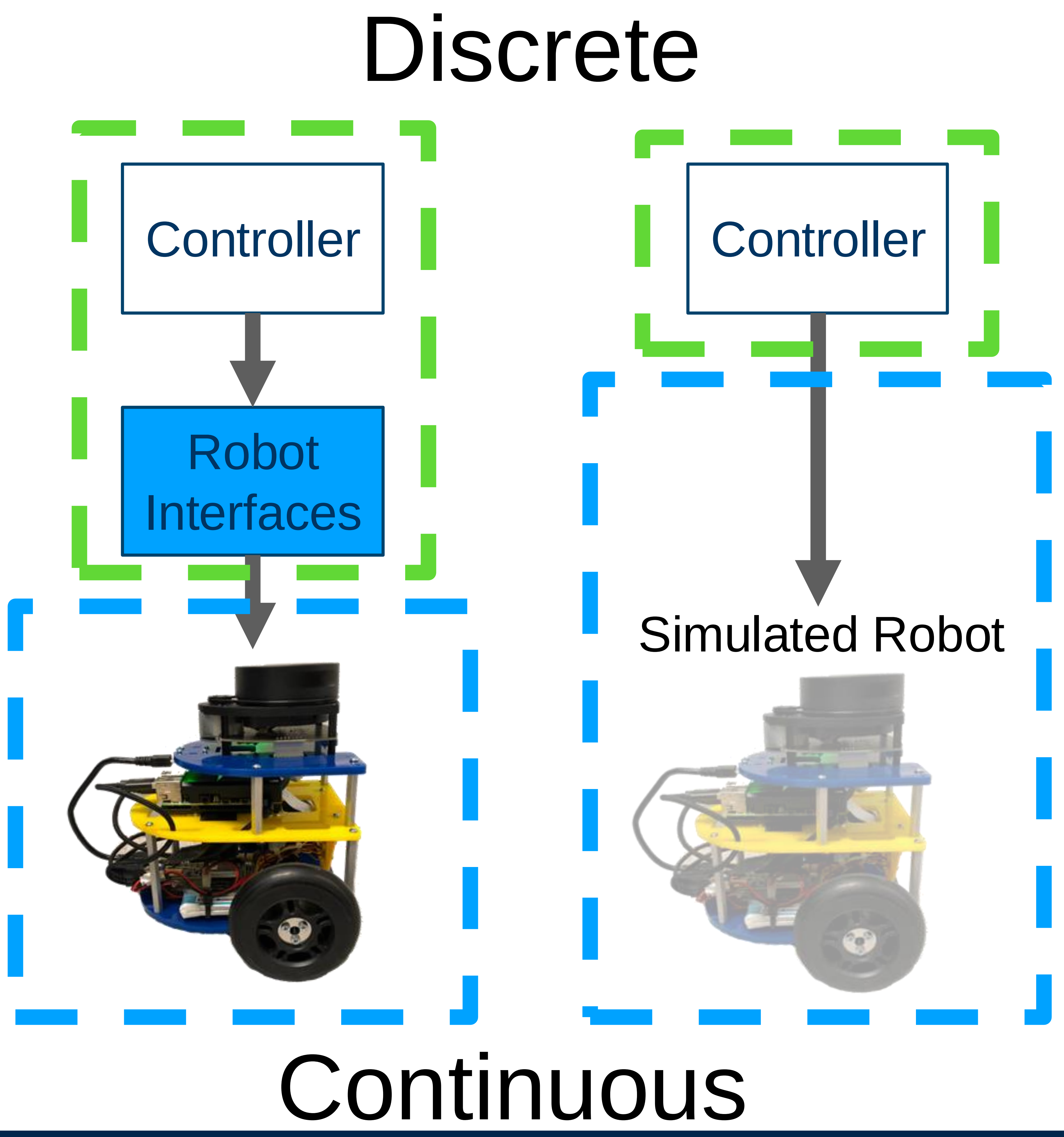
Future Work: Specifications (cont'd)

- Hybrid
  - Model real systems more accurately
  - Upwards Zero Crossings
- Possible Ideas:
  - Differential Dynamic Logic (dL)
    - Parallelism?
  - Barrier Certificates



Future Work: Robotics Integrations

- Provide an intuitive interface for system designers
- Current implementation:
  - Getter and setter functions to access robot variables
  - Deterministic model for verification
- Re-use as much verified code as possible
  - Ideally, reuse the entire verified controller



Automated Braking in MARVeLus

$$\mathrm{needToBrake}(d,\ v) = \left(d - \frac{v^2}{2b} - v \cdot dt - \frac{v \cdot dt}{2}\right) \leq 0$$

“Are we too close to do anything other than brake?”

```
let vfi  :   {v: float | v > 0.} = 0.8;
let dt   :   {v: float | v > 0.} = 0.1;
let b    :   {v: float | v > 0.} = 0.136;
let xl   :   {v: float | v > 0.} = 5.;
```

Constants

```
let rec (df, vf, af):
  {(d:float)*(v:float)*(a:float)|
```

Base Type Spec

$(\ (d > 0)$

Safety Condition

$\wedge ((\neg \mathrm{needToBrake}(d,\ v) \wedge (a \leq 0)) \vee (\mathrm{needToBrake}(d,\ v) \wedge (a = -b)))$

Follower Decisions

$\wedge \left(d - \frac{v^2}{2b} - \frac{v \cdot dt}{2}\right) > 0 \wedge (v \geq 0)$

Dynamics Invariant

```
}) = (xl, vfi, 0.) fby
```

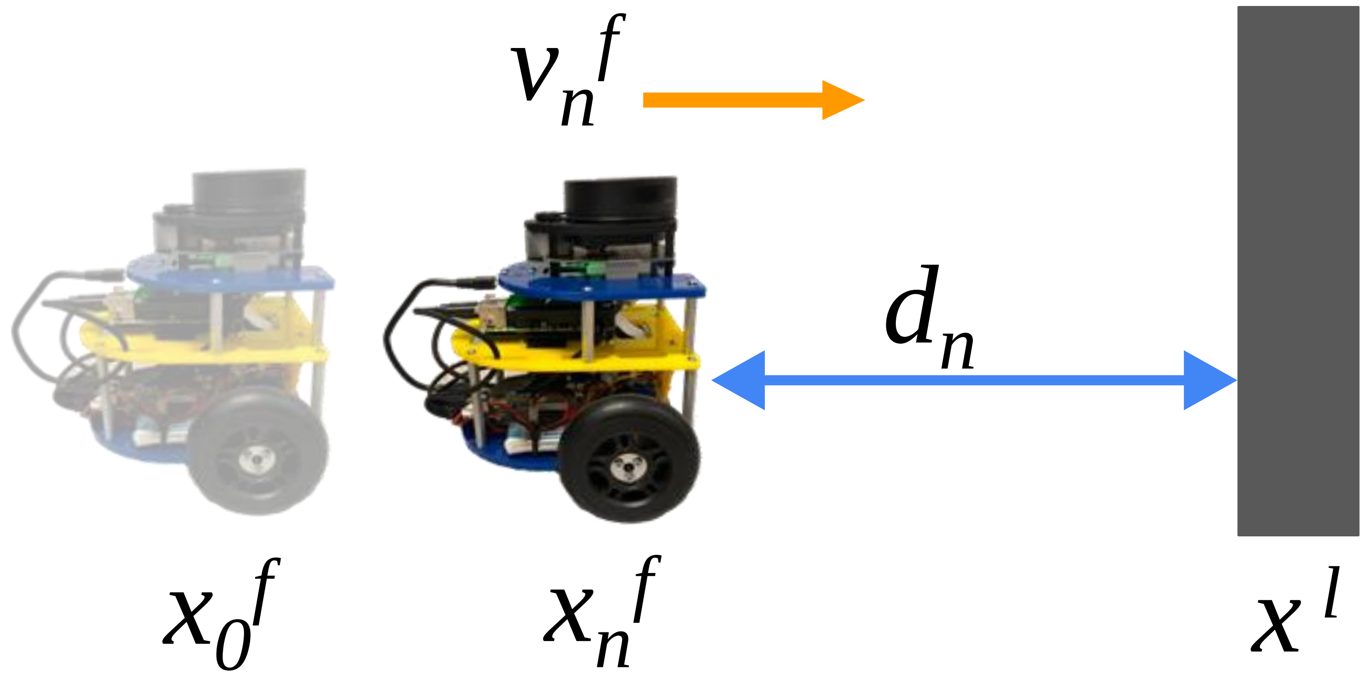
Initial State

```
  let v_next = max(vf + (af * dt), 0.)   models   (robot_get “vel”)    in
  let d_next = df -. (v_next * dt)       models   (robot_get “dist”)   in
```

Dynamics Evolution

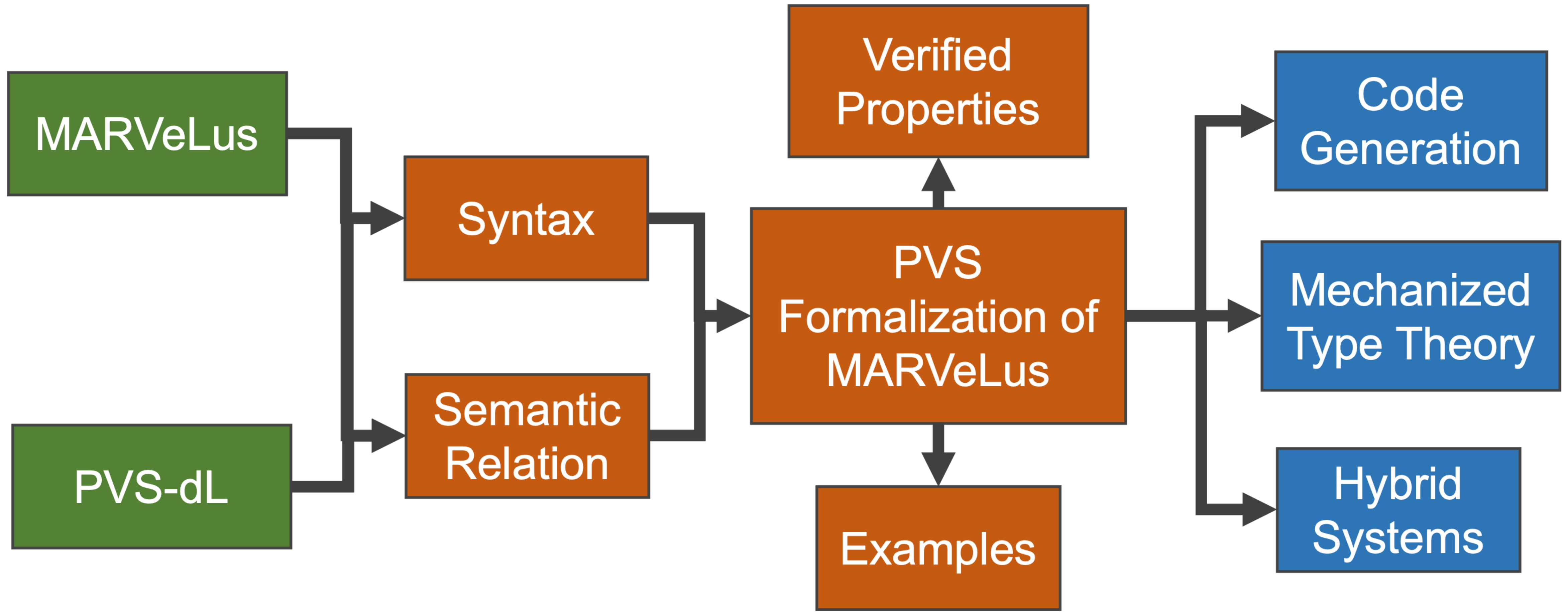
```
  (d_next, v_next,
      if needToBrake(d_next, v_next) then -b else amax)
  in (robot_str “brake” af); (df, vf, af)
```

Follower Control



Future Work: Proof Assistant Embedding

- Goal: Embed MARVeLus in a proof assistant such as PVS
  - Build off existing dL embedding for hybrid
  - Mechanize the type system
  - Enable code generation from specifications (inspired by PRECiSA)
- Existing work embeds Lustre in Coq and PVS



Summary

- Robots and other CPS need **formal verification**
- MARVeLus provides **formal verification** and **execution** in a **unified robotics platform**
- **Synchronous programs** can be enhanced with **refinement types**
- Verified MARVeLus programs can execute on **real hardware**

Acknowledgements

- The Zélus Team
  - Marc Pouzet and Timothy Bourke
- Ranjit Jhala and Niki Vazou
- NASA Langley Formal Methods
  - Tanner Slagel, Lauren White, Laura Titolo, Aaron Dutle and César Muñoz
- Many more

Contacts

- Email: chenjw@umich.edu
- Website: www.jchenrobotics.com
- Our Lab: https://marvl.engin.umich.edu

ICFP 2024 Paper