Formal Design Theories and Tools for Safety-Critical Cyber-Physical Systems

Naijun Zhan

School of Computer Science, Peking University

2024 Summer School on Trustworthy Software

SEI of ECNU, Shanghai, July 9-12

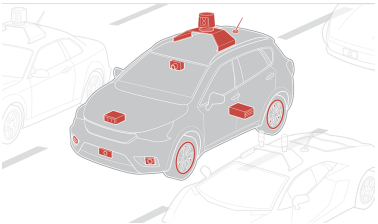
Cyber-Physical Systems (CPS)

"[…] cyber-physical systems (CPS) refers to a new generation of systems with integrated computational and physical capabilities that can interact with humans through many new modalities. The ability to interact with, and expand the capabilities of, the physical world through computation, communication, and control is a key enabler for future technology developments."

[Radhakisan Baheti and Helen Gill : CPS. The Impact of Control Technology, 2011]

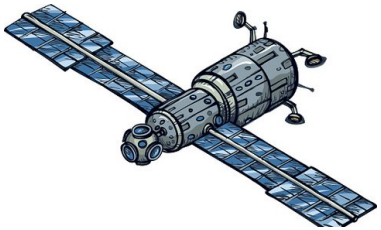
Cyber-Physical Systems (CPS)

An open, interconnected form of embedded systems; many are safety-critical.



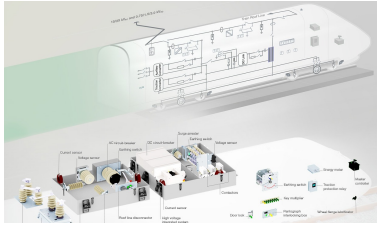
©TheOneBrief

automobiles



©clipartlogo

spacecrafts

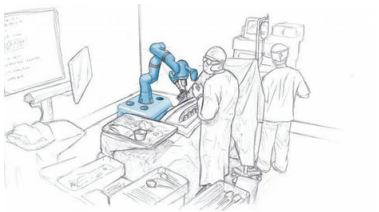


©Sécheron

high-speed rail

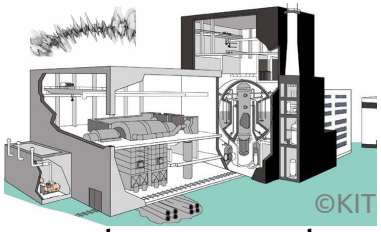
©VectorStock

nuclear reactors



©Scoop.it

robot surgeon

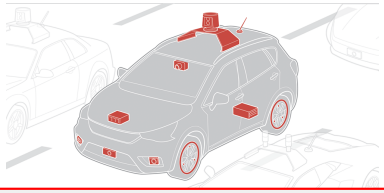


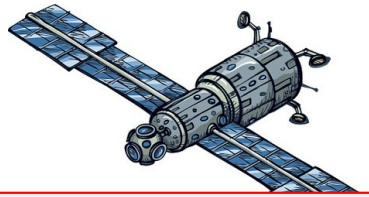
©KIT

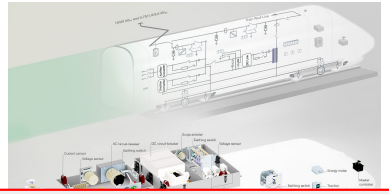
robust control

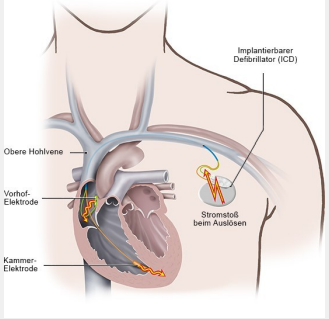
Cyber-Physical Systems (CPS)

An open, interconnected form of embedded systems; many are safety-critical.









212 patients died of defibrillator failure

(USA, 1997 – 2003)

40 passengers died plus 172 injured

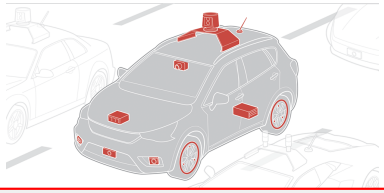
(China, 2011.7.23)

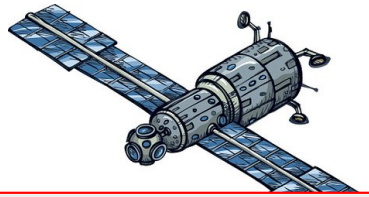
31 billion Yen loss on ASTRO-H

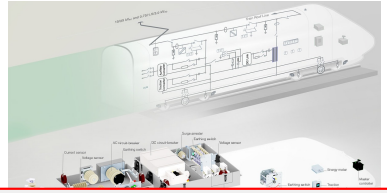
(Japan, 2016.3.26)

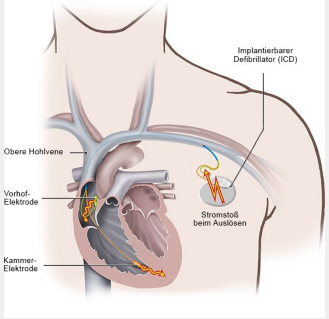
Cyber-Physical Systems (CPS)

An open, interconnected form of embedded systems; many are safety-critical.









212 patients died of defibrillator failure

(USA, 1997 – 2003)

40 passengers died plus 172 injured

(China, 2011.7.23)

31 billion Yen loss on ASTRO-H

(Japan, 2016.3.26)

"How can we provide people with CPS they can bet their lives on?" — Jeannette M. Wing, former AD for CISE at NSF

Model-based Design (MBD)

How to design reliable complex embedded systems is a grand challenge for computer science and control theory. **Major Challenges : complexity** & **reliability**

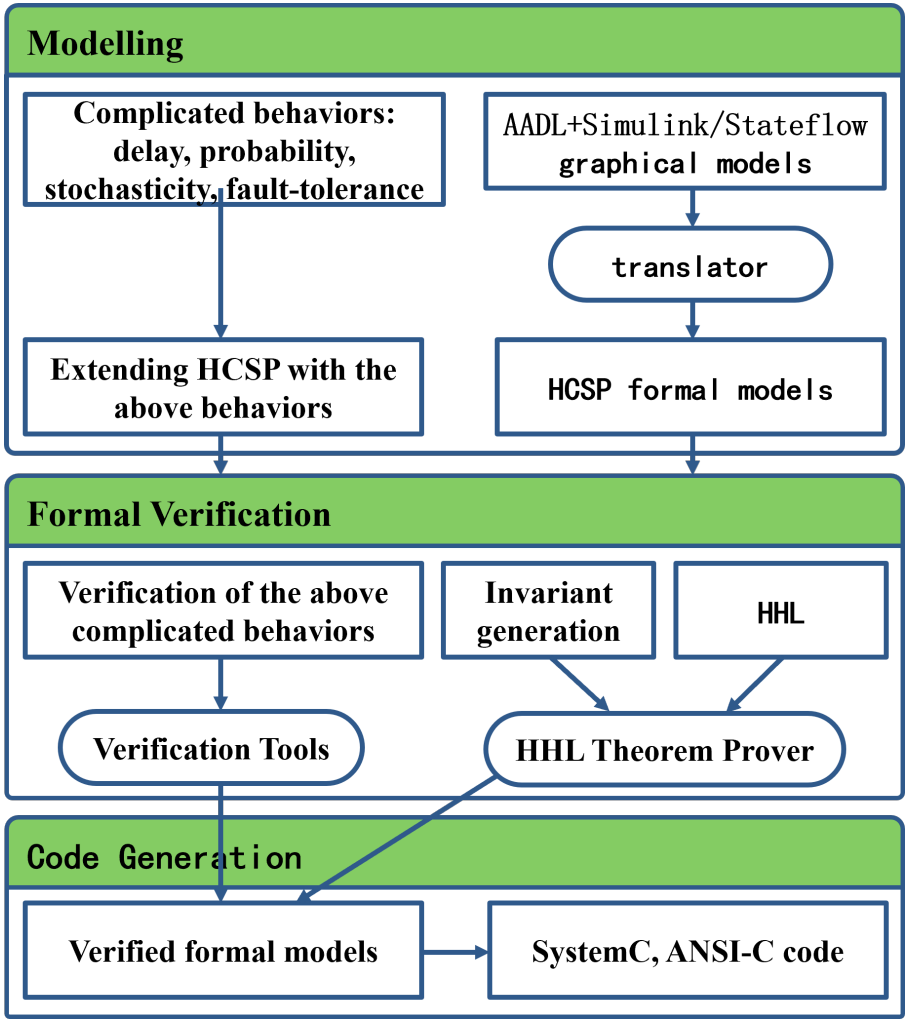
Divide-and-conquer by MBD

- Horizontal : Composition/decomposition
- Vertical : Abstraction/refinement

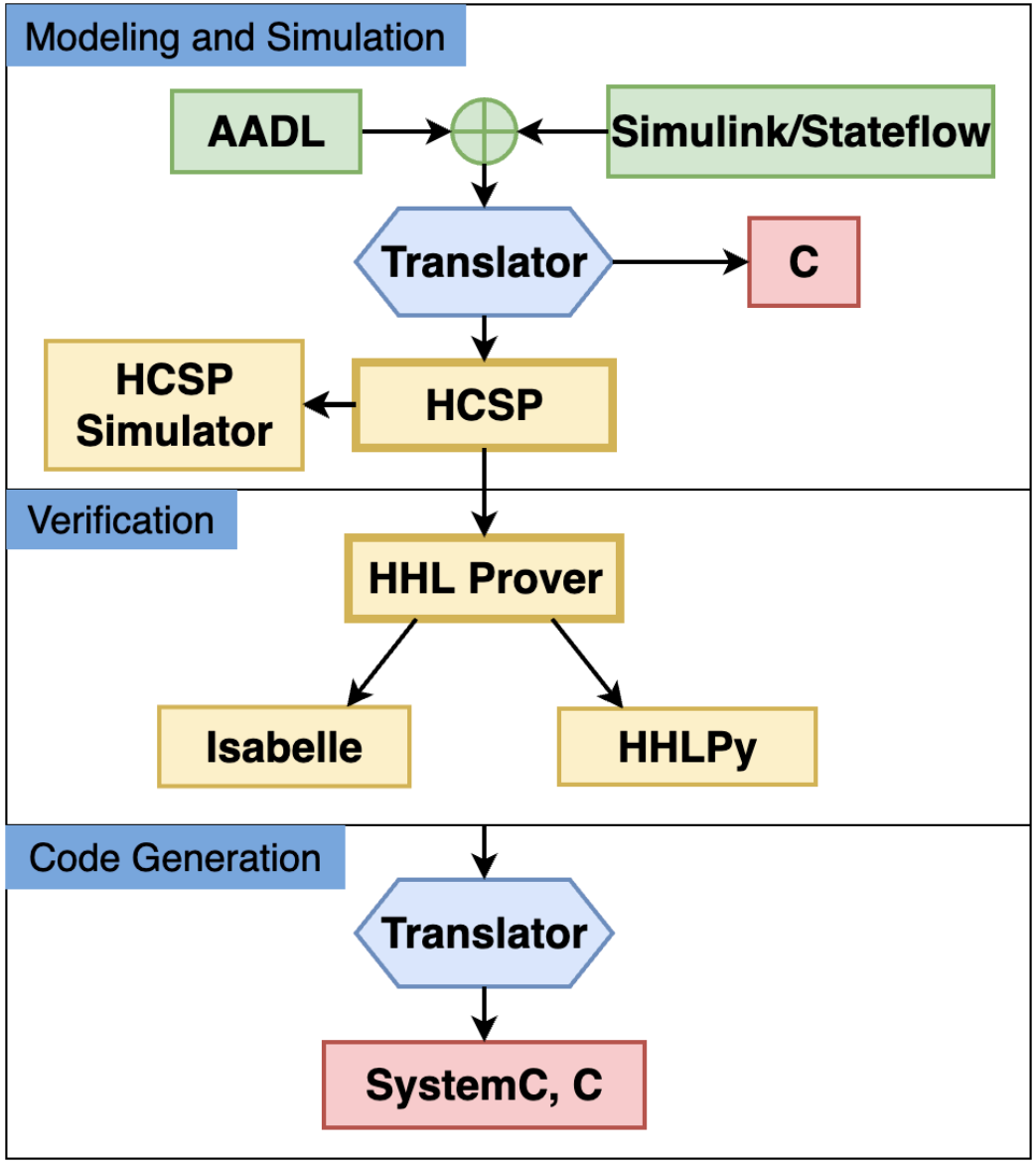
Design paradigms of MBD

- **Simulation-based design**
  - easy to use, efficiency; incomplete
- **Formal design**
  - reliable; difficult, expensive

Is it possible to combine these two paradigms to exploit their advantages and avoid their disadvantages?



Toolchain : MARS [ICCPS'24]



Schedule and References

Schedule

- Part 1 : Modeling and Simulation
- Part 2 : Verification : Hybrid Hoare Logic + Invariant Generation
- Part 3 : Code Generation
- Part 4 : Synthesis

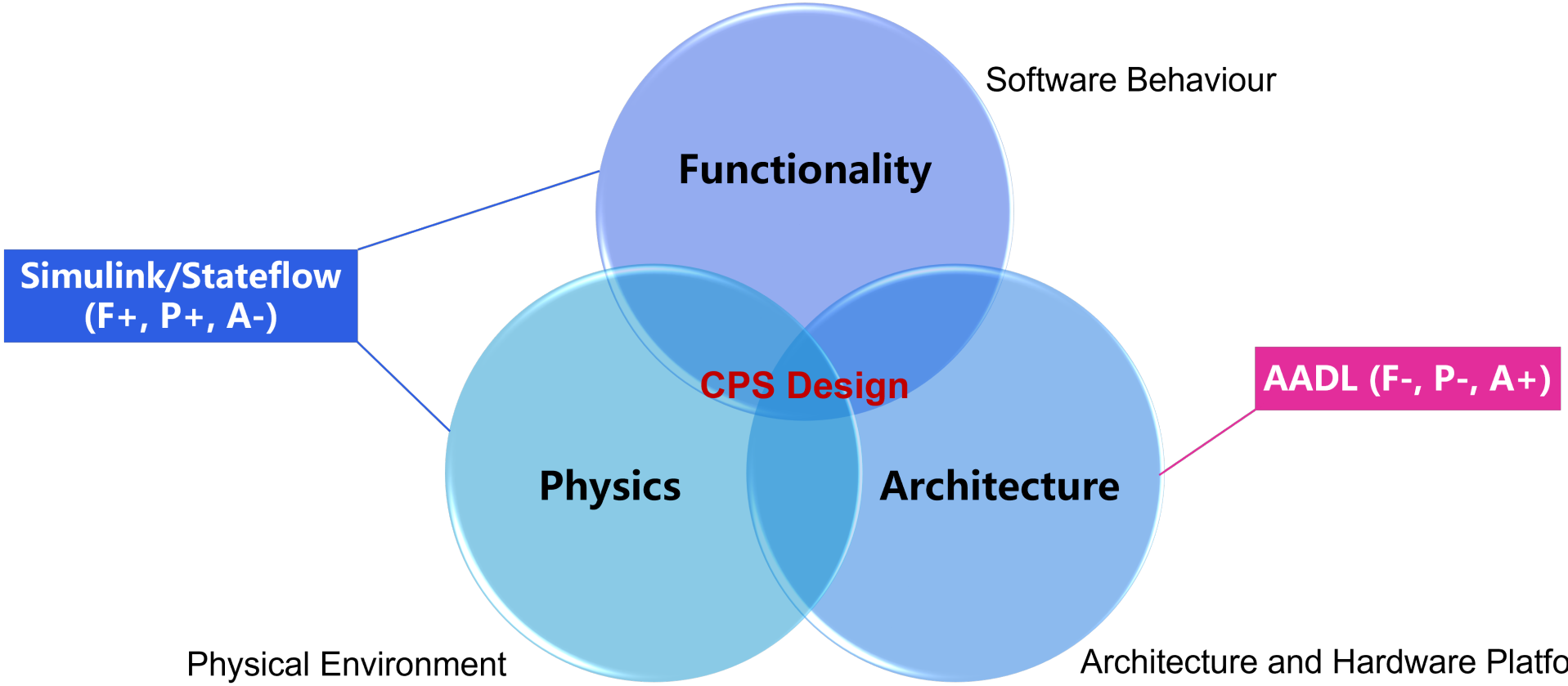
References

- N. Zhan, S. Wang and H. Zhao (2016) : Formal Verification of Simulink/Stateflow Diagrams. Springer-Verlag.
- J. Liu, J. Lv, Z. Quan, N. Zhan, et al. (2010) : A calculus for HCSP. APLAS 2010.
- J. Liu, N. Zhan and H. Zhao (2011) : Computing semi-algebraic invariants for polynomial dynamical systems. EMSOFT'11.
- H. Zhao, N. Zhan, D. Kapur, and K.G. Larsen (2012) : A "hybrid" approach for synthesizing optimal controllers of hybrid systems : A case study of the oil pump industrial example. FM 2012.
- H. Zhao, M. Yang, N. Zhan, et al. (2014) : Formal verification of a descent guidance control program of a lunar lander. FM 2014.
- G. Yan, L. Jiao, S. Wang, L. Wang and N. Zhan (2020) : Automatically generating SystemC code from HCSP formal mdoels. ACM TOSEM.
- X. Xu, B. Zhan, S. Wang, X. Jin, J.-P. Talpin and N. Zhan (2021) : Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow. TCS.
- X. Xu, B. Zhan, S. Wang, J.-P. Talpin and N. Zhan (2022) : Semantics foundation for cyber-physical systems using higher-order UTP. ACM TOSEM.
- Q. Wang, M. Chen, B. Xue, N. Zhan, J.-P. Katoen (2022) : Encoding inductive invariants as barrier certificates synthesis via difference-of-convex programming. Inf&Com.

Part 1 : Modeling and Simulation

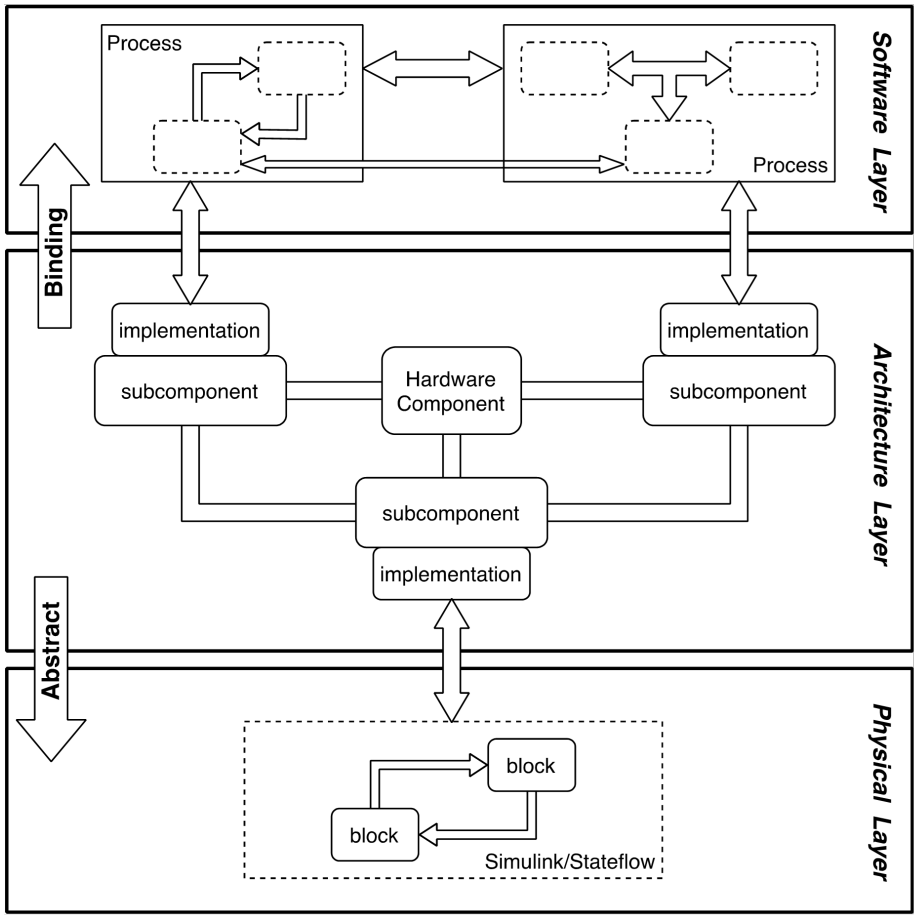
AADL $\oplus$ Simulink/Stateflow : A Combimed Graphical Modelling Language [TCS'21]

Why AADL ⊕ Simulink/Stateflow



AADL⊕Simulink/Stateflow	A unified graphical co-modelling formalism supporting the design of CPSs from all these three dimensions uniformly	**F+P+A**

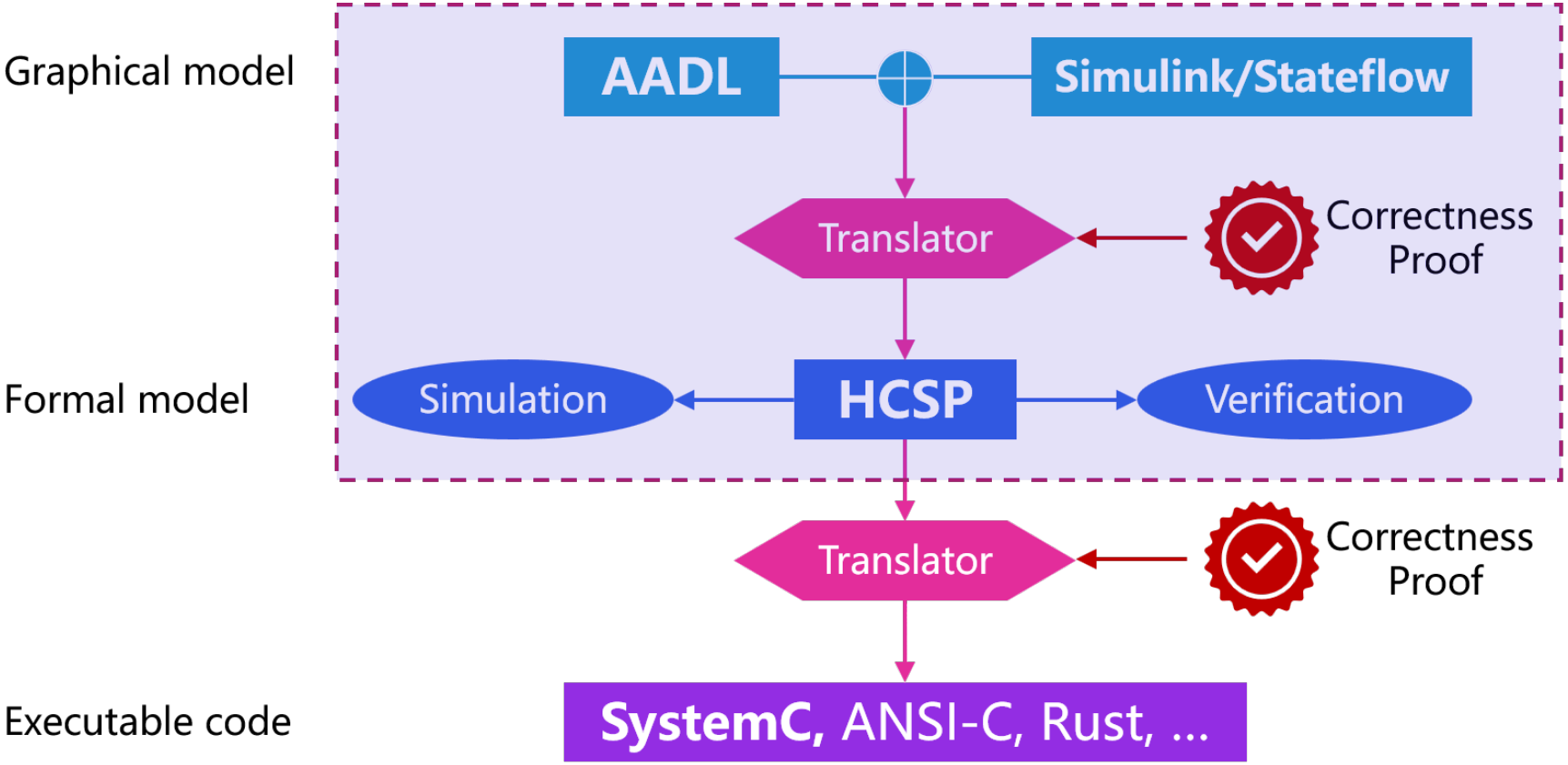
Overview of AADL ⊕ Simulink/Stateflow




```
thread PI_ctr
    features
        veh_v: in data port;
        des_v: in data port;
        des_a: out data port;
    properties
        dispatch_protocol => periodic;
        period => 7ms;
        compute_deadline => 7ms;
        compute_execution_time => 1ms;
        priority => 1;
end PI_ctr

thread implementation PI_ctr.imp
    annex Simulink{** ./PI_ctr_imp.xml **};
end PI_ctr.imp
```

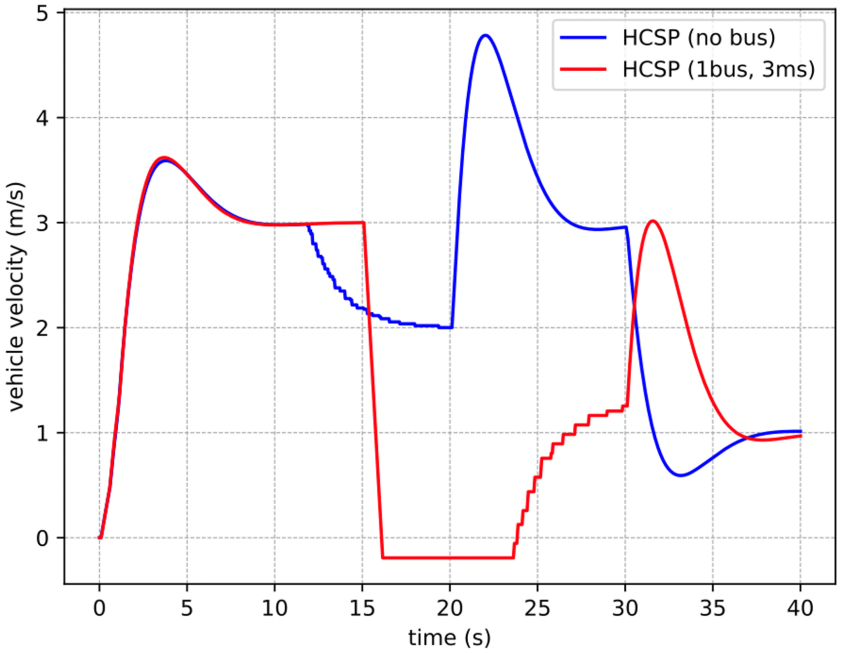
```
abstract vehicle
    features
        veh_a: in data port;
        veh_s: out data port;
        veh_l_v: out data port;
        veh_w_v: out data port;
end vehicle;

abstract implementation vehicle.imp
    annex Simulink{** ./vehicle_imp.xml **};
end vehicle.imp;
```

Model Transformation



Hybrid Communicating Sequential Processes (HCSP)

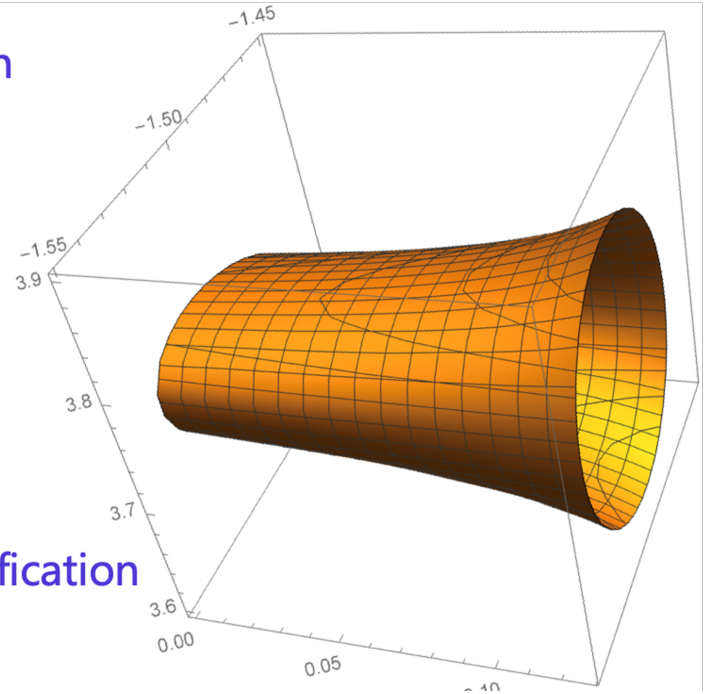
$$
\begin{aligned}
P &::= \mathsf{skip} \mid x := e \mid ch?x \mid ch!e \mid P; Q \mid B \rightarrow P \mid P \sqcap Q \mid P^* \mid [\!]_{i \in I}(io_i \longrightarrow Q_i) \mid \\
&\quad \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \mid \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(io_i \longrightarrow Q_i) \\
S &::= P_1 \| P_2 \| \ldots \| P_n \text{ for some } n \geq 1
\end{aligned}
$$



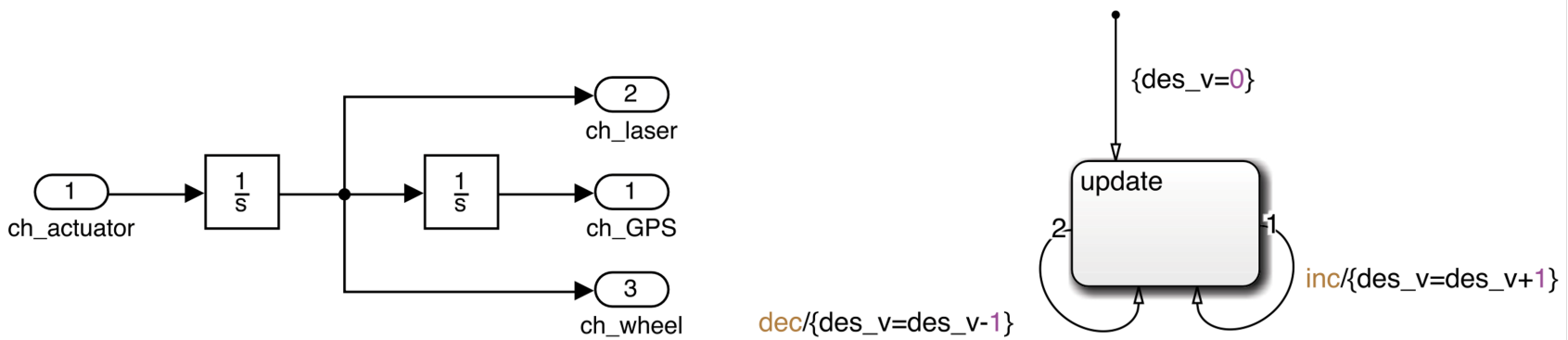
Simulation

&

Verification



From Simulink/Stateflow to HCSP [EMSOFT'13,ATVA'15]



(a) `vehicle.imp` (b) `pan_ctr_th.imp`

$$
\begin{array}{l}
v := 0; s := 0; \\
sent_{laser} := \textbf{false}; sent_{wheel} := \textbf{false}; sent_{GPS} := \textbf{false}; \\
\text{while } \neg(sent_{laser} \wedge sent_{wheel} \wedge sent_{GPS}) \\
\qquad [] \begin{pmatrix} ch_{laser}!v & \longrightarrow & sent_{laser} := \textbf{true} \\ ch_{wheel}!v & \longrightarrow & sent_{wheel} := \textbf{true} \\ ch_{GPS}!s & \longrightarrow & sent_{GPS} := \textbf{true} \end{pmatrix} \\
\text{endwhile}; ch_{acturator}?a; \\
\left(\langle \dot{s} = v, \dot{v} = a \& \textbf{true} \rangle \trianglerighteq [] \begin{pmatrix} ch_{laser}!v & \longrightarrow & \text{skip} \\ ch_{wheel}!v & \longrightarrow & \text{skip} \\ ch_{GPS}!s & \longrightarrow & \text{skip} \\ ch_{acturator}?a & \longrightarrow & \text{skip} \end{pmatrix} \right)^*
\end{array}
$$

$$
\begin{array}{l}
des_v := 0; \\
\left(\begin{array}{l} inputs?event; \\ event == \text{"inc"} \rightarrow des_v := des_v + 1; \\ event == \text{"dec"} \rightarrow des_v := des_v - 1; \\ outputs!des_v \end{array} \right)^*
\end{array}
$$

L. Zou, et al. **Verifying Simulink diagrams via a hybrid Hoare logic prover**. *EMSOFT* 2013, pp. 1-9.
N. Zhan, et al. **Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach**. *Springer*, 2017.

From AADL to HCSP (Scheduler and Dispatcher)

```
module SCHEDULE_HPF(sid) ::=
  Pool := [];
  run_now := -1; run_prior := -1
  (
  reqProcessor[sid][_tid]?prior ⟶
      if run_prior > prior then
          Pool := put(Pool, [prior, _tid])
      else
          run_now != -1 → preempt[sid][run_now]!;
          run_now := _tid;
          run_prior := prior;
          run[sid][run_now]!
      endif
  [] free[sid][_tid]? ⟶
      assert(_tid == run_now);
      if len(Pool) > 0 then
          (run_prior, run_now) := get_highest(Pool)
          Pool := delete(Pool, run_now);
          run[sid][run_now]!
      else
          run_prior := -1; run_now := -1
      endif
  [] exit[sid][_tid]? ⟶ Pool := delete(Pool, _tid)
  )*
```

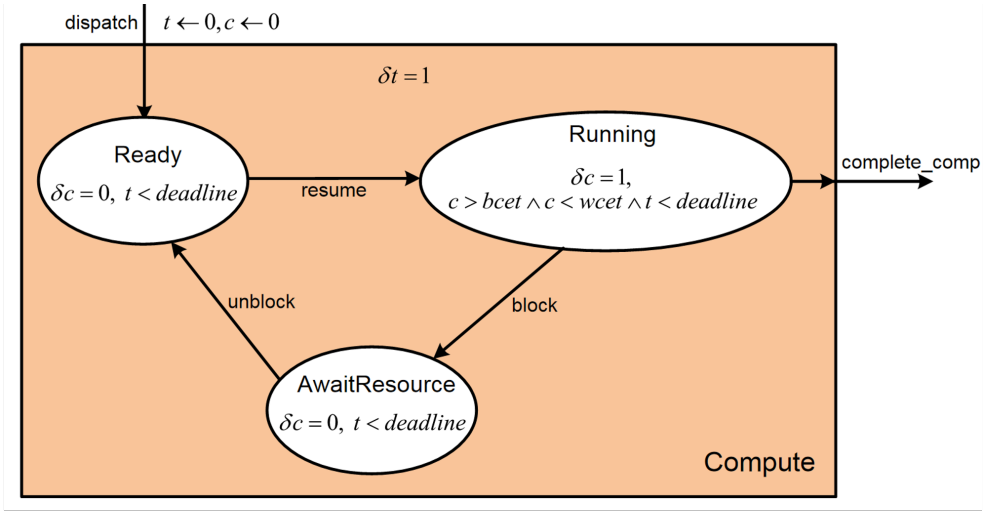
```
module DIS_aperiodic(send, out_port, recv, in_port) ::=
  queue := [];
  (
  if len(queue) == 0 then
      outputs[send][out_port]?event;
      queue := push(queue, event)
  else
      outputs[send][out_port]?event ⟶
          queue := push(queue, event)
      [] dis[recs][in_port]!head(queue) ⟶
          queue := tail(queue)
  endif
  )*
```

```
module DIS_periodic(tid, period) ::=
  (
  wait(period);
  dis[tid]!
  )*
```

- Processor ⇒ Scheduler (FIFO, RMS, DMS, HPF)
- Dispatchers are for threads

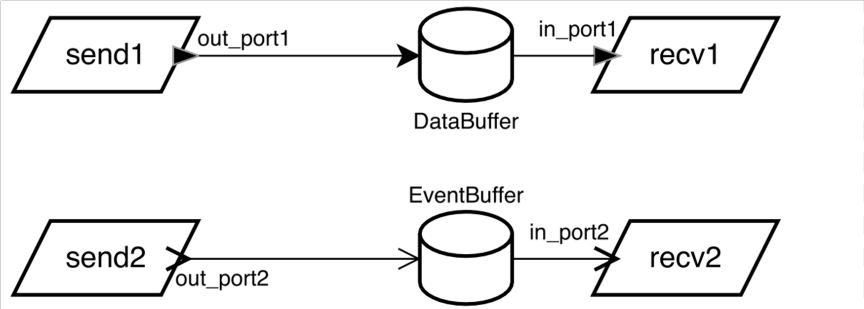
From AADL to HCSP (Thread)

```
module EXE(tid, prior, WCET, deadline) ::=
  @Initialisation;
  state := “wait”
  (
  if state == “wait” then
      dis[tid]?; @Input;
      t := 0; comp_complete := 0; state := “ready”
  elif state == “ready” then
      reqProcessor[tid]!prior;
      <t_dot = 1 & t < deadline>⊵
          [](run[tid]? ⟶ state := “running”);
      state == “ready” →
          exit[tid]! ⟶ state := “wait”
          []run[tid]? ⟶ state := “running”
  elif state == “running” then
      @RUNNING
  else  # state == “await”
      <t_dot = 1 & t < deadline>⊵
          [](unblock[tid]? ⟶ state := “ready”);
      state == “await” → state := “wait”
  endif
```

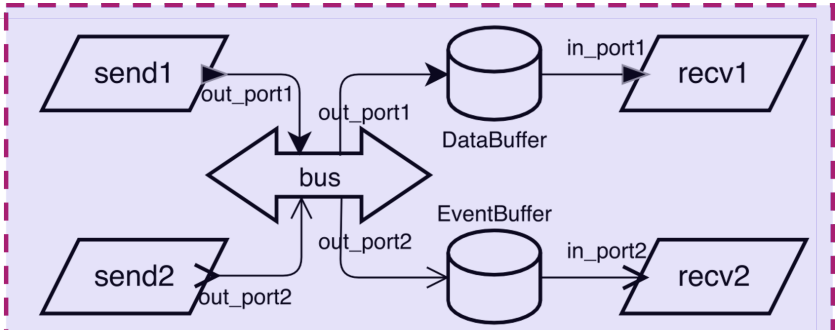



```
procedure RUNNING ::=
  if comp_complete == 0 then
      c := 0; @DiscreteCompute; comp_complete := 1
  else
      <t_dot = 1, c_dot = 1 & c < WCET ∧ t < deadline> ⊵
          []preempt[tid]? ⟶ state := “ready”
      state == “running” →
          if c == WCET then
              @Output
          else
              preempt[tid]? ⟶ state := “wait”
              []free[tid]! ⟶ state := “wait”
          endif
  endif
```

From AADL to HCSP (Connection)



(a) Buffers between ports



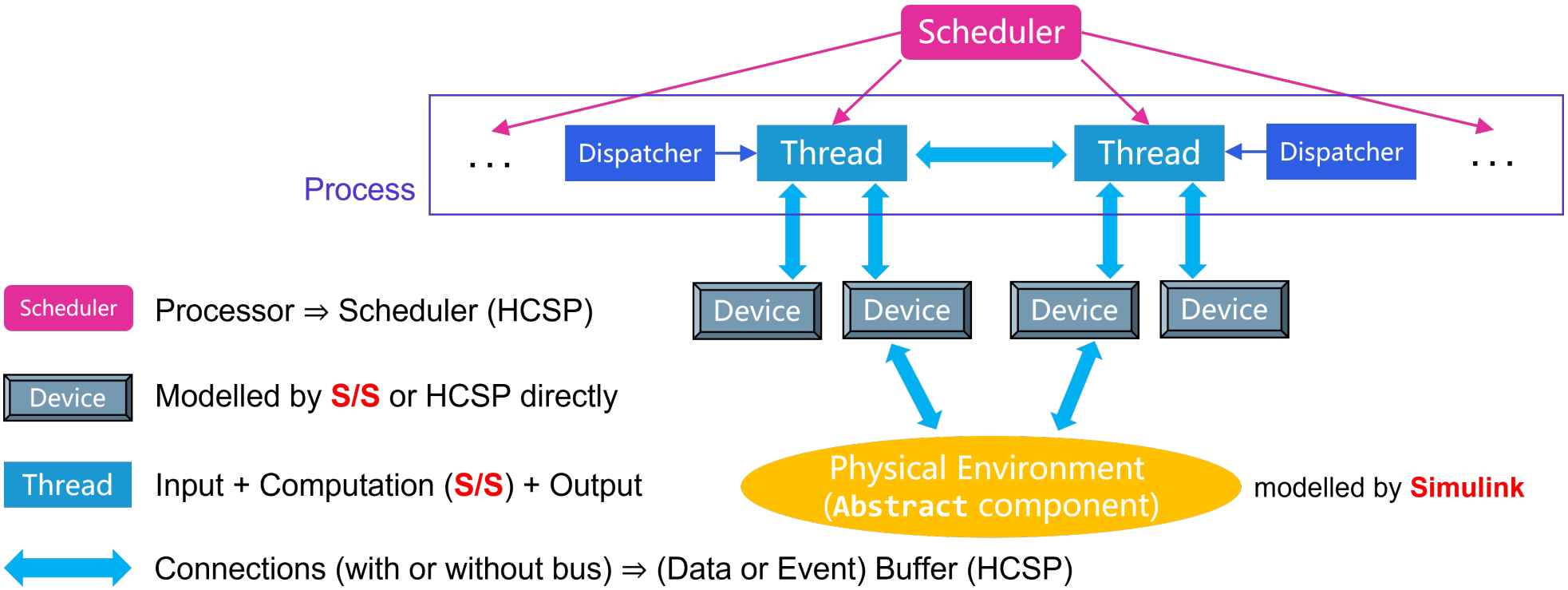
(b) Two sender threads share one bus

```
module BUS(bus_id, send1, out_port1, send2, out_port2) ::=
  (
  reqBus[send1]? ⟶
      outputs[send1][out_port1]?data;
      @BLOCK2;
      outputs[bus_id][out_port1]!data
  [] unblock[send1]! ⟶ skip
  [] reqBus[send2]? ⟶
      outputs[send2][out_port2]?event;
      @BLOCK1;
      outputs[bus_id][out_port2]!event
  [] unblock[send2]! ⟶ skip
  )*
```

```
procedure BLOCK1 ::=
  t := 0;
  while t < latency do
      <t_dot = 1 & t < latency> ⊵
          [] block[send1]! ⟶ skip
  endwhile
```

```
procedure BLOCK2 ::=
  t := 0;
  while t < latency do
      <t_dot = 1 & t < latency> ⊵
          [] block[send2]! ⟶ skip
  endwhile
```

From AADL ⊕ Simulink/Stateflow to HCSP

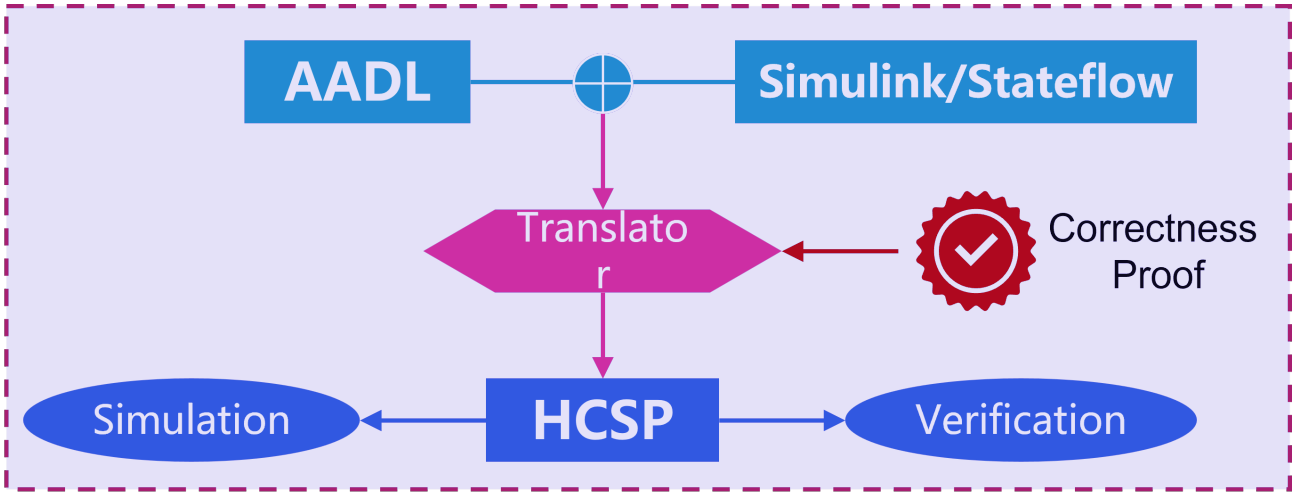


System ::= Scheduler || Dispatchers || Threads || Devices || Connections || Physical

A system is composed by the communication of its components

Implementation in **MARS**

- ✓ Co-modelling
  - AADL
  - Simulink/Stateflow
- ✓ Hybrid annex
  - Simulink/Stateflow
  - HCSP
- ✓ Scheduling analysis
  - HPF scheduler
- ✓ Latency analysis
  - Bus latency
- ✓ Safety analysis
  - Simulation
  - Verification
- ✓ Verification
  - Hybrid Hoare Logic



- ✗ Other AADL components and features, such as Memory
- ✗ Other scheduling policies, such as FIFO, RMS, DMS
- ✗ Verification for complex models
- ✗ …

Part 1 : Modeling and Simulation

Correctness from AADL $\oplus$ Simulink/Stateflow to HCSP by Higher-order UTP [ACM TOSEM'22]

Correctness According to Bisimulation

Definition of weak bisimulation

Let $T_i = \langle S_i, A_i, \rightarrow_i \rangle$ be two transition systems for $i = 1, 2$. A relation $R \subseteq S_1 \times S_2$ is a weak bisimulation, iff

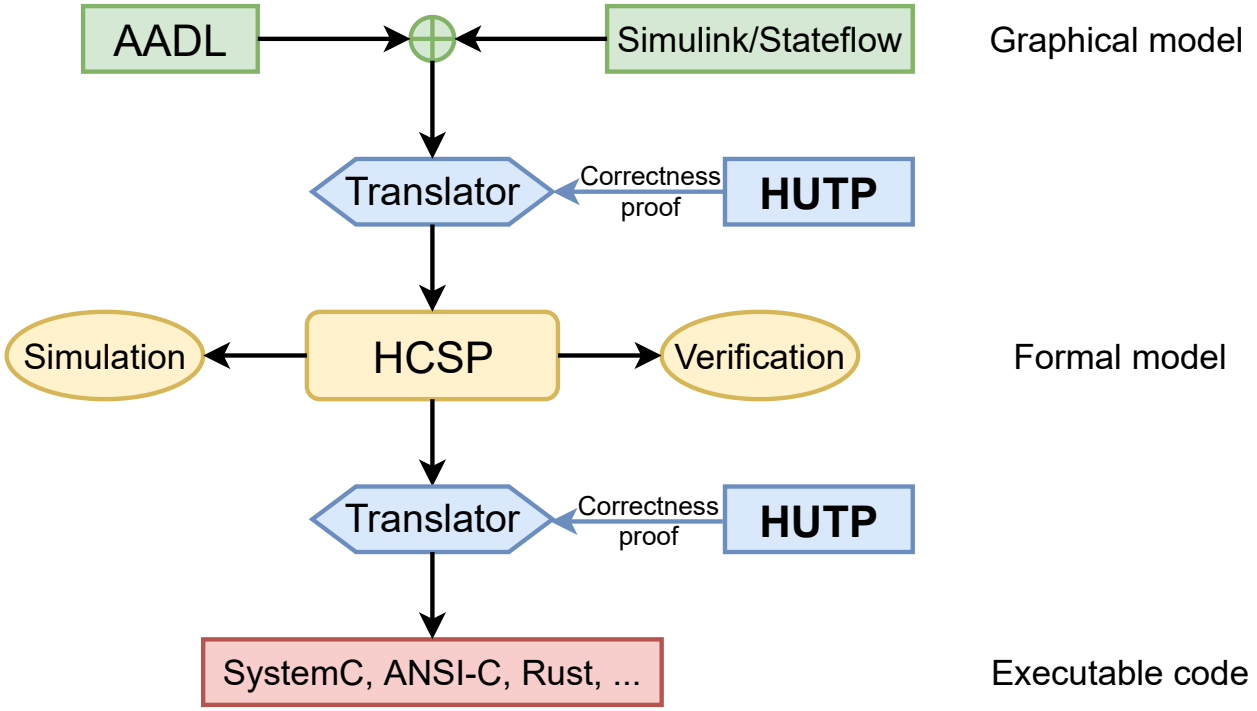
- R is symmetric
- $\forall (s, t) \in R$, s and t are equivalent under a variable mapping from T_1 to T_2
- if $s \Rightarrow_1^a s'$ holds, then there exists a path $t \Rightarrow_2^a t'$ such that $(s', t') \in R$

Correctness of translation

The translation from AADL$\oplus$S/S to HCSP is correct, i.e., there exists a weak bisimulation relation between AADL$\oplus$S/S and its translated HCSP.

Correctness of Model Transformation

MARS : A toolkit for Modelling, Analysis, and veRification of hybrid Systems

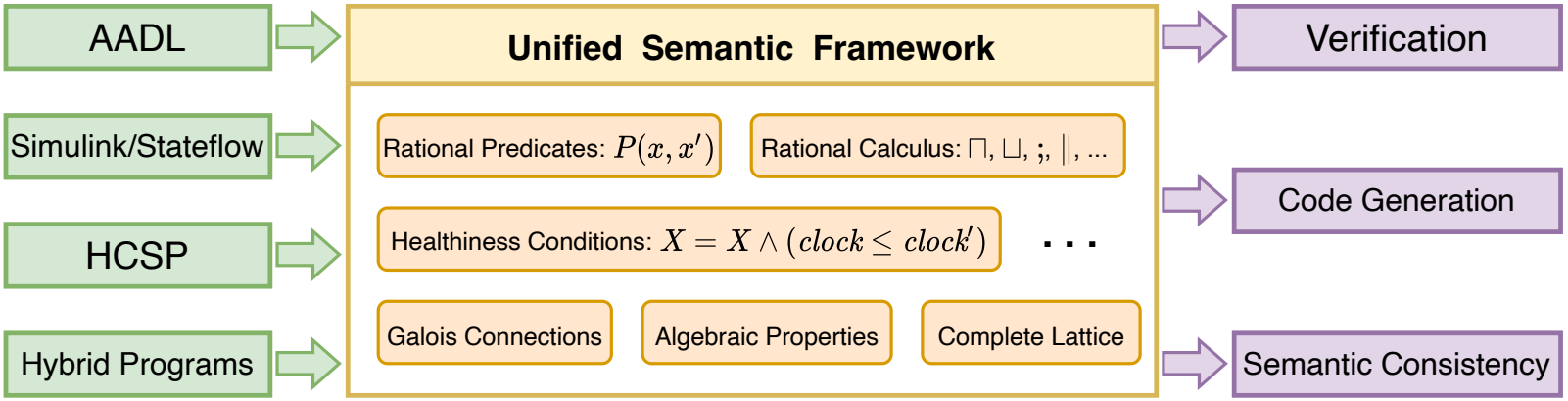


Question

How to guarantee the consistency among different paradigms at different levels of abstraction?

Unifying Theories of Programming (UTP)

- Proposed by Tony Hoare and Jifeng He in 1990s
- The idea of UTP is from the Grand Unifying Theory in physics
- It unifies different programming paradigms and models with first-order relational calculus
- Programs are predicates : alphabetised relational calculus
- Galois connections link different models and paradigms



Unifying Theories of Programming (UTP)

Programs as Predicates : (α, P)

- α consists of initial and final observations $\boldsymbol{x}$ and $\boldsymbol{x}'$, and P is a predicate on α
- Example : $x' = x + 1 \wedge y' = y$ denotes $x := x + 1$ given $\alpha = \{x, y, x', y'\}$

$$
\begin{array}{rcll}
x := e & \widehat{=} & x' = e \wedge \boldsymbol{x}' \backslash x' = \boldsymbol{x} \backslash x & \text{assignment} \\
P \fatsemi Q & \widehat{=} & \exists \boldsymbol{x}_0 \cdot P[\boldsymbol{x}_0/\boldsymbol{x}'] \wedge Q[\boldsymbol{x}_0/\boldsymbol{x}] & \text{sequence} \\
\text{if } b \text{ then } P \text{ else } Q & \widehat{=} & (b \wedge P) \vee (\neg b \wedge Q) & \text{conditional} \\
 & \neq & (b \Rightarrow P) \wedge (\neg b \Rightarrow Q) & \\
P \sqcap Q & \widehat{=} & P \vee Q & \text{(meet)} \\
P \sqcup Q & \widehat{=} & P \wedge Q & \text{(join)} \\
\text{local } x \cdot P & \widehat{=} & \exists x, x' \cdot P & \text{localisation} \\
\text{while } b \text{ do } P & \widehat{=} & \mu X. (\text{if } b \text{ then } P \fatsemi X \text{ else Skip}) & \text{loop}
\end{array}
$$

Refinement

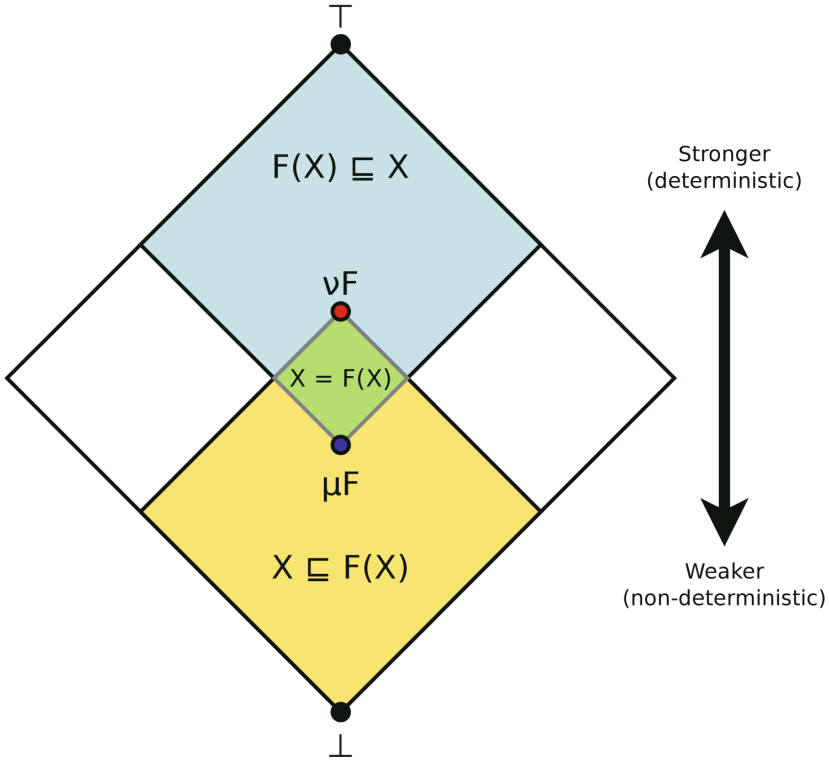
Let P and Q be the predicates with the same alphabet $\alpha = \{\boldsymbol{x}, \boldsymbol{x}'\}$. Then, P refines Q, denoted $Q \sqsubseteq P$, if $\forall \boldsymbol{x}, \boldsymbol{x}' \cdot P \Rightarrow Q$.

- $x' = x + 1 \sqsubseteq x := x + 1$ and $x' > x \wedge y' = y \sqsubseteq x := x + 1$ given $\alpha = \{x, y, x', y'\}$

Unifying Theories of Programming (UTP)

Complete Lattice

A set of predicates $\mathbb{P}$ forms a complete lattice if every subset of $\mathbb{P}$ has both a meet ($\sqcap$) and a join ($\sqcup$).



Fixed Points

- $\mu F \;\widehat{=}\; \bigsqcap\{X \mid F(X) \sqsubseteq X\}$
- $\nu F \;\widehat{=}\; \bigsqcup\{X \mid X \sqsubseteq F(X)\}$

Tarski's Theorem

Let $\mathbb{P}$ be a complete lattice and let $F : \mathbb{P} \to \mathbb{P}$ be a monotonic function, then the set of fixed points of F in $\mathbb{P}$ is also a complete lattice.

Unifying Theories of Programming (UTP)

Healthiness Conditions

- If a predicate P is a fixed point of $\mathcal{H}$, i.e., $P = \mathcal{H}(P)$, then it is $\mathcal{H}$-healthy.
- Idempotent : $\mathcal{H} \circ \mathcal{H} = \mathcal{H}$
- Monotonic : $P \sqsubseteq Q$ implies $\mathcal{H}(P) \sqsubseteq \mathcal{H}(Q)$

Tarski's Theorem

If $\mathbb{P}$ forms a complete lattice, so does $\{P \in \mathbb{P} \mid \mathcal{H}(P) = P\}$.

Example : a healthiness condition

Given $\alpha = \{x, x', ok, ok'\}$, where ok' denotes whether the program has terminated. If the preceding program is non-terminated ($\neg ok$), the current program P should do nothing but keep the values of all variables unchanged, expressed by the following healthiness condition :

$$\mathcal{H}(P) \quad = \quad P \lhd ok \rhd (x' = x \wedge ok' = ok)$$

HUTP as a Semantics Foundation for CPSs

Timed Trace Model

Synchronous Communication

- $ch?x$ receives a value along channel ch and assigns it to the variable *x*.
- $ch!e$ sends the value of the expression *e* of state variables along channel ch.

Trace Blocks

- $\langle ch*, d\rangle$ is a communication block, where $* \in \{?, !\}$ and *d* is a value along ch.
- $\langle \delta, RS\rangle$ is a wait block, where $0 < \delta \leq +\infty$ and RS is a set of waiting channel operations, called a **ready set**.
- τ denotes an internal action such as a timeless discrete computation.

Axioms for Canonical Timed Traces

- Associativity : $\forall \mathtt{tt}_0, \mathtt{tt}_1, \mathtt{tt}_2 \cdot (\mathtt{tt}_0 \frown \mathtt{tt}_1) \frown \mathtt{tt}_2 = \mathtt{tt}_0 \frown (\mathtt{tt}_1 \frown \mathtt{tt}_2)$
- Unit : $\forall \mathtt{tt} \cdot \epsilon \frown \mathtt{tt} = \mathtt{tt} \frown \epsilon = \mathtt{tt}$
- Left Zero : $\forall \mathtt{tt} \cdot \langle +\infty, RS\rangle \frown \mathtt{tt} = \langle +\infty, RS\rangle$
- Additivity : $\langle \delta_0, RS\rangle \frown \langle \delta_1, RS\rangle = \langle \delta_0 + \delta_1, RS\rangle$

HUTP as a Aemantics Foundation for CPSs

Timed Trace Model

Examples

- A divergence : $\langle \frac{1}{2}, \emptyset \rangle ^\frown \langle \frac{1}{4}, \emptyset \rangle ^\frown \cdots ^\frown \langle \frac{1}{2^n}, \emptyset \rangle ^\frown \cdots$
- A deadlock : $\tau ^\frown \langle sensor!, 4.5 \rangle ^\frown \langle +\infty, \{actuator?\} \rangle$

Assumptions for Parallel Composition

- Each pair of processes in parallel communicate over a set of common channels *I*
- Each $ch?x$ or $ch!e$ can only be owned by one sequential process
- Any pair of $ch?x$ and $ch!e$ cannot be owned by one sequential process

$$\mathtt{tt}_0 \|_I \mathtt{tt}_1 \quad \rightsquigarrow \quad \mathtt{tt}$$

Examples

- $\langle 2, \{ch?\} \rangle ^\frown \langle ch?, 1.3 \rangle \|_{\{ch\}} \langle 2, \{dh!\} \rangle ^\frown \langle ch!, 1.3 \rangle \quad \rightsquigarrow \quad \langle 2, \{ch?, dh!\} \rangle ^\frown \langle ch, 1.3 \rangle$
- $\langle dh?, \pi \rangle ^\frown \langle 2, \{dh?\} \rangle \|_{\{ch\}} \langle 1, \emptyset \rangle ^\frown \langle 1, \{ch?\} \rangle \rightsquigarrow \langle dh?, \pi \rangle ^\frown \langle 1, \{dh?\} \rangle ^\frown \langle 1, \{ch?, dh?\} \rangle$

HUTP as a Semantics Foundation for CPSs

Hybrid Process

Time+State+Trace

- $now, now' \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$ stand for the start- and end-**time**, respectively.
- $s, s' \in \mathbb{D}$ denote the respective **state** values at now and now', and $\underset{\sim}{s} : \mathbb{R}_{\geq 0} \to \mathbb{D}$ records its dynamic trajectory of **state**. Let s, $\underset{\sim}{\mathrm{s}}$, and s' denote the vectors of state.
- tr and tr' represent **canonical timed traces** at now and now', respectively.

$$\text{Hybrid Process} \quad ::= \quad P(now, now', \mathrm{s}, \underset{\sim}{\mathrm{s}}, \mathrm{s}', tr, tr')$$

Example : assignment

$$s := s + 1 \quad ::= \quad now = now' < +\infty \wedge s' = s + 1 \wedge \mathrm{s}' \backslash \{s'\} = \mathrm{s} \backslash \{s\} \wedge tr' = tr^\frown \tau$$

Example : deadlock

$$\begin{aligned} [\![\dot{\mathrm{s}} = \mathbf{0}, \{ch?\}]\!]_\infty \quad ::= \quad & now < now' = +\infty \wedge \mathrm{s} = \underset{\sim}{\mathrm{s}}(now) \wedge \\ & \forall t \in (now, +\infty). \dot{\underset{\sim}{\mathrm{s}}}(t) = \mathbf{0} \wedge tr' = tr^\frown \langle +\infty, \{ch?\} \rangle \end{aligned}$$

HUTP as a Semantics Foundation for CPSs

Hybrid Process

Ill Behaviours

$ti > ti'$ (time goes backwards), $\underset{\sim}{s}(t) = 1$ if $t \in \mathbb{Q}$ and 0 otherwise (Dirichlet function), ...

Healthiness conditions are used to characterise the model of computation

Healthiness Conditions for Hybrid Processes

A hybrid process is a fixed point of $X = \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4(X)$, where

$$
\begin{array}{lcl}
\mathcal{H}_0(X) & = & X \wedge now \leq now' \wedge tr \leq tr' \\
\mathcal{H}_1(X) & = & X \wedge now = \lim(tr) \wedge now' = \lim(tr') \\
\mathcal{H}_2(X) & = & (now = now' \wedge tr = tr') \lhd (now = +\infty \vee |tr| = +\infty) \rhd X \\
\mathcal{H}_3(X) & = & (\exists s'.X) \lhd (now' = +\infty \vee |tr'| = +\infty) \rhd X \\
\mathcal{H}_4(X) & = & X \wedge RC \wedge SD
\end{array}
$$

with

$$
\begin{array}{lcl}
RC & \widehat{=} & \forall k \cdot \forall t \in [now, now') \cdot \underset{\sim}{s}_k(t) = \underset{\sim}{s}_k(t^+) \\
SD & \widehat{=} & \forall k \cdot \forall t \in (now, now') \cdot \exists d_0, d_1 \cdot \dot{\underset{\sim}{s}}_k(t^-) = d_0 \wedge \dot{\underset{\sim}{s}}_k(t^+) = d_1
\end{array}
$$

and $\lim(tr)$ the time span of timed trace tr.

HUTP as a Semantics Foundation for CPSs

Hybrid Process

Meet ($\sqcap$) and Join ($\sqcup$)

$$\mathsf{HP}_0 \sqcap \mathsf{HP}_1 \;\widehat{=}\; \mathsf{HP}_0 \vee \mathsf{HP}_1 \quad \text{and} \quad \mathsf{HP}_0 \sqcup \mathsf{HP}_1 \;\widehat{=}\; \mathsf{HP}_0 \wedge \mathsf{HP}_1$$

Sequential Composition ($\fatsemi$)

$$\mathsf{HP}_0 \fatsemi \mathsf{HP}_1 \;\widehat{=}\; \exists now_*, s_*, tr_* \cdot \mathsf{HP}_0[now_*, s_*, tr_*/now', s', tr'] \wedge \mathsf{HP}_1[now_*, s_*, tr_*/now, s, tr]$$

Complement ($\neg_{\wr}$)

$$\neg_{\wr}\mathsf{HP} \;\widehat{=}\; \mathcal{H}(\neg\mathsf{HP}), \text{ where } \mathcal{H} = \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4$$

Higher-Order Quantification ($\exists$)

$$\exists \underset{\sim}{s} \cdot \mathsf{HP} \quad \widehat{=} \quad \bigvee \left\{ \mathsf{HP}[f, g/\underset{\sim}{s}, \dot{\underset{\sim}{s}}] \;\middle|\; \begin{array}{c} f : [now, now') \to \mathbb{D} \qquad g : (now, now') \rightharpoonup \mathbb{D} \\ \forall t \in (now, now') \cdot \dot{f}(t^-) = g(t^-) \wedge \dot{f}(t^+) = g(t^+) \end{array} \right\}$$

Parallel-by-Merge

Let s_0 and s_1 be state variables ($\mathrm{s}_0 \cap \mathrm{s}_1 = \emptyset$) of HP_0 and HP_1, respectively, then

$$\mathsf{HP}_0 \|_M \mathsf{HP}_1 \quad \widehat{=} \quad \mathcal{H}\left((\mathsf{HP}_0[now'_X, tr'_X/now', tr'] \wedge \mathsf{HP}_1[now'_Y, tr'_Y/now', tr'] \wedge tr = tr') \fatsemi M \right)$$

HUTP as a Semantics Foundation for CPSs

Hybrid Process

Refinement

Let $\alpha(\mathsf{HP}_0) = \alpha(\mathsf{HP}_1)$. Then, the refinement of HP_0 by HP_1 is defined by :

$$\mathsf{HP}_0 \sqsubseteq \mathsf{HP}_1 \quad \widehat{=} \quad \forall \mathit{now}, \mathit{now}', tr, tr', \mathbf{s}, \underset{\sim}{\mathbf{s}}, \mathbf{s}' \cdot \mathsf{HP}_1 \Rightarrow \mathsf{HP}_0.$$

We say that $\mathsf{HP}_0 \equiv \mathsf{HP}_1$ iff $\mathsf{HP}_0 \sqsubseteq \mathsf{HP}_1$ and $\mathsf{HP}_1 \sqsubseteq \mathsf{HP}_0$.

Complete Lattice

The set of hybrid processes $\mathbb{HP}$ forms a complete lattice with top and bottom :

$$\begin{array}{lclcll} \top_{\mathsf{HP}} & \widehat{=} & \bigsqcup \mathbb{HP} & = & \mathcal{H}(\mathbf{false}) & \text{Miracle} \\ \bot_{\mathsf{HP}} & \widehat{=} & \bigsqcap \mathbb{HP} & = & \mathcal{H}(\mathbf{true}) & \text{Chaos} \end{array}$$

where $\mathcal{H} = \mathcal{H}_0 \circ \mathcal{H}_1 \circ \mathcal{H}_2 \circ \mathcal{H}_3 \circ \mathcal{H}_4$.

Chaos

For any hybrid process HP, $\bot_{\mathsf{HP}} \fatsemi \mathsf{HP} \neq \bot_{\mathsf{HP}}$ and $\bot_{\mathsf{HP}} \| \mathsf{HP} \neq \bot_{\mathsf{HP}}$.

HUTP as a Semantics Foundation for CPSs

Normal Design

Auxiliary Variables

ok and ok' denote the process starts and terminates successfully, respectively.

$$P \vdash Q \quad \widehat{=} \quad (ok \wedge P) \Rightarrow (ok' \wedge Q)$$

Normal Design

- P and Q are relations denote pre- and post-conditions, respectively.
- $P \vdash Q$ is called **normal** if $\alpha(P) = \{\boldsymbol{x}\}$ and $\alpha(Q) = \{\boldsymbol{x}, \boldsymbol{x}'\}$.
- Normal designs can be treated as A/G **contracts**.
- $x \geq 0 \vdash x' = \sqrt{x}$ is a normal design but $x > x' \vdash x' = x - 1$ is not.

HUTP as a Semantics Foundation for CPSs

Normal Hybrid Design

Auxiliary Variables

- ok' (ok) : the current (preceding) process executes normally
- $\neg ok'$ ($\neg ok$) : the current (preceding) process gets into run-time errors

Design

$$Pre \vdash Post \quad \widehat{=} \quad (ok \wedge Pre) \Rightarrow (ok' \wedge Post)$$

Hybrid Condition

A hybrid process is a hybrid condition if it is a fixed point of $\neg_{\wr} X = \neg_{\wr} X \fatsemi \bot_{\mathsf{HP}}$.

Normal Hybrid Design

Let HC be a hybrid condition and HP a hybrid process. Then,

$$\mathsf{HC} \vdash \mathsf{HP} \quad \widehat{=} \quad (ok \wedge \mathsf{HC}) \Rightarrow_{\wr} (ok' \wedge \mathsf{HP})$$

is a normal hybrid design, where $X \Rightarrow_{\wr} Y \quad \widehat{=} \quad \neg_{\wr} X \vee Y$.

HUTP as a Semantics Foundation for CPSs

Normal Hybrid Design

Theorem for Basic Operations

- $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcap (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \mathsf{HC}_1) \vdash (\mathsf{HP}_0 \vee \mathsf{HP}_1)$
- $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqcup (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \vee \mathsf{HC}_1) \vdash ((\mathsf{HC}_0 \Rightarrow_\wr \mathsf{HP}_0) \wedge (\mathsf{HC}_1 \Rightarrow_\wr \mathsf{HP}_1))$
- $(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \fatsemi (\mathsf{HC}_1 \vdash \mathsf{HP}_1) = (\mathsf{HC}_0 \wedge \neg_\wr(\mathsf{HP}_0 \fatsemi \neg_\wr \mathsf{HC}_1)) \vdash (\mathsf{HP}_0 \fatsemi \mathsf{HP}_1)$

Theorem for Parallel Composition

Let $\mathsf{NHD}(M) \mathrel{\widehat{=}} (ok_X \wedge ok_Y) \Rightarrow_\wr (M \wedge ok')$, then

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \|_{\mathsf{NHD}(M)} (\mathsf{HC}_1 \vdash \mathsf{HP}_1) \quad = \quad \begin{pmatrix} \neg_\wr(\neg_\wr \mathsf{HC}_0 \|_{\bot_{\mathsf{HP}}} \neg_\wr \mathsf{HC}_1) \\ \wedge \neg_\wr(\neg_\wr \mathsf{HC}_0 \|_{\bot_{\mathsf{HP}}} \mathsf{HP}_1) \\ \wedge \neg_\wr(\neg_\wr \mathsf{HC}_1 \|_{\bot_{\mathsf{HP}}} \mathsf{HP}_0) \end{pmatrix} \vdash (\mathsf{HP}_0 \|_M \mathsf{HP}_1)$$

Definition for Parallel Composition

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \|_M^{\mathsf{HP}} (\mathsf{HC}_1 \vdash \mathsf{HP}_1) \quad \mathrel{\widehat{=}} \quad (\mathsf{HC}_0 \wedge \mathsf{HC}_1) \vdash (\mathsf{HP}_0 \|_M \mathsf{HP}_1)$$

HUTP as a Semantics Foundation for CPSs

Normal Hybrid Design

Refinement

$$(\mathsf{HC}_0 \vdash \mathsf{HP}_0) \sqsubseteq (\mathsf{HC}_1 \vdash \mathsf{HP}_1) \text{ iff } \mathsf{HC}_1 \sqsubseteq \mathsf{HC}_0 \text{ and } \mathsf{HP}_0 \sqsubseteq (\mathsf{HC}_0 \wedge \mathsf{HP}_1)$$

Complete Lattice

Normal hybrid designs form a complete lattice with top and bottom

$$\begin{array}{lcll} \top_{\mathsf{NHD}} & \widehat{=} & \bot_{\mathsf{HP}} \vdash \top_{\mathsf{HP}} & \text{Miracle} \\ \bot_{\mathsf{NHD}} & \widehat{=} & \top_{\mathsf{HP}} \vdash \bot_{\mathsf{HP}} & \text{Chaos} \end{array}$$

Chaos

For any normal hybrid design NHD. Then,

$$\bot_{\mathsf{NHD}} \fatsemi \mathsf{NHD} \quad = \quad \bot_{\mathsf{NHD}} \| \mathsf{NHD}_{(M)} \mathsf{NHD} \quad = \quad \bot_{\mathsf{NHD}} \|^{\mathsf{HP}}_{M} \mathsf{NHD} \quad = \quad \bot_{\mathsf{NHD}}$$

Remember that $\bot_{\mathsf{HP}} \fatsemi \mathsf{HP} \neq \bot_{\mathsf{HP}}$ and $\bot_{\mathsf{HP}} \| \mathsf{HP} \neq \bot_{\mathsf{HP}}$.

Correctness of Model Transformation in **MARS**

From AADL $\oplus$ Simulink/Stateflow to HCSP

Prove the correctness of translation using HUTP

Graphical	Simulink diagram D	$\Longrightarrow$	$[\![\mathsf{D}]\!]_{\mathsf{HUTP}}$
	$\Downarrow$ MARS		$\Vert\!\vert$
Formal	HCSP program $[\![\mathsf{D}]\!]_{\mathsf{HCSP}}$	$\Longrightarrow$	$[\![[\![\mathsf{D}]\!]_{\mathsf{HCSP}}]\!]_{\mathsf{HUTP}}$
	$\Downarrow$ MARS		$\Vert\sqcap$
Executable	Code $[\![[\![\mathsf{D}]\!]_{\mathsf{HCSP}}]\!]_{\mathsf{CODE}}$ (SystemC, ANSI-C, Rust, ...)	$\Longrightarrow$	$[\![[\![[\![\mathsf{D}]\!]_{\mathsf{HCSP}}]\!]_{\mathsf{CODE}}]\!]_{\mathsf{HUTP}}$ (Not yet finished)

[1] L. Zou et al. **Verifying Simulink diagrams via a hybrid Hoare logic prover**. EMSOFT'13.

[2] N. Zhan et al. **Formal Verification of S/S Diagrams : A Deductive Approach**. Springer, 2017.

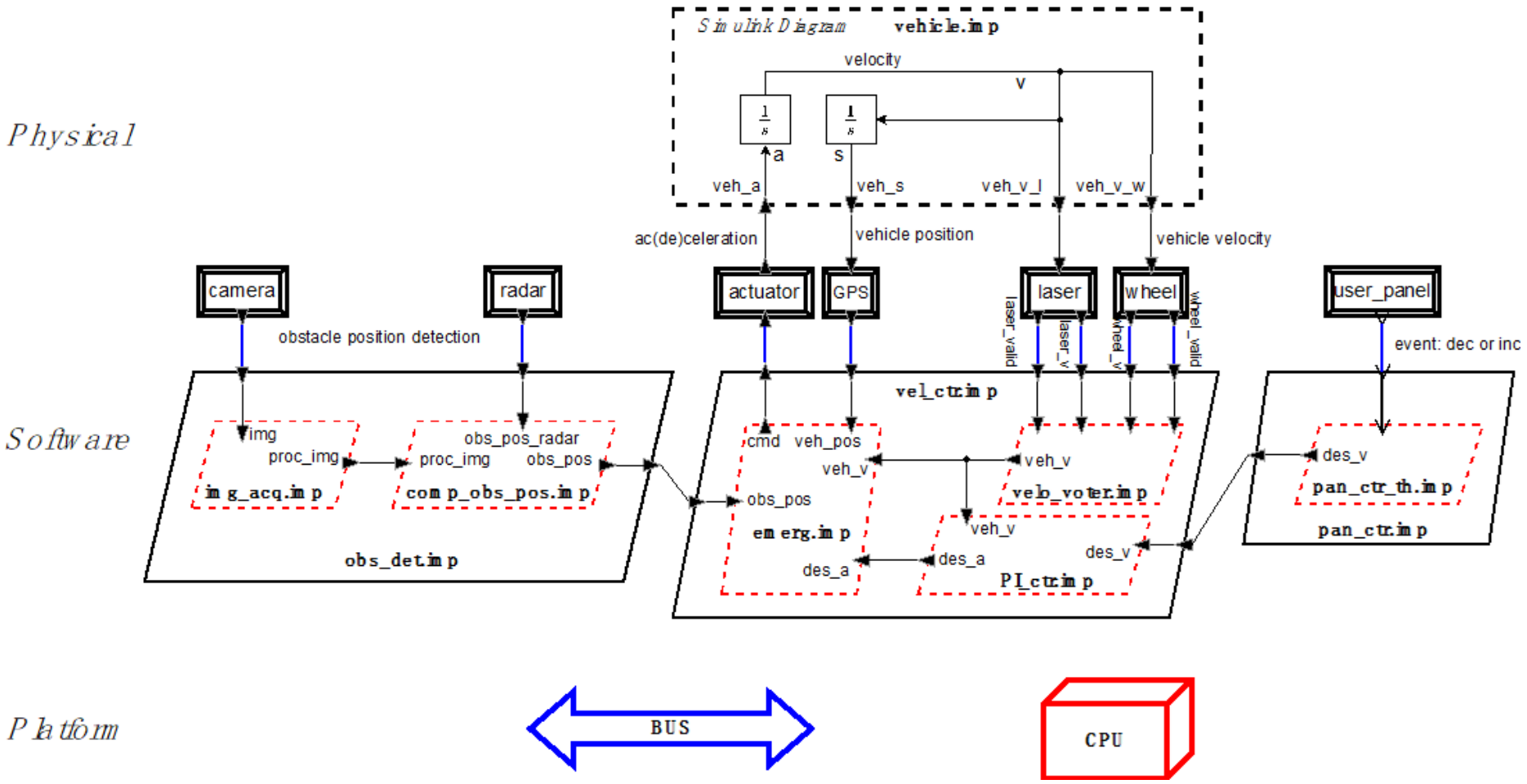
[3] G. Yan et al. **Automatically generating SystemC code from HCSP formal mdoels**. ACM TOSEM 29(1).

[4] S. Wang et al. **Formally Verified C Code Generation from Hybrid Communicating Sequential Processes**. ICCPS'24.

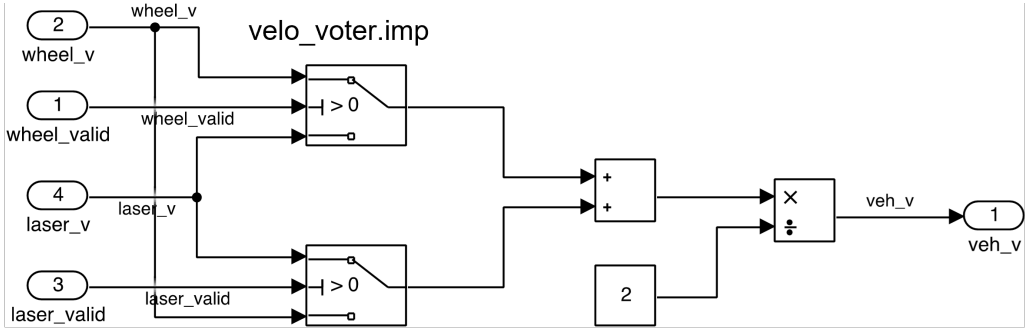
Part 1 : Modeling and Simulation

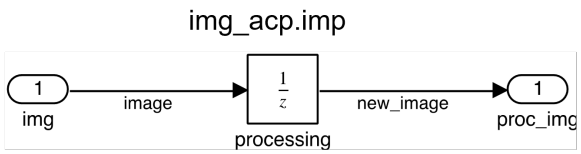
Case Study : A Cruise Control System [TCS'22, Jones' Festchrift]

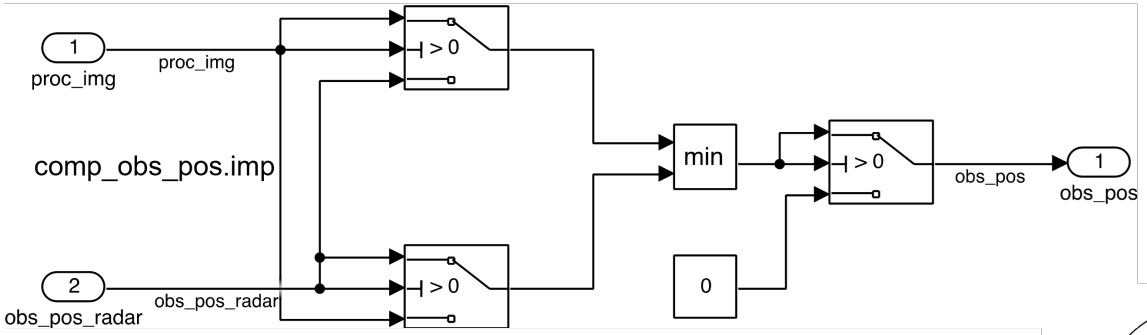
Architecture of CCS

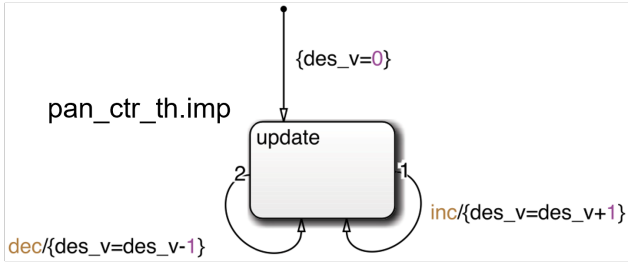


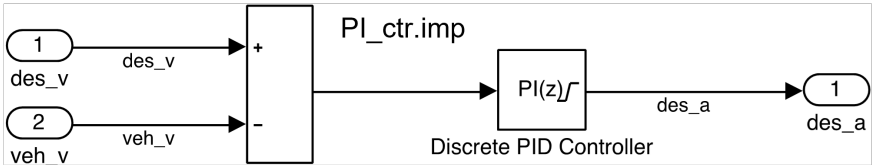
Details of CCS

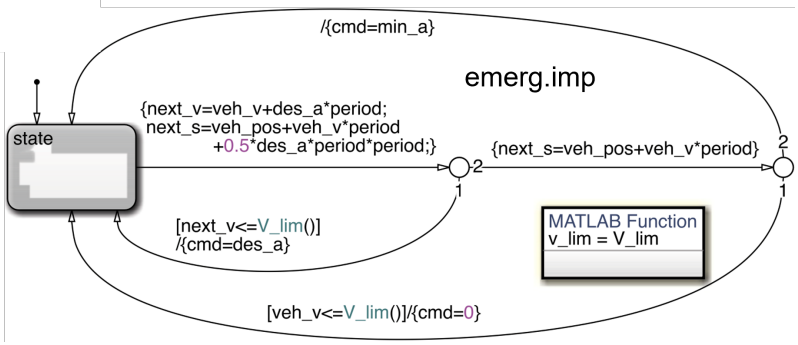












Verification of CCS

Abstracted HCSP processes

$$
\begin{array}{rcl}
\text{System} & ::= & \text{Plant} \| \text{Control} \\
\text{Plant} & ::= & v := v_0; s := s_0; P2C!v; P2C!s; C2P?a; (\langle \dot{s} = v, \dot{v} = a \& \mathbf{true} \rangle \trianglerighteq [](P2C!v \longrightarrow (P2C!s; C2P?a)))^* \\
\text{Control} & ::= & \left(P2C?v; P2C?s; v' := v + a_{des} \cdot period; s' := s + v \cdot period + \frac{1}{2} \cdot a_{des} \cdot period^2; C2P!a(s,v); \text{wait}(period)\right)^*
\end{array}
$$

$$
v_{lim}(s) = \begin{cases} v_{max}, & \text{if } s_{obs} - s \geq \frac{v_{max}^2}{-2a_{min}} \\ \sqrt{-2a_{min} \cdot (s_{obs} - s)}, & \text{if } 0 < s_{obs} - s < \frac{v_{max}^2}{-2a_{min}} \\ 0, & \text{otherwise} \end{cases}
$$

$$
a(s, v) = \begin{cases} a_{des} & \text{if } v_{next} \leq v_{lim}(s_{next}) \\ 0 & \text{else if } v \leq v_{lim}(s + v \cdot period) \\ a_{min} & \text{otherwise} \end{cases}
$$

The safety property can be implied by the loop invariant $\boxed{loop_inv: s \leq s_{obs} \wedge v \leq v_{lim}(s)}$

We can prove that $\boxed{loop_inv(s,v) \rightarrow loop_inv(s',v')}$ where $\begin{cases} v' = v + a(s,v) \cdot period \\ s' = s + v \cdot period + 0.5 \cdot a(s,v) \cdot period^2 \end{cases}$

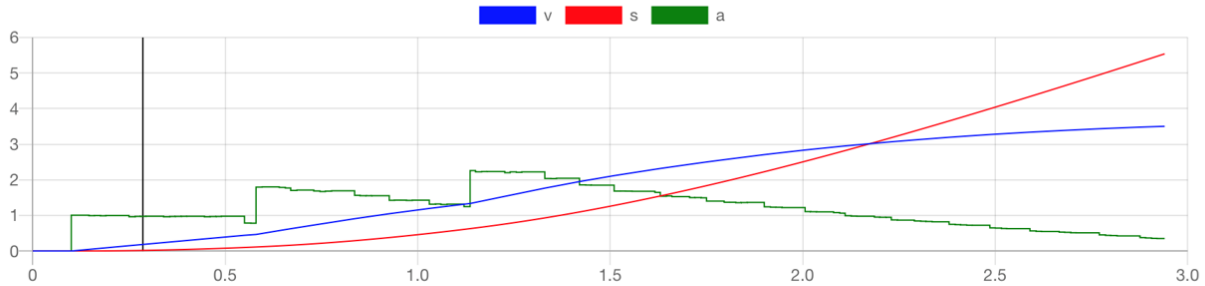
Simulation of Translated HCSP Process

HCSP Simulator | Read HCSP File | system.txt | Number of steps: 100000 | Showing 10000 | starting from 0

10000+ events. Cur step | Show events only | Show details

Process: vehicle_imp Show Process Show callstack

```
v:  0.185
s:  0.017
PHY_vehicle_imp_veh_a:  0
PHY_vehicle_imp_veh_s:  1
PHY_vehicle_imp_veh_v_l:  1
PHY_vehicle_imp_veh_v_w:  1
a:  0.974   Hide graph
```



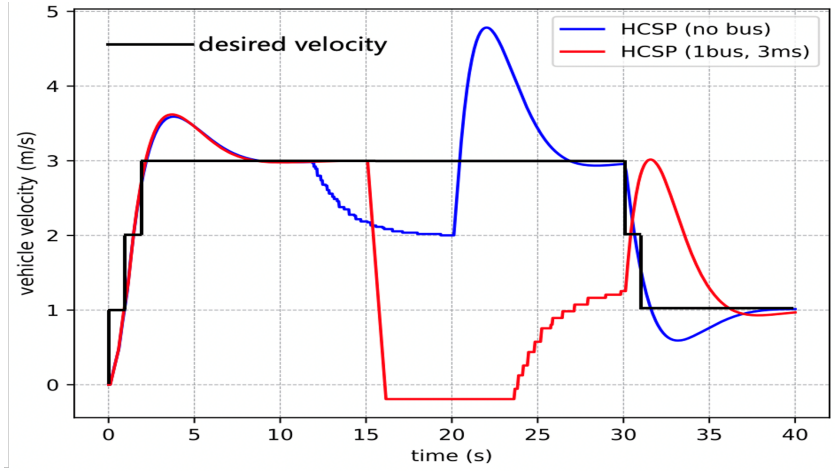
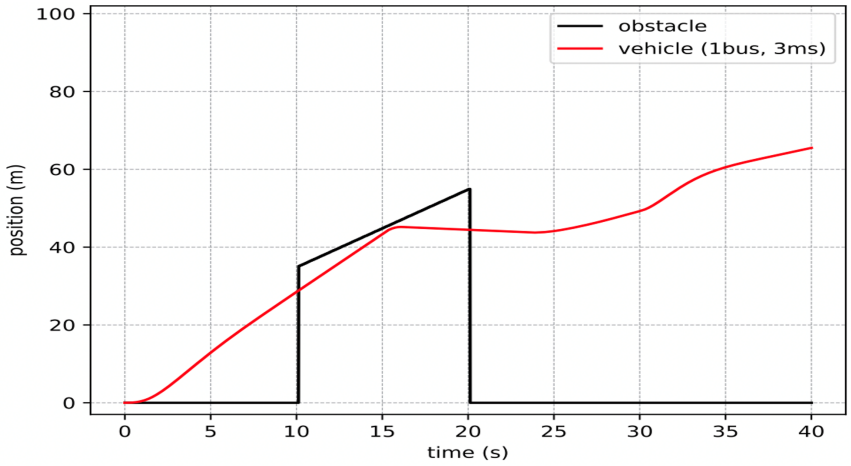
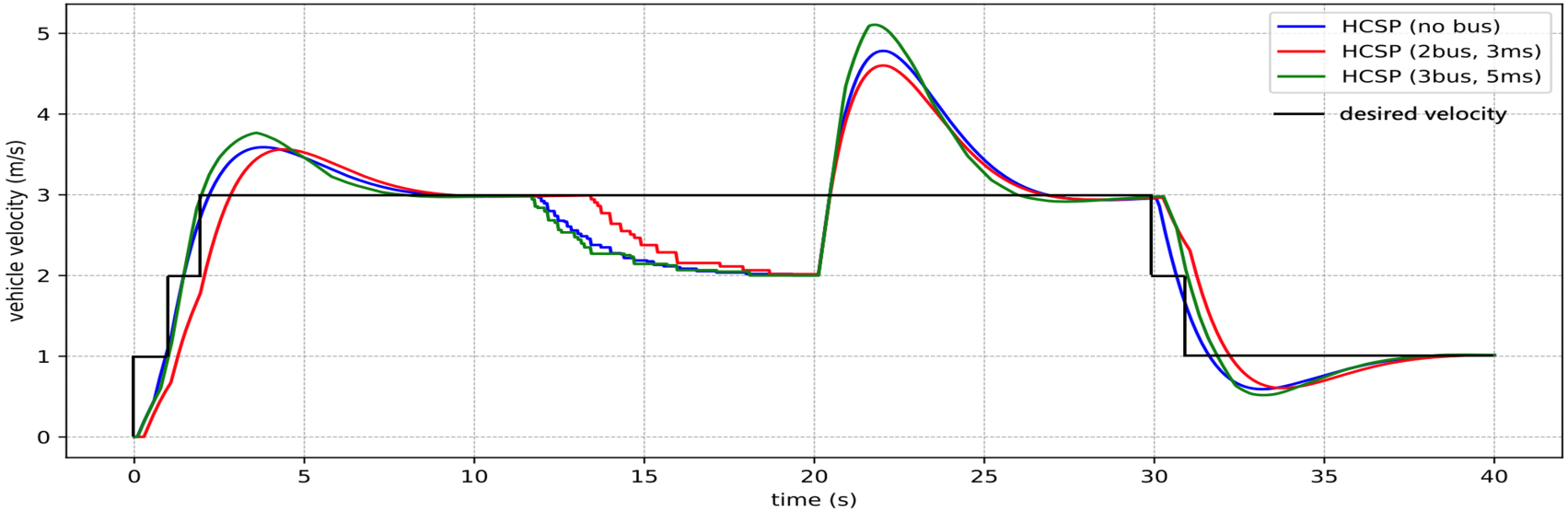
Process: ACT_img_acq_imp Hide Process Show callstack

```
(
  dis["img_acq_imp"]!;
  wait(0.045)
)**
  Show graph
```

Process: img_acq_imp Hide Process Show callstack

```
9856 step
9857 delay 0.001
9858 step
9859 step
9860 IO inputs[actuator][cmd] 0.974
9861 step
9862 IO outputs[PHY_vehicle_imp][veh_a] 0.974
9863 step
9864 IO block[emerg_imp]
9865 step
9866 IO free[0][emerg_imp]
9867 step
9868 step
```

scheduler: Show Process Show callstack
vehicle_imp: Show Process Show callstack
ACT_img_acq_imp: Hide Process Show callstack
img_acq_imp: Hide Process Show callstack
ACT_comp_obs_pos_imp: Show Process Show callstack
comp_obs_pos_imp: Show Process Show callstack
ACT_PI_ctr_imp: Show Process Show callstack
PI_ctr_imp: Hide Process Show callstack
ACT_emerg_imp: Hide Process Show callstack

Callstacks:
vehicle_imp:line18: skip,

Simulation of CCS