Formally verifying properties of Cyber-physical Systems (CPS) developed in Hybrid Input Output Automata (HIOA)

Partha Roop, Avinash Malik, Sidharta Andalam

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

BioRemulation™ Research Group
The University of Auckland

December 2016

www.pretzel.ece.auckland.ac.nz/bio

Outline

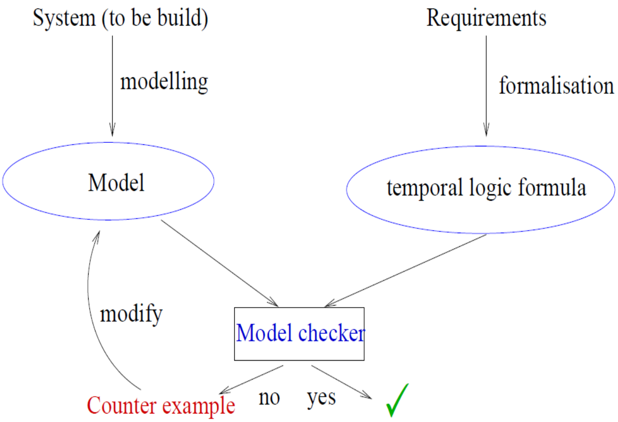
Learning outcomes

1. Formal property verification of Cyber-physical Systems (CPS) designed as Hybrid Input Output Automata (HIOA).

Current techniques for validation of CPS

1. Social (not formal)
   - Code reviews
   - Extreme/Pair programming
2. Methodological (less formal)
   - Design patterns
   - Test-driven development
3. Formal methods (robust)
   - Sound type systems, e.g., Coq
   - Mathematical Logical frameworks
   - Formal verification frameworks, e.g., Model checking

The general approach to model checking



Requirements and assumptions in model checking

Requirements

- *Model* specification should be precise to avoid any ambiguities in the result:
  - Formal language – We will use a language called *Promela*
  - Formal precise semantics
- *Temporal Properties* should be precise to avoid any ambiguities:
  - Formal language – we will use Linear Temporal Logic (LTL)
  - Formal precise semantics

Assumption

The model checker is bug free

Current state of the art in verification of CPS modeled as HIOA

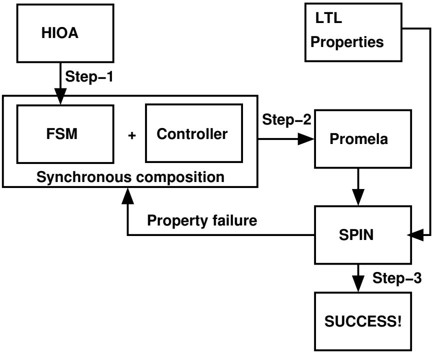
SpaceEx [1]

1. Designed for *safety* property (something bad will **never** happen) verification of CPS designed as Hybrid Automata (HA)
2. Converts trajectories of HA into over-approximated polyhedron.
3. Cannot do *liveness* (something **good** will eventually happen) property verification
4. Is *not* scalable for large systems
5. Reachability is undecidable, only best effort.

Dreach [2]

1. Designed for *safety* property verification of CPS designed as HA
2. Converts HA trajectories into Satisfiability Modulo Theory (SMT) formulas to be verified by SMT solvers
3. Cannot do *liveness* property verification

Proposed Solution



Steps

- *Step-1*: Compile plant HIOA specification to Finite State Machine (FSM)
- *Step-2*: Compile the synchronous composition of the plant and controller FSMs into Promela
- *Step-3*: Model check the LTL properties on the Promela model.
- Modify HIOA or controller if property does **not** verify and repeat above steps, else success!

Advantages

- Safety property verification
- Liveness property verification
- Scalability

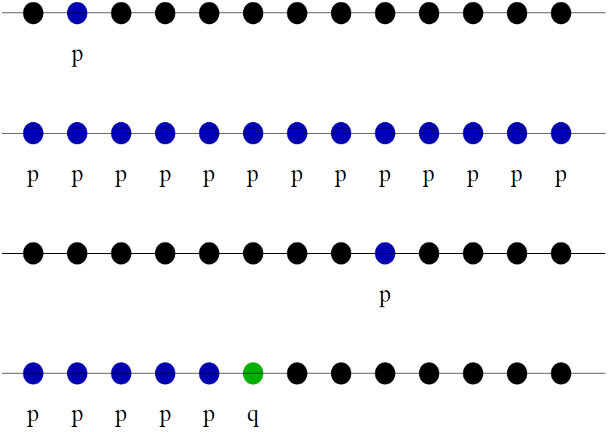
LTL formulas

- Are a subset of CTL*
- Are distinct from CTL
- Contain a **single** universal quantifier, i.e., they should hold for all traces of the system.

Def.. *Let* $\pi = s_0 s_1 s_2 \ldots$ *a path,* φ *an LTL formula.* $\pi \models \varphi$ *is inductively defined as follows.*

- $\pi \models p, p \in AP$ *iff* p *holds in* s_0 *(i.e.* $p \in L(s_0)$*)*
- $\pi \models \neg\varphi$ *iff not* $\pi \models \varphi$
- $\pi \models \varphi \vee \psi$ *iff* $\pi \models \varphi$ *oder* $\pi \models \psi$
- $\pi \models X\,\varphi$ *iff* $\pi^1 \models \varphi$
- $\pi \models G\,\varphi$ *iff* $\forall i \geq 0 : \pi^i \models \varphi$
- $\pi \models F\,\varphi$ *iff* $\exists j \geq 0 : \pi^j \models \varphi$
- $\pi \models \varphi\, U\, \psi$ *iff* $\exists k \geq 0 : \pi^k \models \psi$ *and* $\forall j, 0 \leq j < k, \pi^j \models \varphi$.

Formal LTL syntax and semantics



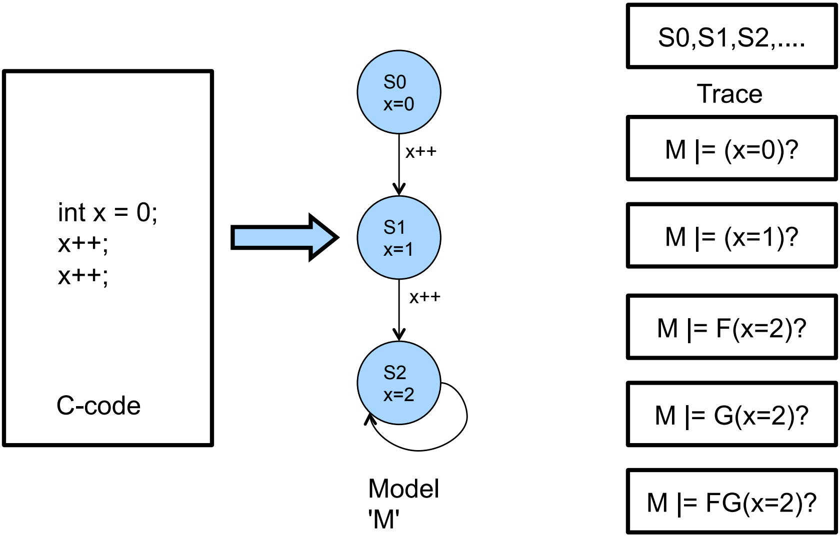
$\models Xp$

$\models G\,p$

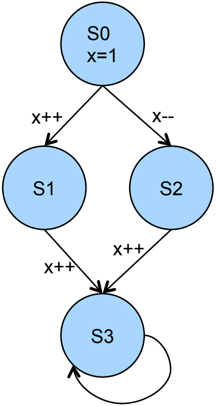
$\models F\,p$

$\models p\,U\,q$

Example 1 of LTL



Example 2 of LTL



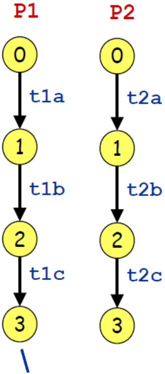
M |= F(x=3)?

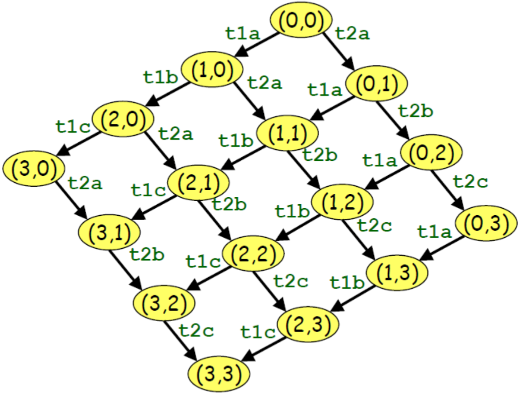
- SPIN = Simple Promela Interpreter
- SPIN model-checking LTL properties on models of concurrent software program.
- The *model* of the concurrent software program is modeled in a language called "Promela", which is very similar to "C".
- SPIN is used by NASA for designing mars rover software.
- Used by Airbus and Toyota for their breaking system.
- It is industrial strength model-checker developed by Bell Labs.

Example of Promela

```
proctype P1() { t1a; t1b; t1c }
proctype P2() { t2a; t2b; t2c }
init { run P1(); run P2() }
```

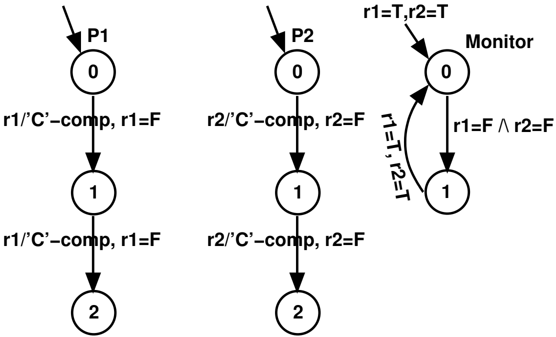
No atomicity



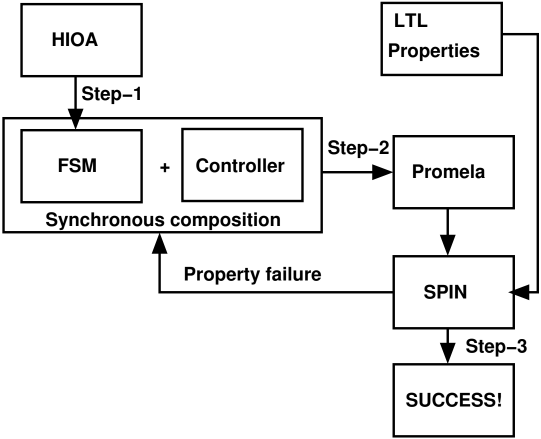


1. Build a *synchronous* product on top of asynchronous product
2. Promela natively only allows `int` types and smaller, we need to support floating point types (but what about precision?)

Solutions for the challenges



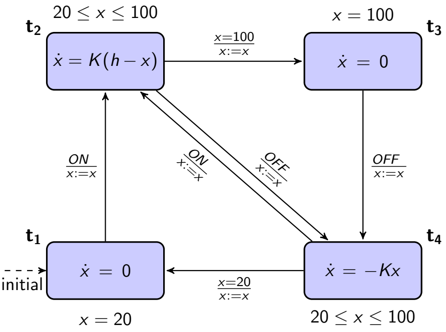
Recall our approach

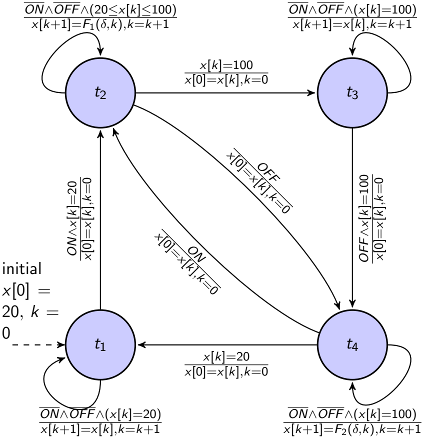


Step-1: Compiling HIOA to a FSM

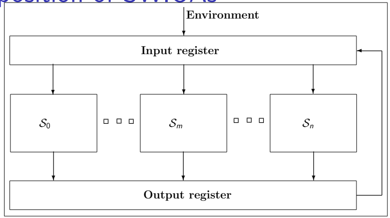
HIOA to Synchronous Witness Input Output Automata (SWIOA)

Recall from last lecture

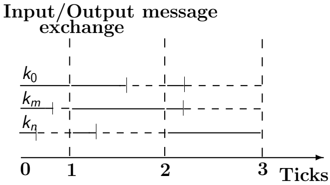




Composition of SWIOAs



(a) Modular execution of SWIOAs



(b) SWIOA execution trace

Remarks

1. Each SWIOA reads the inputs from the environment, e.g., pacemaker in case of the heart model, or other SWIOA's outputs via a shared register (or memory) called the input register.
2. Each SWIOA performs a local tick.
3. Upon completion of its local tick, each SWIOA produces outputs to a shared register (or memory).
4. Each SWIOA waits until **all** other SWIOAs have completed their individual local ticks, i.e., they *barrier synchronize*.
5. Once every SWIOA has completed a local tick, the produced outputs from the individual SWIOAs is transfered into the input register, the environment inputs are read into the input register and these steps are repeated – same as the monitor we saw previously!

Verifying properties of the water tank heating system

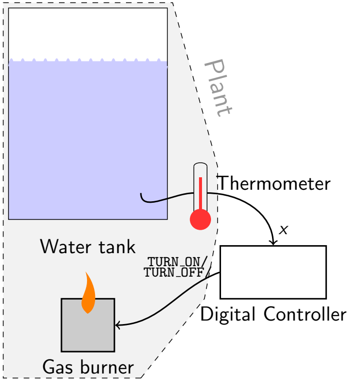
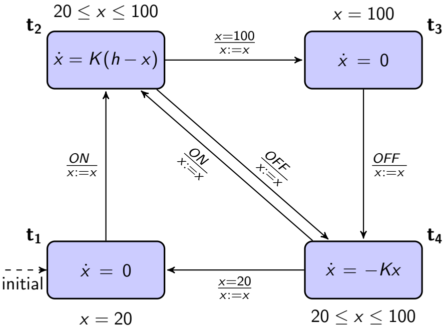
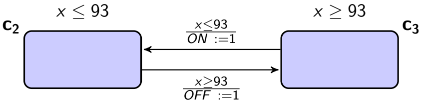


Figure: Water tank system overview

Verifying properties of the water tank heating system



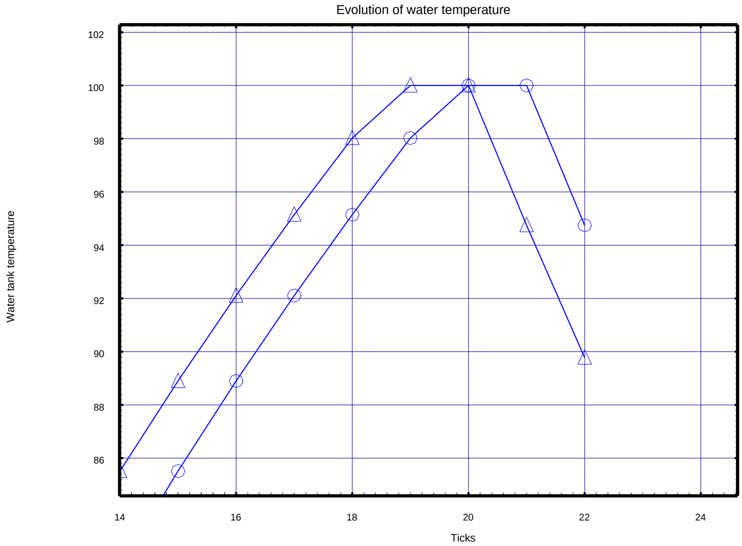


1. The Haskell code for the `WaterTankController` system
2. What properties we want to verify
   - We want to make sure that the water temperature never exceeds 100^o C $\Rightarrow$ **safety property**.
3. Safety property verification for increasing δ: $\delta = 0.5$ and $\delta = 0.7$.

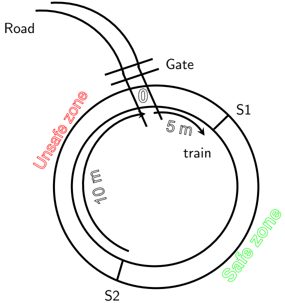
1. The Haskell code for the `WaterTankController` system
2. What properties we want to verify
   - We want to make sure that the water temperature never exceeds 100^o C $\Rightarrow$ **safety property**.
3. Safety property verification for increasing δ: $\delta = 0.5$ and $\delta = 0.7$.
   - At what temperature should we switch off the burner?
   - For $\delta = 0.5$, 93 is OK.

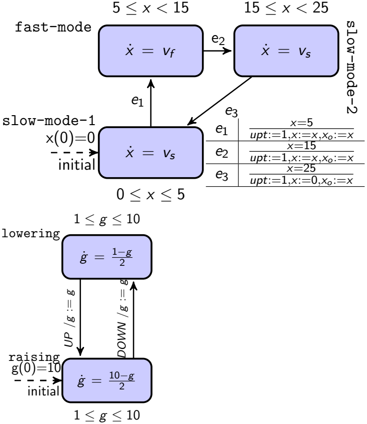
1. The Haskell code for the `WaterTankController` system
2. What properties we want to verify
   - We want to make sure that the water temperature never exceeds 100^o C $\Rightarrow$ **safety property**.
3. Safety property verification for increasing δ: $\delta = 0.5$ and $\delta = 0.7$.
   - At what temperature should we switch off the burner?
   - For $\delta = 0.5$, 93 is OK.
   - 88^o C, for $\delta = 0.7$, not *just* because of δ being too large (Nyquist criterion), but, also because of delay semantics between HIOA

Simulation results



Train Gate example





Train Gate System

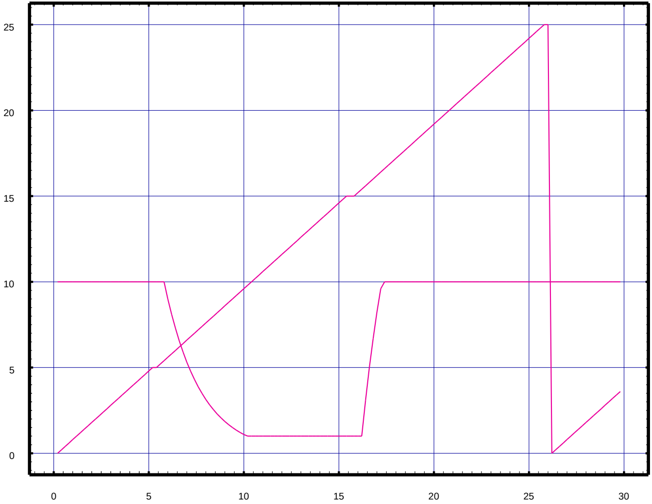
- Safety property: $G!(train\_pos == 0 \wedge GATE\_DOWN)$

Train Gate System

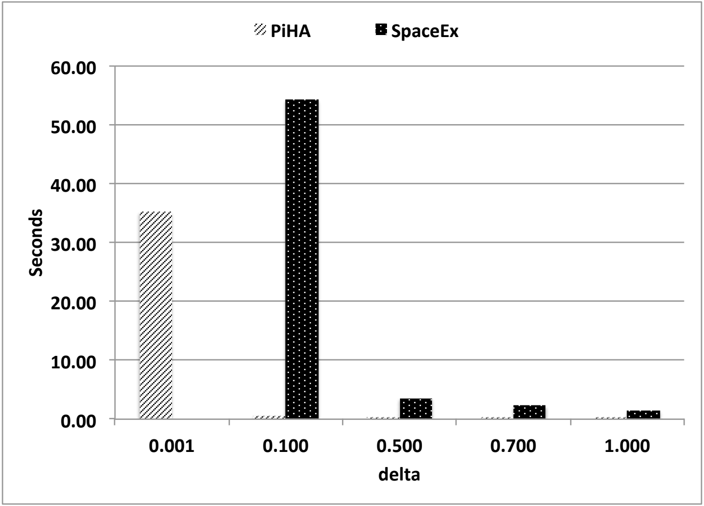
- Safety property: $G!(train\_pos == 0 \wedge GATE\_DOWN)$
- One possible way to implement the controller so that the safety property is satisfied is to **not** move the gate down at all!
- The above approach is *safe*, but *useless*
- Liveness property:
  $G((Gate\_DOWN \rightarrow F(Gate\_UP)) \wedge (Gate\_UP \rightarrow F(Gate\_DOWN)))$

- Safety property: $G!(train\_pos == 0 \wedge GATE\_DOWN)$
- One possible way to implement the controller so that the safety property is satisfied is to **not** move the gate down at all!
- The above approach is *safe*, but *useless*
- Liveness property: $G((Gate\_DOWN \rightarrow F(Gate\_UP)) \wedge (Gate\_UP \rightarrow F(Gate\_DOWN)))$
- Why did it fail? – Train model is wrong
- What is wrong? – The $\leq$ sign causes non-determinism!
- One possible option for the HIOA is to remain in location `slow-mode-1` indefinitely, when train reaches 5.
- Same with correct Train model – OK!.

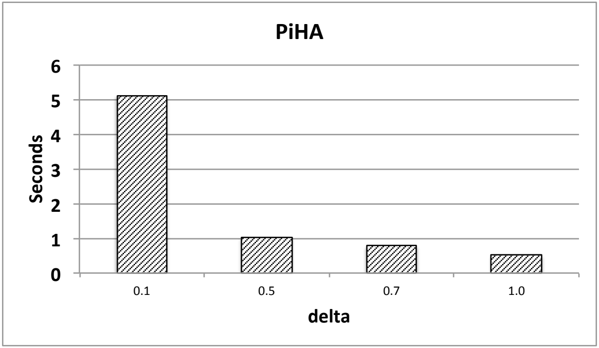
Train Gate Simulation results



Scalability – Runtime comparison SpaceEx vs. PiHa with reducing δ for WaterTankController safety property



Scalability – Liveness property verification PiHa **only**



Conclusions

1. PiHa + SPIN based property verification is scalable.
2. Reachability in PiHa +SPIN based approach is decidable.
3. Liveness property verification has become possible for the *very first time* for HIOA models, using the PiHa + SPIN approach.

Future Work

1. The heart HIOA model verification.
2. Handling non-monotonic Ordinary Differential Equations (ODEs)

Goran Frehse et al. "SpaceEx: Scalable Verification of Hybrid Systems". In: *Proceedings of the 23rd International Conference on Computer Aided Verification*. CAV'11. Snowbird, UT: Springer-Verlag, 2011, pp. 379–395. ISBN: 978-3-642-22109-5. URL: http://dl.acm.org/citation.cfm?id=2032305.2032335.

Sicun Gao, Soonho Kong, and Edmund Clarke. "Satisfiability modulo odes". In: *arXiv preprint arXiv:1310.8278* (2013).