Bridging the gap between embedded systems and automation systems

Partha S Roop

BioRemulation™ Research Group
The University of Auckland

August 5 2016

www.pretzel.ece.auckland.ac.nz/bio
1.1 Embedded and Automation Systems—are they really different?

(a) Automation system
(b) Embedded systems

- **Processing Unit:**
  - PLCs
  - Mains (24dc, 120Vac, 220Vac)

- **Application:**
  - Control

- **I/O response:**
  - PLC scan cycle (polling)

- **Manufacturer provided connectors**

- **Physical size:**
  - Large footprint

- **IEC 61131 (i.e. ladder logic, structured text, sequential function charts)**

- **Distribution:**
  - Distributed

- **Timing:**
  - Real-time

- **Connectivity:**
  - Manufacturer supported buses

- **Certification:**
  - Required

- **Quantity of deployment:**
  - Small numbers

- **Hardened for operational environments**

- **Programming:**
  - Shell: Power

- **I/O devices:**
  - Microcontroller I/O pins

---

(a) Yoong et al. Model driven design using IEC61499, Springer 2015.
1.1 Embedded and Automation Systems—are they really different?

(a) Automation system
- Processing Unit: PLCs
- Power: Mains (24dc, 120Vac, 220Vac)
- Timing: Real-time
- Connectivity: Manufacturer supported buses
- Certification: Required
- Physical size: Large footprint
- Application: Control
- Shell: Hardened for operational environments
- I/O response: PLC scan cycle (polling)
- Quantity of deployment: Small numbers

(b) Embedded systems
- Processing Unit: Microcontrollers
- Power: Mains or batteries
- I/O response: Interrupts or polling
- I/O devices: Microcontroller I/O pins
- Timing: Real-time and non-real-time
- Connectivity: Up-to the designer
- Certification: Mostly not required
- Distribution: Real-time and non-real-time
- Application: Control and data orientated
- Shell: Up-to the designer
- Physical size: Small footprint
- Quantity of deployment: Large numbers

---

Real-time control using PLCs alone is not feasible any longer.

PLCs are combined with FPGAs to bridge the gaps in timing. \(^a\): Using an FPGA allowed for encoding position signals to be handled directly from the sensors. No intermediate processing or amplification device was required, thereby reducing noise and increasing processing speed. In a process cycle faster than 1 ms, the valve position is measured and speed is calculated as both are compared to the set point. Movement is corrected using a PID algorithm. To keep the hydraulic circuit balanced, pressure values in the front and back of the cylinder are simultaneously controlled to avoid instantaneous peaks.

Question: How to develop a systematic approach that bridges the gap without using ad-hoc solutions?

\(^a\) Greenled D (2013) How embedded systems are changing automation, Automation World.
A family of languages developed in France in the early 80s, with identical view of concurrency. Inspired by synchronous circuits: all components trigger relative to a global clock. The reactive system operates infinitely fast relative to its environment. This is known as the synchrony hypothesis. Interleaving disappears in this semantics due to strict notions of causality.

---

A. Benveniste et al., The synchronous languages twelve years later, proceedings of IEEE, 91(1), 2003
IEC61499 – a cruise controller

4.2.1 The cruise control example

We now illustrate how to map a function block network to an equivalent Esterel program using Fig. 4.4, which models a cruise control system. It consists of three concurrently operating function blocks that have the following functionalities:

- **CruiseManager**: computes the state of the system and the desired cruising speed based on the current vehicle's speed, the buttons pressed, and the input from the brake pedal;
- **SpeedGauge**: computes the actual speed based on the input from a rotary encoder; and,
- **Throttle**: regulates the throttle position by computing the difference between the desired cruising speed and the actual speed, taking into account the cruise control's state and the depression of the accelerator. For illustration, the ECC for the Throttle function block has been depicted in Fig. 4.5.
module CruiseControl:
    input INIT, FootBrake, AccelHold, AccelRelease, CCOff;
    input Resume, Clock, RUN, Distance : value integer;
    output ThrottleChange, ThrottleValue : value integer;
    output SpeedChange, CurrentSpeed : value integer;
    signal Lever_SetDesiredSpeed, Lever_INITO,
             Lever_DesiredSpeed : value integer, ... in
        run Throttle [...] 
    || 
        run CruiseController [...] 
    || 
        run CruiseControlLever [...] 
    || 
        run SpeedGauge [...] 
end signal
end module
4.3.1.6 Sequential statement

Rule 4.11 expresses the fact that the sequence does not finish, if its left branch, \( t \), does not.

\[
\begin{align*}
\text{If } t \text{, then } & D \\
\text{otherwise, then } & D'
\end{align*}
\]

(4.11)

If the left branch pauses, so does the sequence.

\[
\begin{align*}
\text{If } t \text{, then } & D \\
\text{otherwise, then } & D
\end{align*}
\]

(4.12)

Moreover, if the left branch raises an exception (by exiting a trap), its right branch will never get executed.

\[
\begin{align*}
\text{If } t \text{, then } & D \\
\text{otherwise, then } & D
\end{align*}
\]

(4.13)

Otherwise, control will be immediately transferred to the right branch, \( u \), when \( t \) finishes.

\[
\begin{align*}
\text{If } t \text{, then } & D \\
\text{otherwise, then } & D
\end{align*}
\]

(4.14)
5.1 Revisiting Delayed Communication

We begin here by revisiting the issue of delayed communication, this time, reasoning about it in a more formal manner. Each input and output of a function block can be thought of as a sequence of values in some domain, space, or time. For a particular execution trace of the program in Fig. 4.4, each horizontal marker denotes a particular discrete time instant, such that $s_n(t)$ is some non-negative integer and $t$ is some non-negative real value. By stating that send and receive operations between function blocks are effectively take the form of equation 5.1:

\[ s_n(t) = \text{stateVar} \]

Equation 5.1 ensures the "pipelining" of the send and receive operations between function blocks. For a particular execution trace of the program in Fig. 4.7, each horizontal marker denotes a particular discrete time instant, such that $s_n(t)$ is some non-negative integer and $t$ is some non-negative real value. The input events in the diagram have all been delayed by one tick, then until the particular discrete time instant, such that $s_n(t)$ is some non-negative integer and $t$ is some non-negative real value. If we assume that accelPressed and brakePressed are emitted during initialization. Nothing happens until brakePressed occurs, the stateVar is set to 1 and await on line 30 ends. Meanwhile, nothing happens until brakePressed occurs. Meanwhile, stateVar is set to 2 and await on line 11 ends. ThrottleChg is then emitted. Time event connection loop in a function block network; (b) shows the pseudocode for implementing delayed communications between function blocks.

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);
\]

\[
\text{EI3}(k) = \text{EO1}(k-1);
\]

\[
\text{EI5}(k) = \text{EO2}(k-1);
\]

\[
\text{EI7}(k) = \text{EO3}(k-1);
\]

\[
\text{EI1}(k) = \text{EO3}(k-1);\]

\[
\text{EI3}(k) = \text{EO1}(k-1);\]

\[
\text{EI5}(k) = \text{EO2}(k-1);\]

\[
\text{EI7}(k) = \text{EO3}(k-1);\]

\[
\text{EI1}(k) = \text{EO3}(k-1);\]

\[
\text{EI3}(k) = \text{EO1}(k-1);\]

\[
\text{EI5}(k) = \text{EO2}(k-1);\]

\[
\text{EI7}(k) = \text{EO3}(k-1);\]

\[
\text{EI1}(k) = \text{EO3}(k-1);\]

\[
\text{EI3}(k) = \text{EO1}(k-1);\]

\[
\text{EI5}(k) = \text{EO2}(k-1);\]

\[
\text{EI7}(k) = \text{EO3}(k-1);\]

\[
\text{EI1}(k) = \text{EO3}(k-1);\]

\[
\text{EI3}(k) = \text{EO1}(k-1);\]

\[
\text{EI5}(k) = \text{EO2}(k-1);\]

\[
\text{EI7}(k) = \text{EO3}(k-1);\]

\[
\text{EI1}(k) = \text{EO3}(k-1);\]

\[
\text{EI3}(k) = \text{EO1}(k-1);\]

\[
\text{EI5}(k) = \text{EO2}(k-1);\]

\[
\text{EI7}(k) = \text{EO3}(k-1);\]
Semantics: parallel

\[
\begin{align*}
t, D \xrightarrow{O, \perp} t', D' \\
t \parallel u, D \xrightarrow{O, \perp} t' \parallel u, D' \\
u, D \xrightarrow{Q, \perp} u', D' \\
t \parallel u, D \xrightarrow{Q, \perp} t \parallel u', D'
\end{align*}
\]  \hspace{1cm} (4.15)

\[
\begin{align*}
u \xrightarrow{I, p} D' \\
t \xrightarrow{I, p} D' \\
u, D \xrightarrow{Q, \perp} u', D' \\
t \parallel u, D \xrightarrow{Q, \perp} t \parallel u', D'
\end{align*}
\]  \hspace{1cm} (4.16)

Rule 4.17 uses the completion code synchronizer to specify the synchronized behaviour of the parallel statement. When both \( t \) and \( u \) perform finished transitions, the parallel statement synchronizes their execution using their completion codes.

\[
\begin{align*}
t, D \xrightarrow{\emptyset, k} t', D \\
u, D \xrightarrow{\emptyset, l} u', D \\
t \parallel u, D \xrightarrow{\emptyset, \text{syn}(k, l)} t \parallel u', D
\end{align*}
\]  \hspace{1cm} (4.17)

As already mentioned in the description for data assignment, write-write concurrency on variables is disallowed, while read-write concurrency is semantically forbidden by rule 4.8. This means that \( t \) and \( u \) will never operate on the same variables in the same instant.

---

Outcomes

- **Determinism**: Given any state and any valid input combination, at most one transition is enabled.
- **Reactivity**: Given any state and any valid input combination, at least one transition is enabled.
- **Theorem**: Synchronous function blocks are deterministic and reactive.

The unification of embedded systems and automation systems requires the consideration of cyber-physical systems i.e. not just the controller but also the plant.

---

Cyber-physical systems (CPS)\(^a\) use distributed embedded controllers to control physical processes. Examples may be found in several domains: automotive, robotics, medical devices, and smart grids.


\(^1\)Figure reproduced from http://icc.mtu.edu/cps/
Hybrid automata (HA) is a major enabler for the formalization of CPS. A combination of ODEs to model the continuous dynamics and FSMs to model the discrete mode changes that are induced by the controller.

- Model: Car
  Discrete: Changing gears
  Continuous: Throttle control

- Model: Cell biology
  Discrete: External stimulus
  Continuous: Flow of ions
A water tank temperature controller

Plant

Water tank

Gas burner

Thermometer

Digital Controller

Figure 2. An example of a water tank heating system – adapted from [10]. Figure 2(a) presents the plant and the discrete controller. Figures 2(c) and 2(d) depict the continuous controller. Figure 2(e) illustrates the state-space of the hybrid system.

Definition 2. The semantics of an HIOA is specified using the notion of hybrid system makes this discrete change. The trace is a set of discrete points sampled every
Temperature of water inside a tank may be modelled as $x(t) = le^{-Kt} + h(1 - e^{-Kt})$ where:
- $l$ is the initial temperature.
- $K$ is a constant that depends on the tank conductivity.
- $h$ is a constant that depends on the power of the gas burner.
A hybrid automata example

- Four locations t1, ..t4 that represent the discrete modes.
- Each location has some flow predicates that specify the rate of change of the continuous variables.

---

Invariants are associated with locations e.g. $20 \leq x \leq 100$ is an invariant associated with $t_1$. Execution remains in a location until the invariant holds.

Some locations may have initialization conditions that provide the initial values of the variables.

A transition is enabled when the input is present and the jump condition associated with the transition holds. When a given transition is taken the final value of the variables are updated.
A hybrid automata example

![Diagram of hybrid automata example](image-url)
A hybrid automata example
A hybrid automata example

State: $t_2$, $x = 29.40$
- ON
- OFF

$20 \leq x \leq 100$
$\dot{x} = k(x - h)$

$t_2$
$T \land x = 100 \land \dot{x}' = 0$

$x = 100$
$t_3$
$T \land \dot{x} = 0$

$\dot{x} = 0$
$t_4$
$20 \leq x \leq 100$
$T \land x = 20 \land \dot{x}' = 0$

ON $\land \dot{x}' = x$

OFF $\land \dot{x}' = x$

Initial $\dot{x} = 0$

Temperature (°C)

Time (seconds)
A hybrid automata example

\[
\begin{align*}
\dot{x} &= k(h-x) \\
t &\in [20, 100] \\
x &= 100 \\
\dot{x} &= 0 \\
t &= 20 \\
20 \leq x \leq 100 \\
\dot{x} &= -kx \\
\end{align*}
\]
A hybrid automata example

State: t2 \( x: 46.20 \)

- ON
- OFF

20 \( \leq x \leq 100 \)

\( x = k(h - x) \)

\( 20 \leq x \leq 100 \)

\( x = 100 \)

\( x = 0 \)

\( x = 0 \)

\( x \leq 20 \)

\( x > 20 \)

\( T \wedge x = 100 \wedge x' = x \)

\( ON \wedge x = x \)

\( OFF \wedge x = x \)

\( ON \wedge x' = x \)

\( OFF \wedge x' = x \)

\( t2 \)

\( t3 \)

\( t4 \)

\( x = 20 \)

\( x = 0 \)

\( x = 0 \)

\( x \leq 20 \)

\( x > 20 \)

\( t1 \)

\( t4 \)

Temperature (°C)

Time (seconds)
A hybrid automata example
A hybrid automata example
A hybrid automata example

State: t2 \( x = 67.12 \)

ON

OFF

\( 20 \leq x \leq 100 \)

\( x \neq 100 \land \dot{x} = x \)

\( x = 100 \)

\( \dot{x} = 0 \)

Initial

\( x = 0 \)

\( T \land x = 0 \land \dot{x} = 0 \)

\( 20 \leq x \leq 100 \)

\( 0 \leq x \leq 100 \)

Temperature (°C)

Time (seconds)
A hybrid automata example

State: t2 \( x \leq 73.11 \)

ON \quad OFF

\[ x = k(h - x) \]

\[ 20 \leq x \leq 100 \]

\[ T \wedge x = 100 \wedge x' = x \]

\[ t2 \]

\[ ON \wedge x' = x \]

\[ OFF \wedge x' = x \]

\[ t3 \]

\[ x = 100 \]

\[ t4 \]

\[ 20 \leq x \leq 100 \]

\[ T \wedge x = 20 \wedge x' = x \]

\[ x' = 0 \]

\[ x = 20 \]

\[ initial \]

\[ 0 \leq x \leq 100 \]

\[ 0 \leq x \leq 120 \]

\[ Time (\text{seconds}) \]

\[ Temperature (^\circ C) \]
A hybrid automata example
A hybrid automata example

\[ x^* = k(h - x) \]

State: t2  \( x: 83.83 \)
ON  OFF

ON  \( x^* = x \)
OFF  \( x^* = x \)

\( 20 \leq x \leq 100 \)

\( x = 100 \)

\( x^* = 0 \)

\( x = 20 \)

\( x^* = 0 \)

\( x = 20 \leq x \leq 100 \)

Temperature (°C)

Time (seconds)
A hybrid automata example

State: t2 \( x < 88.61 \)

ON

OFF

\( 20 \leq x \leq 100 \)

\[ x \dot{} = k(h - x) \]

\( t2 \)

\( x = 100 \)

\[ x \dot{} = 0 \]

\( t3 \)

\( x = 20 \)

\[ x \dot{} = 0 \]

\( t1 \)

\( 20 \leq x \leq 100 \)

\[ x \dot{} = -kx \]

\( t4 \)

Temperature (°C)

Time (seconds)
A hybrid automata example

State: $t_2 \quad x: 93.05$
- ON
- OFF

$20 \leq x \leq 100$
$x_\bullet = k(h-x)$

$t_2$

$ON \land x' = x$

$t_3$

$x = 100$

$OFF \land x' = x$

$ON \land x' = x$

$OFF \land x' = x$

$t_1 \quad x = 20$
$x_\bullet = 0$

$t_4 \quad 20 \leq x \leq 100$
$x_\bullet = -kx$

Graph:
- Temperature vs. Time (seconds)
- Initial State
- Transition States

(University of Erlangen-Nuremberg) Invasive Computing Seminar
A hybrid automata example
A hybrid automata example

\[
x_\bullet = k(h - x)
\]

State: t3 \( x = 100.00 \)
- ON
- OFF

\[
x = 100
\]

\[
x_\bullet = 0
\]

\[
x = 20
\]

\[
x_\bullet = 0
\]

\[
T \land x = 20 \land x' = x
\]

\[
T \land x = 100 \land x' = x
\]

\[
OFF \land x' = x
\]

\[
ON \land x' = x
\]

\[
ON \land x = 20
\]

\[
20 \leq x \leq 100
\]

\[
20 \leq x \leq 100
\]

\[
x_\bullet = -kx
\]

\[
x_\bullet = 0
\]

\[
T \land x = 100 \land x' = x
\]

\[
ON \land x' = x
\]

\[
OFF \land x' = x
\]

\[
ON \land x = 20
\]

\[
20 \leq x \leq 100
\]

\[
20 \leq x \leq 100
\]

\[
T \land x = 20 \land x' = x
\]

\[
T \land x = 100 \land x' = x
\]

\[
OFF \land x' = x
\]

\[
ON \land x' = x
\]

\[
ON \land x = 20
\]

\[
20 \leq x \leq 100
\]

\[
20 \leq x \leq 100
\]

\[
T \land x = 100 \land x' = x
\]

\[
ON \land x' = x
\]

\[
OFF \land x' = x
\]

\[
ON \land x = 20
\]
A hybrid automata example

State: t3 \( x: 100.00 \)
- ON
- OFF

20 \( \leq x \leq 100 \)

\( \dot{x} = k(h - x) \)

\( T \wedge x = 100 \wedge \dot{x}' = x \)

\( t2 \)

ON \( \wedge \dot{x}' = x \)

OFF \( \wedge x \leq x \)

\( t3 \)

\( x: 100 \)

\( \dot{x} = 0 \)

\( t4 \)

20 \( \leq x \leq 100 \)

\( \dot{x} = k(x) \)

\( T \wedge x = 20 \wedge \dot{x}' = x \)

initial

\( x = 20 \)

\( 0 \) to \( 120 \)

Temperature (°C)

Time (seconds)
A hybrid automata example

\[ x_0 = k(h - x) \]

State: t3, x: 100.00
ON
OFF

20 ≤ x ≤ 100

\[ T \land x = 100 \land x' = x \]

x = 100

\[ T \land x = 20 \land x' = x \]

x = 20

\[ T \land x = 100 \land x' = x \]

\[ T \land x = 20 \land x' = x \]

\[ T \land x = 100 \land x' = x \]

\[ T \land x = 20 \land x' = x \]

Initial

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

\[ x_0 = 0 \]

Temperature (°C)

Time (seconds)
A hybrid automata example
A hybrid automata example

State: $t_3 \quad x: 100.00$

- **ON**
  - $20 \leq x \leq 100$
  - $\dot{x} = k(h - x)$

- **OFF**
  - $x = 100$
  - $\dot{x} = 0$

- **Initial**
  - $x = 20$
  - $\dot{x} = 0$

- **Transition**
  - $T \land x = 100 \land x' = x$
  - $T \land x = 20 \land x' = x$
  - $T \land x' = x$

Graph:

- Temperature ($^\circ C$)
  - $0 \rightarrow 120$

- Time (seconds)
  - $0 \rightarrow 50$
A hybrid automata example

State: t4  \( x = 100.00 \)
ON     OFF

\[ 20 \leq x \leq 100 \]
\[ x^\bullet = k(h - x) \]
t2

ON \( \land x' = x \)

ON \( \land x = 100 \land x' = x \)
t3

\[ x = 100 \]

OFF \( \land x = x \)

\[ x^\bullet = 0 \]

\[ x^\bullet = -kx \]

\[ t1 \quad x = 20 \quad \text{initial} \]

Time (seconds)

Temperature (°C)

\[ 20 \leq x \leq 100 \]
\[ t4 \quad 100 \leq x \leq 100 \]
A hybrid automata example

\[ x_0 = k(h - x) \]

- State: t4 \( x: \) 92.77
- ON
- OFF

- ON \( \wedge x = 100 \wedge x' = x \)
- OFF \( \wedge x = x \)

- initial \( x = 20 \)
- t1

- \( 20 \leq x \leq 100 \)
- t2

- \( x = 100 \)
- t3

- \( x = 0 \)
- t4

- \( 20 \leq x \leq 100 \)

- Temperature (°C)

- Time (seconds)
A hybrid automata example

State: t4  $x: 86.07$

- ON
- OFF

$20 \leq x \leq 100$

$t2$

$x^\bullet = k(h - x)$

$T \land x = 100 \land x' = x$

$t3$

$x = 100$

$ON \land x' = x$

$OFF \land x' = x$

$t1$

$x = 20$

$x^\bullet = 0$

initial

$t4$

$20 \leq x \leq 100$

$x^\bullet = -kx$

Graph:

- Temperature (°C)
- Time (seconds)
A hybrid automata example
A hybrid automata example

\[ x^* = k(h - x) \]

State: t4  \( x: 74.07 \)

ON  \( 20 \leq x \leq 100 \)
OFF

ON  \( x = 100 \) \( x^* = 0 \)
OFF  \( x = 100 \) \( x^* = 0 \)

ON  \( x^* = x \) \( t2 \)
OFF  \( x^* = x \) \( t3 \)

ON  \( x = 20 \) \( t1 \)
OFF  \( x = 20 \) \( t4 \)

\[ x^* = -kx \]

Temperature (°C)

Time (seconds)
A hybrid automata example
A hybrid automata example

State: t2 \( x: 63.80 \)

\( ON \quad OFF \)

\[ 20 \leq x \leq 100 \]

\[ x^* = k(h - x) \]

\[ \text{t2} \]

\[ \text{ON} \wedge x = 100 \wedge x' = x \]

\[ x = 100 \]

\[ \text{t3} \]

\[ \text{OFF} \wedge x' = x \]

\[ ON \wedge x' = x \]

\[ \text{initial} \]

\[ x = 0 \]

\[ \text{t1} \]

\[ T \wedge x = 20 \wedge x' = x \]

\[ x^* = 0 \]

\[ \text{t4} \]

\[ 20 \leq x \leq 100 \]

\[ x^* = -kx \]

\[ \text{Time (seconds)} \]

\[ \text{Temperature (°C)} \]

0 10 20 30 40 50

0 20 40 60 80 100 120

\( (University \ of \ Erlangen-Nuremberg) \)

Invasive \ Computing \ Seminar
A hybrid automata example

\[ x^\bullet = k(h - x) \quad \text{t1} \]

\[ x = 20 \]

\[ 20 \leq x \leq 100 \]

\[ T \land x = 100 \land x' = x \]

\[ \text{t2} \]

\[ x = 100 \]

\[ \text{OFF} \land x' = x \]

\[ \text{t3} \]

\[ x^\bullet = 0 \]

\[ \text{initial} \]

\[ ON \land x' = x \]

\[ \text{t4} \]

\[ 20 \leq x \leq 100 \]

\[ T \land x = 20 \land x' = x \]

\[ x^\bullet = -kx \]

\[ \text{State: t2} \quad x: 70.03 \]

\[ \text{ON} \quad \text{OFF} \]

\[ \text{Temperature (°C)} \]

\[ \text{Time (seconds)} \]
A hybrid automata example

\[ x^* = k(h - x) \]

State: t2    \( x: 75.81 \)
ON            OFF

20 \( \leq x \leq 100 \)

\[ x^* = 100 \land x' = x \]

\[ ON \land x' = x \]

\[ \neg ON \land x = x \]

\[ OFF \land x' = x \]

\[ t2 \]

\[ x = 20 \]

\[ t1 \]

\[ x = 0 \]

\[ t3 \]

\[ x = 100 \]

\[ t4 \]

\[ 20 \leq x \leq 100 \]

\[ x^* = 0 \]

\[ T \land x = 20 \land x' = x \]

\[ T \land x = 100 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ Time (seconds) \]

\[ Temperature (°C) \]

(University of Erlangen-Nuremberg)
A hybrid automata example

State: t2  \( x = 81.17 \)

ON  OFF

\( 20 \leq x \leq 100 \)

\( x = k(100 - x) \)

\( T \wedge \) \( x = 100 \wedge x' = x \)

\( x = 100 \)

\( x' = 0 \)

ON \( \wedge x' = x \)

OFF \( \wedge x' = x \)

\( x \leq x \)

ON \( \wedge x' = x \)

\( x \geq x \)

OFF \( \wedge x' = x \)

\( x = 0 \)

\( x' = k x \)

\( T \wedge \) \( x = 20 \wedge x' = x \)

\( x = 20 \)

\( T \wedge \)

\( 20 \leq x \leq 100 \)

\( x' = 0 \)

\( x' = -k x \)

\( t1 \)

\( x = 20 \)

\( t4 \)

\( 20 \leq x \leq 100 \)

\( x' = 0 \)

\( t3 \)

\( x = 100 \)

Temperature (°C)

Time (seconds)
A hybrid automata example

\[ x^* = k(h - x) \]

State: t2 \( x: 86.15 \)
ON \quad OFF

\[ 20 \leq x \leq 100 \]

\[ x = 100 \]

\[ x = 0 \]

initial

\[ x = 20 \]

\[ x^* = 0 \]

\[ T^x = 100 \wedge x' = x \]

\[ ON \wedge x' = x \]

\[ OFF \wedge x' = x \]

\[ 20 \leq x \leq 100 \]

\[ T^x = 20 \wedge x' = x \]

Temperature (°C)

Time (seconds)

(UUiversity of Erlangen-Nuremberg) Invasive Computing Seminar
A hybrid automata example

\[ x^* = k(h - x) \]

State: t2 \( \quad \times: 90.76 \)
- ON
- OFF

ON \( \land x = 100 \land x' = x \)

OFF \( \land x = 100 \land x' = x \)

ON \( \land x' = x \)

OFF \( \land x' = x \)

\[ x^* = 0 \]

\[ x = 20 \]

\( 20 \leq x \leq 100 \)

\[ x^* = -kx \]

\[ x = 100 \]

\[ x = 20 \]

\( 20 \leq x \leq 100 \)

\[ x = 0 \]

initial

Temperature (°C)

Time (seconds)
A hybrid automata example

State: t4  x: 95.00

ON

OFF

\[ 20 \leq x \leq 100 \]

\[ x^* = k(h - x) \]

\[ x = 100 \]

\[ x^* = 0 \]

t2

ON \land x' = x

OFF \land x' = x

\[ t_3 \]

\[ t_4 \]

\[ 20 \leq x \leq 100 \]

\[ x = 20 \]

\[ x^* = 0 \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 100 \land x' = x \]

\[ T \land x = 20 \land x' = x \]

\[ x^* = -kx \]

initial

\[ T \land x = 100 \land x' = x \]

\[ T \land x = 20 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]

\[ T \land x = 120 \land x' = x \]

\[ T \land x = 0 \land x' = x \]
A hybrid automata example

State: t4 \quad x: 81.77

ON

OFF

20 \leq x \leq 100

\dot{x} = k(h - x)

T \land x = 100 \land x' = x

x = 100

\dot{x} = 0

ON \land x = x

OFF \land x = x

ON \land x' = x

OFF \land x' = x

initial

x = 20

\dot{x} = 0

T \land x = 20 \land x' = x

x = 20

\dot{x} = -kx

20 \leq x \leq 100

t1
t2
t3
t4

Temperature (°C)

Time (seconds)
A hybrid automata example

\[
\begin{align*}
\text{State: t4} & \quad x = 75.86 \\
\text{ON} & \quad \rightarrow \quad \text{OFF}
\end{align*}
\]

20 \leq x \leq 100

\[
x' = k(h - x)
\]

\[
t2
\]

\[
\begin{align*}
\text{ON} & \quad \rightarrow \quad \text{OFF}
\end{align*}
\]

\[
x = 100
\]

\[
\begin{align*}
\text{OFF} & \quad \rightarrow \quad \text{ON}
\end{align*}
\]

\[
x = 0
\]

\[
t3
\]

\[
\begin{align*}
\text{ON} & \quad \rightarrow \quad \text{OFF}
\end{align*}
\]

\[
x = 20
\]

\[
\begin{align*}
\text{OFF} & \quad \rightarrow \quad \text{ON}
\end{align*}
\]

\[
x' = -kx
\]

\[
t4
\]

\[
\begin{align*}
20 \leq x \leq 100
\end{align*}
\]

Graph:

- X-axis: Time (seconds)
- Y-axis: Temperature (°C)

Initial conditions:

- Initial state: x = 0
- x = 20

Transition rules:

- ON: x = 100
- OFF: x = 0
A hybrid automata example
A hybrid automata example

State: t4   \( x: 65.29 \)
ON  OFF

\( 20 \leq x \leq 100 \)

\( x' = k(h - x) \)

\( x = 100 \)

\( t2 \)

\( t3 \)

\( x' = x \)

\( x' = x \)

\( x' = x \)

\( x' = x \)

\( x = 20 \)

\( x = 0 \)

\( x = 0 \)

\( x' = -kx \)

\( x' = -kx \)

\( x' = -kx \)

Initial

\( t1 \)

\( t4 \)

\( 20 \leq x \leq 100 \)

\( x = 20 \)

\( x = 0 \)

\( 0 \leq x \leq 100 \)

Graph:

Temperature (°C)

Time (seconds)
A hybrid automata example

State: t4 \( x = 60.57 \)
ON \( 20 \leq x \leq 100 \)
OFF

\( x^\bullet = k(h - x) \)

\( x = 100 \)

\( x^\bullet = 0 \)

\( x = 20 \)

\( x^\bullet = 0 \)

\( t1 \)

\( t2 \)

\( t3 \)

\( t4 \)

\( 20 \leq x \leq 100 \)

\( x^\bullet = -kx \)

\( ON \wedge x = 100 \wedge x' = x \)

\( OFF \wedge x' = x \)

\( ON \wedge x' = x \)

\( OFF \wedge x' = x \)

Initial

Temperature (°C)

Time (seconds)
A hybrid automata example

\[ x_0 = k(h - x) \]

State: t4 \( x: 56.19 \)

ON \( x = 100 \)
OFF \( x = x' \)

\[ x_0 = 0 \]

ON \( x' = x \)
OFF \( x = x' \)

initial \( x = 20 \)

\[ x_0 = 0 \]

ON \( x = 20 \)
OFF \( x' = x \)

\[ x_0 = -kx \]

20 ≤ x ≤ 100

x = 100

20 ≤ x ≤ 100

Time (seconds)

Temperature (°C)
A hybrid automata example

State: $t_4$, $x = 52.13$

ON

OFF

$20 \leq x \leq 100$

$x_\cdot = k(h - x)$

$T \land x = 100 \land x' = x$

$t_2$

$ON \land x' = x$

$OFF \land x' = x$

$x = 100$

$x_\cdot = 0$

$t_3$

$ON \land x = 0$

$OFF \land x = 0$

$x_\cdot = -kx$

$t_4$

$20 \leq x \leq 100$

$x_\cdot = k(h - x)$

$ON \land x' = x$

initial

$t_1$

$x = 20$

$T \land x = 20 \land x' = x$

$120$

Temperature ($^\circ C$)

$0$

$0$

$10$

$20$

$30$

$40$

$50$

Time (seconds)
A hybrid automata example
A hybrid automata example
A hybrid automata example
A hybrid automata example
A hybrid automata example
A hybrid automata example

State: t2 \[ x: 63.49 \]

ON

OFF

\[ 20 \leq x \leq 100 \]

\( x^* = k(h - x) \)

\( T \land x = 100 \land x' = x \)

\( x = 100 \)

\( x^* = 0 \)

\( \text{ON} \land x' = x \)

\( \text{OFF} \land x' = x \)

\( \text{ON} \land x' = x \)

\( \text{OFF} \land x' = x \)

\[ x^* = 0 \]

\[ x = 20 \]

\( T \land x = 20 \land x' = x \)

\( x^* = 0 \)

\[ 0 \leq x \leq 100 \]

\( x' = -kx \)

\( x^* = 0 \)

\( x = 20 \)

\( 20 \leq x \leq 100 \)

\( T \land x = 20 \land x' = x \)

\( x^* = 0 \)

\( x = 100 \)

\( x^* = 0 \)

\( \text{ON} \land x' = x \)

\( \text{OFF} \land x' = x \)

\( \text{ON} \land x' = x \)

\( \text{OFF} \land x' = x \)

Temperature (°C)

Time (seconds)
A hybrid automata example

\[
\begin{align*}
\text{State: t2} & \quad x: 69.74 \\
\text{ON} & \quad \text{OFF} \quad \text{ON} \quad \text{OFF} \\
20 \leq x \leq 100 & \quad x = 100 \\
\dot{x} &= k(h - x) & \dot{x} &= 0 \\
\text{t2} & \quad \text{t3} \\
\text{ON} \wedge x = 100 \wedge x' = x & \quad \text{OFF} \\
\text{OFF} \wedge x = x & \quad \text{OFF} \wedge x' = x \\
\text{t1} & \quad \text{t4} \\
\text{initial} & \quad x = 20 \\
\dot{x} &= 0 & \dot{x} &= -kx \\
20 \leq x \leq 100 & \quad 20 \leq x \leq 100 \\
\end{align*}
\]
A hybrid automata example

Invasive Computing Seminar
A hybrid automata example

\[ x^* = k(h - x) \]

- **t2**: \( 20 \leq x \leq 100 \)
- **t3**: \( x = 100 \)
- **t4**: \( 20 \leq x \leq 100 \)

\[ \text{State: t2} \quad x: 80.93 \]

\[ \text{ON} \quad \text{OFF} \]

\[ x^* = 0 \]

\[ \text{ON} \quad \text{OFF} \]

\[ x^* = -kx \]

\[ x = 20 \]

\[ \text{initial} \]

\[ x^* = k(h - x) \]

\[ T \wedge x = 100 \wedge x' = x \]

\[ x = 20 \]

\[ T \wedge x = 20 \wedge x' = x \]

\[ \text{Temperature (°C)} \]

\[ \text{Time (seconds)} \]
A hybrid automata example
A hybrid automata example

![Diagram of a hybrid automata example](image)

- **State:** t2, x: 90.55
  - ON
  - OFF

- **Conditions:**
  - $20 \leq x \leq 100$
  - $x' = x$
  - $T \land x = 20 \land x' = x$
  - $T \land x = 100 \land x' = x$
  - $x = 0$
  - $x = 100$

- **Initial State:** $x = 20$
- **Outputs:**
  - Temperature (°C)
  - Time (seconds)
A hybrid automata $H = < Loc, Edge, \Sigma, Inv, Flow, Jump >$ where

- $Loc = \{l_1, \ldots, l_n\}$ representing $n$ control modes or locations.
- $\Sigma$ is the input alphabet comprising of event names.
- $Edge \subseteq Loc \times \Sigma \times Loc$ are the set of edges between locations.
- Three sets for the set of continuous variables, their rate of change and their updated values represented as follows: $X = \{x_1, \ldots, x_m\}$ $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_m\}$ $X' = \{x'_1, \ldots, x'_m\}$.
- $Init(l)$: Is a predicate whose free variables are from $X$. It specifies the possible valuations of these when the HA starts in $l$.
- $Inv(l)$: Is a predicate whose free variables are from $X$ and it constrains these when the HA resides in $l$.
- $Flow(l)$: Is a predicate whose free variables are from $X \cup \dot{X}$ and it specifies the rate of change of these variables when the HA resides in $l$.
- $Jump(e)$: Is a function that assigns to the edge $e$ a predicate whose free variables are from $X \cup X'$. This predicate specifies when the mode switch using $e$ is possible. It also specifies the updated values of the variables when this mode switch happens.
Semantics

Definition

The semantics of a HA \( H = < Loc, Edge, \Sigma, Inv, Flow, Jump > \) is provided using a timed transition system \( TTA = < Q, Q_0, \Sigma, \rightarrow > \)

- \( Q \) is for the form \((l, v)\) where \( l \) is a location and \( v \in [\mathbb{X} \rightarrow \mathbb{R}] \) such that \( v \) satisfies \( Inv(l) \). \( Q \) is called the state space of \( H \).
- \( Q_0 \subseteq Q \) of the form \((l, v)\) such that \( v \) satisfies \( Init(l) \).
- \( \rightarrow \) is the set of transitions consisting of either:
  - Discrete transitions: For each edge \( e = (l, \sigma, l') \), we have \((l, v) \xrightarrow{\sigma} (l', v')\) if \((l, v) \in Q\), \((l', v') \in Q\) and \((v, v')\) satisfy \( Jump(e) \). These take zero time.
  - Continuous transitions: When control remains in a location and time progresses. Here the continuous variables evolve according to the ODEs as long as the invariant holds.
Simulation-based validation

- This validation is usually done open-loop.
- System under validation is stimulated using an input trace to observe its response.
- Limitations: Coverage criteria dependent, exhaustive simulation infeasible.

Simulation for validating a pacemaker
What is emulation?

- Operating a **controller** under test in closed-loop with the actual physical process (the **plant or the environment**) [5].
- The design of the controller follows the principles of real-time systems.
- The controller is digital in nature, while the plant usually exhibits continuous dynamics and is uncontrollable.

Emulation for validating a pacemaker
(Actual heart + pacemaker model)
Limitations of emulation

- The plant and the controller may need to be designed in parallel i.e. a rehabilitation robot.
- Model-in-the-loop simulation using Simulink and Stateflow: semantic issues [7, 1] and issues with model fidelity.
- Ptolemy [6] and Zélus [1] are tools with formal semantics. However, these are suitable for the modelling of closed systems using HA models. Also, like SL/SF they interact dynamically with ODE solvers. This is not good for emulation.
- Potemy has incorporated a QSS-based solver [2] to overcome the above. This, however, is unsuitable for open systems.
Problem statement

There is no approach for black-box validation of controllers (say Pacemakers) using real-time plant models. This requires:

- Open models of the plant using a network of hybrid input output automata (HIOA [4]).
- Formal semantics of HIOA models and their compositions.
- Automatic techniques for modular code generation.
- Static timing analysis of the plant for plant-controller timing compatibility i.e. correct timing verification to ensure that the sampling time of the plant and controller match [3]

We propose the new technique of remulation for this.
What is remulation?

Remulation stands for reverse emulation using an executable model of the plant, we term a plant-on-a-chip (PoC). During remulation, the plant-controller relationship is reversed.

- We have to synthesize a suitable model of the r-controller (the traditional plant).
- The r-plant (the usual controller) acts as an environment for the r-controller. The r-plant is black-box in nature.

Remulation for validating a pacemaker
(Heart model (real-time) + pacemaker actual/model)
As alternatives to the actual plant-based models, commercial tools such as Simulink® or Stateflow® [12] provide modular code generation from hybrid system models. Validation is extensively used in the automotive domain [17]. Here, a model of the plant is implemented on a suitable hardware for the closed-loop system formed by deploying the PoC and the black box controller together used for controller validation [16]. Of particular importance is the need for emulation-based validation due to the reliance on numerical solvers when generating C code from the plant model. However, these rely on code generation from these tools. Moreover, we show in the existing works that the generated code from these tools have scalability issues associated with these tools. Additionally, we demonstrate in Section 5 that the generated code from these tools have scalability issues associated with these tools. Consequently, our code generation relies on the well-known tenets of the synchronous approach. An overview of the proposed methodology is presented in Figure 1. We introduce an abstraction called Synchronous Hybrid Input Output Automata (S-HIOA), which is the default standard for designing parallel composition. Such S-HIOA are easy to compile since dynamic numerical solvers hinder the WCET analysis. However, the negative decidability results mean that both the reachability problem over hybrid automata, i.e., even the simplest class, called Hybrid Input Output Automata (HIOA) of Lynch et al. [19], is undecidable [10]. Several restrictions have been made open-source.

The design of CPS requires both formal analysis [21] and testing-based validation. Of particular importance is the need for emulation-based validation [16], as outlined above. Invasive Computing Seminar
Overview

1. Introduction
2. Background
3. Motivation / problem statement
4. Methodology
5. Compiling HIOA
   - Compilation overview
   - The first step of the compilation procedure
   - The second step of the compilation procedure
   - Correctly handling the invariant conditions
6. From a cell to the conduction network
   - Models
7. Results

(University of Erlangen-Nuremberg) Invasive Computing Seminar
Figure: Overview of the proposed modular code generation approach
Overview

1. Introduction
2. Background
3. Motivation / problem statement
4. Methodology
5. Compiling HIOA
   - Compilation overview
   - The first step of the compilation procedure
   - The second step of the compilation procedure
   - Correctly handling the invariant conditions
6. From a cell to the conduction network
   - Models
7. Results
**Discretisation**: In HA semantics, when control resides in a location, there are infinite valuations of variables in any given interval of time. This makes code generation difficult. To facilitate code generation, we make evaluations only at discrete intervals. These intervals correspond to the *ticks* of a synchronous program that will be used for code generation.

**Symbolic solution**: The ODEs which define flow constraints in any location of the form $\dot{x} = f(x)$ must be of *closed form* nature. This property ensures that such ODEs are symbolically solvable so that the witness functions needed for the generated code are symbolically computable.

**Monotonicity**: All witness functions must be monotonic. This property is needed so that the generated code can compute correct valuation of invariants and jump conditions.
Recap: the heating system

\[ \begin{align*}
\dot{x} &= K(h - x) \\
20 \leq x \leq 100 &\quad t_2 \\
\dot{x} &= 0 &\quad x = 100 &\quad t_3 \\
\dot{x} &= -K x &\quad 20 \leq x \leq 100 &\quad t_4 \\
\dot{x} &= 0 &\quad x = 20 &\quad t_1 \\
\text{initial} &\quad x = 20
\end{align*} \]
Step-1: translating ODEs to witness functions

Symbolic approach using the synchronous abstraction

\[ \dot{x} = K(h - x), \quad K = 150, \quad h = 0.075 \]  \hspace{1cm} (1)

\[ x[k] = C_1 \times e^{-0.075 \times \delta \times k} \]  \hspace{1cm} (2)

Explanation

- **Equation (1)**, Ordinary Differential Equation (ODE) captures the evolution of the continuous variable \( x \) that represents the temperature in the tank.
- The witness function, for the ODE, is the symbolic (closed form solution) to the ODE, if one exists.
- **Equation (2)** evolves \( x \) iteratively while the invariant condition \((10 \leq x \leq 100)\) on the location \((t_2)\) holds.
- This iterative evolution of the continuous variables at discrete points in time is akin to transitions on a logical *tick* of a synchronous program.
- We term the Hybrid Input Output Automata (HIOA) obtained after replacing each ODE with its equivalent witness function *Synchronous Hybrid Input Output Automata (SHIOA)*.
Overview

1. Introduction
2. Background
3. Motivation / problem statement
4. Methodology
5. Compiling HIOA
   - Compilation overview
   - The first step of the compilation procedure
   - The second step of the compilation procedure
   - Correctly handling the invariant conditions
6. From a cell to the conduction network
   - Models
7. Results
Compilatjon step 2: Compiling HIOA to SHIOA

\[ aF1 = C1 \times e^{-0.075 \times t} + 150.0 \]

and \( F2 = C1 \times e^{-0.075 \times t} \)
Compilation step 3: SWIOA / back-end code generation

\[ aF_1 = C_1 \times e^{-0.075 \times t} + 150.0 \]

and \[ F_2 = C_1 \times e^{-0.075 \times t} \]
Overview

1. Introduction
2. Background
3. Motivation / problem statement
4. Methodology
5. Compiling HIOA
   - Compilation overview
   - The first step of the compilation procedure
   - The second step of the compilation procedure
   - Correctly handling the invariant conditions
6. From a cell to the conduction network
   - Models
7. Results
Need for saturation

\( \dot{x} = 0.2x \quad \text{if} \quad x \leq 120 \)
\( \dot{x} = 0 \quad \text{if} \quad x \geq 100 \)

\( \dot{y} = 8.5 \quad \text{if} \quad y < 50 \)
\( \dot{y} = 0 \quad \text{if} \quad y \geq 50 \)

\( \dot{z} = -5.5 \quad \text{if} \quad z > -30 \)
\( \dot{z} = 0 \quad \text{if} \quad z \leq -10 \)

(a) Case 1: an increasing function and it does not need saturation
(b) Case 2: due to equality there is a need for saturation
(c) Case 3: a decreasing function and it does not need saturation

**Figure:** The need for saturation depends on the location invariant, the guard in HA, and the step size. Out of the three cases, only Case 2 requires saturation, see Figure 2(d).
Saturation Lemma

Lemma

*It is always possible to uniquely determine the saturation value for any continuous variable at time instant* $k$ *when the state (location) switch from* $l$ *to* $l'$ *is to be taken in a SHIOA.*

Proof.

The proof of this lemma follows from the following observations.

- **Observation 1:** All witness functions $x(t)$ for any $x \in X$ are monotonic in every location *(additional requirement).*
- **Observation 2:** All witness functions $x(t)$ for any $x \in X$ are continuous as they are differentiable in any interval.
- **Observation 3:** Given the above two observations, the saturation value for any variable $x$ always exists in the time interval $[(k - 1) \times \delta, k \times \delta]$ when the location switch happens at instant $k \times \delta.$
Overview

1. Introduction
2. Background
3. Motivation / problem statement
4. Methodology
5. Compiling HIOA
   - Compilation overview
   - The first step of the compilation procedure
   - The second step of the compilation procedure
   - Correctly handling the invariant conditions
6. From a cell to the conduction network
   - Models
7. Results
HA based on Ventricular AP.
Conduction Network
On average **9.8 times faster** than Simulink. For the heart conduction system it is two orders of magnitude faster.
Scalability Relative to Simulink

The Scalability of Piha and SimulinkR as the number of cells in the Network of Heart Nodes (NHN) model increases. In the example of the NHN, Piha is generally more compact than SimulinkR, being 54% smaller on average when compared to SimulinkR. For instance, SimulinkR memory usage at a 297 cell network is 1.8GB.

Figure 5 shows the scalability in execution time of SimulinkR and Piha of the NHN. The results are shown in Figure 5, with the most obvious feature being that no data is recorded for SimulinkR once the memory usage exceeds this limit. Piha, on the other hand, is able to continue past this point.

Figure 6(a) shows that for all benchmarks the step size in SimulinkR is fixed to 0.1 millisecond step size. For our most complicated example, the NHN, the step size being 5 times larger than SimulinkR.

Figure 6(b) shows that the code generated by SimulinkR is mostly in executable .dll or .so files, whereas Piha is mostly in C code. Similarly, for SimulinkR, we generate equivalent C code, and compile it using a standard C compiler. The C code generator. Similarly, for Piha, we generate equivalent C code, and compile it using the Microsoft Visual C++ compiler. Piha code was compiled with optimisation level O2.

SimulinkR generates automatically optimised executables, whereas Piha is manually optimised. SimulinkR code was compiled using the in-built requirement that the generated code use less than 2GB of memory. This discontinuity represents the point after which the change in gradient of Piha around the number of heart nodes is fixed to 0.1 milliseconds this translates to 1 microsecond.

The experiments were evaluated using an Intel i7-4790 processor with 8 GB RAM on Windows 7. We compare each of these benchmarks against SimulinkR implementation with the memory usage exceeding the limit. Piha, on the other hand, is able to maintain real-time with a model roughly 40 times more scalable.

These results also illustrate that Piha has a smaller increase of memory. This discontinuity represents the point after which the memory usage exceeds this limit.1 Piha, on the other hand, is able to continue past this point.

5 times more scalable
40 vs 200 cells for real-time emulation
Designing Plant-on-a-Chip (PoC) for cyber-physical systems using IEC-61499
Avinash Malik, Partha S Roop, and Théo Steger

Invasive Computing Seminar

Metho
dology fo r P oC design fo r automation
discrete grammable Logic Controller (PLC) programming languages such as IEC-61131-3 [4] and IEC-61499 [5], [6], are based on formal semantics of Pro-
called the P
(CPS), Simulink, Co-simulation. —IEC-61499, Hybrid Automata, Cyber Physical Systems

The plant simulation and the PLC execute at their individual pace – termed as

Lack of time synchronization

emulation of physical processes in industrial automation, without using the actual plant. Thus, a PoC may be used for emulation of diverse controllers connected to a discrete controller to provide real-time response like the one of a suitable abstraction of a plant on a computer chip, so that it can be employed during co-simulation.

During co-simulation, the continuous dynamics of the plant can interact with the model of a PLC or a hardware PLC running the control logic. In the latter case it is also called hardware in the loop simulation. TCP/IP as described in [2] can interact with the model of a PLC or a hardware PLC running the control logic. In the latter case it is also called hardware in the loop simulation. TCP/IP as described in [2]

Most PLC applications are considered safety critical, where a single fault can lead to catastrophic damage [7]. In view of the safety critical nature of PLCs, rigorous validation of the PLC control logic is an essential component of controller design. During the design phase, the controller properties are validated using a plant model. The discrete

Invasive Computing Seminar

Fail–review the specification

Plant specification is synchronous composition of HIOAs (step 1)

Controller designed as CFB

Fail–refine controller

Synchronously composed BFBs into plant CFB (step 5)

Validation Controller CFB || Plant CFB (step 6)

Checking well-formed properties of HIOA (step 2)

Pass

HIOAs translated to SHIOAs (step 3)

SHIOAs translated to BFBs (step 4)

Validated controller

Pass

Fail–review the specification

To this end, we propose the first approach for the design of Plant-

5 INTRODUCTION

We use a well-known formal model for CPS, called Hybrid Input Output Automata (HIOA), as the main vehicle in the proposed formu-

A physical process (the plant) may be described as a synchronous composition of a network of such HIOA. We provide an approach to transform such a network to a Composite Function Block (CFB) in IEC-61499. This transformation is also shown to be semantics preserving.

The plant in order to guide it to the required state. The plant PLC program samples inputs from the plant, computes the desired state of the plant and finally emits outputs back to or OPC [10] are the common communication mechanisms employed during co-simulation.

Fail–refine controller

Validated controller

Validation Controller CFB || Plant CFB (step 6)

Pass

Synchronously composed BFBs into plant CFB (step 5)

Checking well-formed properties of HIOA (step 2)

HIOAs translated to SHIOAs (step 3)

SHIOAs translated to BFBs (step 4)
We discuss the need for a unified design approach for embedded / automation systems.

We adopt the well known synchronous approach for designing the controller (using the IEC61499 standard) and the plant (also using the same standard).

We propose a new technique called reverse emulation (remulation) to design a plant-on-a-chip (PoC).

A PoC is a model of the plant that offers real-time response similar to the actual plant.

A key idea is to avoid dynamic interaction with numerical solvers to enable real-time implementations.

The approach is based on a new class of hybrid automata, we call synchronous hybrid automata.

We have compared our approach with Simulink and the results are favourable.

Future work: comparison with Ptolemy and the QSS-based approach,
Key references


