Model-checking of Real-Time Systems

Paul Pettersson
Embedded Systems, IDT, MDH

A national Swedish Strategic Research Centre

Paul Pettersson

• 1993 – M.Sc. In Computer Science
• 1999 - Ph.D. in Computer Systems,
• 1999-2000 – Research Assistant
  – Aalborg University, Denmark,
• 2000- Lecturer at Uppsala University
  – Design and analysis, planning, schedulability analysis, and model-based testing,
• 2006 – Associate professor (docent) at Uppsala University
• 2007 – Professor at Mälardalen University, IDT.
  – Component-based and resource constrained embedded systems
  – Leader of Embedded systems division and RTS modeling and analysis group (to 2011)
  – Director of Embedded systems research at IDT (from 2012)
Collaborators

**@UPPsala**
- Wang Yi
- Paul Pettersson
- John Håkansson
- Anders Hessel
- Pavel Krcal
- Leonid Mokrushin
- Shi Xiaochun

**@AALborg**
- Kim G Larsen
- Gerd Behrman
- Arne Skou
- Brian Nielsen
- Emmanuel Fleury
- Alexandre David
- Jacob Illum Rasmussen
- Marius Mikucionis

**@Elsewhere**
- Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Jörgen Trettman, Jürgen Brinksma, Martijn Fraanje, Franck Cassez, Magnus Lindahl, François Laroussinie, Patricia Bouyer, Augusto Burgueno, H. Bovmann, D. Latella, M. Lundqvist, Lars Asplund, Justin Pearson...

**@MDH**
- Cristina Seceleanu
- Aida Causevic
- Aneta Vulgarakis
- Jagdish Suryadevara
- Stefan Björnander
- Leo Hatvani

---

**Real-Time Systems**

A system where correctness not only depends on the logical order of events but also on their **timing!!**

**E.g.:**
- Air Bags, Cruise Control, ABS
- Process Control, Production Lines, Robots
- Real-time Protocols
- DVD/CD Players
Real-Time Model-Checking

Plant
Continuous

Controller Program
Discrete

sensors

actuators

Model of environment (user-supplied)

UPPAAL Model

Model of tasks (automatic?)

A – Model: Network of Timed Automata

F – Requirement: temporal logical formula, e.g.
   – Invariant: something bad will never happen, something may happen
   – Liveness: something will eventually happen

Model-Checking

Model: A

Requirement Specification: F

UPPAAL

A satisfies F

Yes!

No!
Diagnostic Information
Formal design and analysis

Modeling ➔ Simulation ➔ Verification

Example model-based verification

- Is this error state reachable?
- Is this component always operating in this state?
- Is this variable value always less than 64?
- Max response time between reaching these two states?
- Is the system guaranteed to reach this state?
Model-checking of Real-Time Systems

- Modelling Formalism
- Algorithm + Datastructures
- Applications

Finite state automata

- Finite state graph, with
  - Set of nodes (states)
  - Set of edges (transitions)
  - Set of labels (actions)
Light Control

Wanted Behaviour:
- pressed once = light
- pressed twice quickly = light will get brighter
- pressed again = light off.

Finite state automata with variables

- Extend FSA with variables e.g.
  - Relational automata and/or guarded commands
    - Guards and assignments on transitions
    - Maybe infinite state, but finite state for bounded domain
  - Time automata is another example
    - Guards and reset over clock variables on transitions
    - Infinite state!
- Semantics: Transition Systems
Timed Automata \textit{Alur & Dill 1990}

- **Guard**
  - Timing constraints e.g. $X>10$

- **Action**
  - Synchronization e.g. $a$

- **Clock reset**
  - Reset clock to 0 e.g. $X:=0$

---

**Light Control**

\textit{Wanted Behaviour:}

- pressed \textit{once} = light
- pressed \textit{twice quickly} = light will get brighter
- pressed \textit{again} = light off.
**SOLUTION:** Add real-valued clock $x$ to measure the delay between press events.

---

**Timed Automata: Semantics**

**Clocks:** $x, y$

- **Guard:** Boolean combination of integer bounds on clocks.
- **Reset:** Action performed on clocks.
- **State:** $(location, x=v, y=u)$ where $v,u$ are in $\mathbb{R}$.

**Transitions**

- Discrete Trans $a$: $(n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)$
- Delay Trans $e(1.1)$: $(n, x=2.4, y=3.1415) \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)$
Timed Automata with Invariants

Clocks: $x, y$

Transitions

$\begin{align*}
(n, x=2.4, y=3.1415) & \xrightarrow{e(2)} (n, x=2.4, y=3.1415) \\
(n, x=2.4, y=3.1415) & \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)
\end{align*}$

Invariants ensure progress!!

Clock Constraints

For set $C$ of clocks with $x, y \in C$, the set of clock constraints over $C$, $\Psi(C)$, is defined by

$$\alpha ::= x < c \mid x - y < c \mid \neg \alpha \mid (\alpha \land \alpha)$$

where $c \in \mathbb{N}$ and $\prec \in \{<, \leq\}$.
Timed Automata: Example

Timed Automata: Example

\[ X := 0 \]

\[ X \geq 2 \]
Timed Automata: Example

\[ 2 \leq x \leq 3 \]

\[ X := 0 \]

\[ X := 0 \]
Timed Automata: Example

\begin{align*}
X &= 0 \\
X &\geq 2 \\
X &\leq 3 \\
X &= 0
\end{align*}
Timed Automata: Example
(periodic task, period 20)

\[
x = 20 \\
x \leq 20 \\
x := 0
\]

Timed Automata: Example
(sporadic task w min period 20)

\[
x \geq 20 \\
x := 0
\]
Timed Automata: Example
(aperiodic task, every 5 to 100)

\[ 5 \leq x \leq 100 \]

\[ x := 0 \]

Timed Automata: Light Switch

- Switch may be turned on whenever at least 2 time units have elapsed since last "turn off".
- Light automatically switches off after 9 time units if it is not pressed.
Semantics Definition

- **Clock valuations:** \( V(C) \quad v : C \rightarrow R \geq 0 \)
- **State:** \((l,v) \) where \( l \in L \) and \( v \in V(C) \)
- **Action transition** \((l,v) \xrightarrow{a}(l',v') \) iff \( g(v) \) and \( v' = v[r] \) and \( \text{Inv}(l')(v') \)
- **Delay transition** \((l,v) \xrightarrow{d}(l,v + d) \) iff \( \text{Inv}(l)(v + d) \) whenever \( d' \leq d \in R \geq 0 \)

Timed Automata: Example

\[
\begin{align*}
\text{(off, x = y = 0)} & \xrightarrow{3.5} \text{(off, x = y = 3.5)} \\
\text{(on, x = y = 0)} & \xrightarrow{\pi} \text{(on, x = y = \pi)} \\
\text{(on, x = 0, y = \pi)} & \xrightarrow{3} \text{(on, x = 3, y = \pi + 3)} \xrightarrow{9-(\pi + 3)} \\
\text{(on, x = 9 - (\pi + 3), y = 9)} & \xrightarrow{\text{click}} \text{(off, x = 0, y = 9)} 
\end{align*}
\]
Networks of Timed Automata
with (finite) integer variables

Example transitions
(l1, m1, ..... , x=2, y=3.5, i=3,.....) \[\tau\] (l2, m2, ..... , x=0, y=3.5, i=7,.....)

Two-way synchronization on complementary actions.
Closed Systems!

Train Crossing [WPD-FORTE'94]
Train Crossing

Communication via channels and shared variable.

Queue

Stopable Area

Gate

How to specify what to check

SPECIFICATION OF REQUIREMENTS
How to specify what to check?!?

Model: A

Requirement Specification: F

A – Model: Network of Timed Automata
F – Requirement: temporal logical formula, e.g.
– Invariant: something bad will never happen, something may happen
– Liveness: something will eventually happen

Specification of Requirements

• TCTL - Timed Computation Tree Logic

P: A

P’s computation tree:

• A \rightarrow C \rightarrow C \rightarrow C \rightarrow \ldots \text{ a path}
• (A,v) \rightarrow (C,v’) \rightarrow \ldots \text{ time = a timed path}
Quantifiers in TCTL

• **E** - exists a path ( ∃ ).
• **A** - for all paths ( ∀ ).
• **[]** - all states in a path ( □ or G).
• **<>** - some state in a path ( ◊ or F).

We shall look at the following combinations:
- **A[], A<>**, **E<>**, and **E[]**.

E<>p – “p Reachable”

• It is possible to reach a state in which p is satisfied.

• p is true in (at least) one reachable state.
A[]p – “Invariantly p”

- p holds invariantly.
- p is true in all reachable states.

A<>p – “Inevitable p”

- p will inevitable become true
  - the automaton is guaranteed to eventually reach a state in which p is true.
- p is true in some state of all paths.
E[ ] p – “Potentially Always p”

- p is potentially always true.

- There exists a path in which p is true in all states.

A[]( g imply A<> p )
A[]( g imply A<> p )

- g leads to p: whenever p is true, g will inevitably become true.

- In UPPAAL: g --> p

---

Bridge Problem

If possible find schedule for all four men to reach safe side in 60 min.
Model-checking of Real-Time Systems

- Modelling Formalism
- Algorithm + Datastructures
- Applications

Algorithms and Datastructures

- How to represent?
- How to analyse?
- Termination?
- Any ideas?
Datastructure: Zones

From infinite to finite

State
\((n, x=3.2, y=2.5)\)

Symbolic state (set)
\((n, 1\leq x\leq 4, 1\leq y\leq 3)\)

Zone:
conjunction of
\(x-y\leq n, x\leftrightarrow n\)

Symbolic Transitions

using Zones

\(n\)
\(x>3\)
\(a\)
\(y:=0\)
\(m\)

\(1\leq x\leq 4\)
\(1\leq y\leq 3\)
delays to

\(1\leq x, 1\leq y\)
\(-2\leq x-y\leq 3\)
conjuncts to

\(3<x, 1<y\)
\(-2\leq x-y\leq 3\)
projects to

\(3<x, y=0\)

Thus \((n, 1\leq x\leq 4, 1\leq y\leq 3) = a \Rightarrow (m, 3<x, y=0)\)
Zones = Conjuctive constraints

• A zone $Z$ is a conjunctive formula:
  $g_1 \& g_2 \& ... \& g_n$
  where $g_i$ is a clock constraint:
  $x_i \sim b_i$ or $x_i - x_j \sim b_{ij}$
• Use a zero-clock $x_0$ (constant 0)
• A zone can be re-written as a set:
  $\{x_i - x_j \sim b_{ij} | \sim \text{is } < \text{ or } \leq, \ i,j \leq n\}$
• This can be represented as a MATRIX, DBM (Difference Bound Matrices)

Operations on Zones

• Delay: $SP(Z)$ or $Z \uparrow$
  $\uparrow \{Z\} = \{u+d | d \in R, u \in \{Z\}\}$
• Weakest pre-condition: $WP(Z)$ or $Z \downarrow$ (the dual of $Z \uparrow$)
  $\downarrow \{Z\} = \{u | u+d \in \{Z\} \text{ for some } d \in R\}$
• Reset: $\{x\}Z$ or $Z(x:=0)$
  $\{x\}Z = \{u[0/x] | u \in \{Z\}\}$
• Conjunction
  $\{Z\&g\} = \{Z\} \cap \{g\}$
An important theorem on Zones

- The set of zones is closed under all constraint operations (including \( x:=x-c \) or \( x:=x+c \))
- That is, the result of the operations on a zone is a zone
- That is, there will be a zone (a finite object i.e a zone/constraints) to represent the sets: \([Z↑], [Z↓], \{x\}Z\)

One-step reachability: \( Si \rightarrow Sj \)

- Delay: \((n,Z) \rightarrow (n,Z')\) where \(Z' = Z↑ \land \text{inv}(n)\)
- Action: \((n,Z) \rightarrow (m,Z')\) where \(Z' = \{x\}(Z \land g)\) if \(g\):

\[
\begin{array}{ccc}
\text{n} & \text{g} & \text{x:=0} \\
\hline \\
\text{n} & \text{g} & \text{x=0} \\
\end{array}
\]

- Successors \((n,Z)=\{(m,Z') \mid (n,Z) \rightarrow (m,Z')\}, Z' \neq \emptyset\)
  - Sometime we write: \((n,Z)\rightarrow (m,Z')\) if \((m,Z')\) is a successor of \((n,Z)\)
Now, we have a search problem

\[(n_0, Z_0)\]

\[S_2, S_3 \ldots \ldots \ldots S_n\]

\[T_2\]

\[T_1\]

Init \(\rightarrow\) Final?

\[\text{INITIAL}\]

\[\text{Passed} := \emptyset;\]

\[\text{Waiting} := \{(n_0, Z_0)\}\]

\[\text{REPEAT}\]

- pick \((n, Z)\) in \text{Waiting}
- if for some \(Z' \supseteq Z\)
  \((n, Z')\) in \text{Passed} then STOP
- else /explore/ add 
  \{(m, U) : (n, Z) \Rightarrow (m, U)\}
  to \text{Waiting};
  Add \((n, Z)\) to \text{Passed}

\[\text{UNTIL}\]

\[\text{Waiting} = \emptyset\]

or

Final is in \text{Waiting}
Forward Reachability

Init -> Final ?

INITIAL Passed := Ø;
Waiting := \{(n0,Z0)\}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' \supseteq Z
(n,Z') in Passed then STOP
- else (explore) add
\{(m,U) : (n,Z) \Rightarrow (m,U)\}
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø or
Final is in Waiting

n,Z'

n,Z

Passed

Waiting

Init -> Final ?

INITIAL Passed := Ø;
Waiting := \{(n0,Z0)\}

REPEAT
- pick (n,Z) in Waiting
- if for some Z' \supseteq Z
(n,Z') in Passed then STOP
- else (explore) add
\{(m,U) : (n,Z) \Rightarrow (m,U)\}
to Waiting;
Add (n,Z) to Passed

UNTIL Waiting = Ø or
Final is in Waiting

n,Z'

n,Z

Passed

Waiting

Init
Forward Reachability

Init -> Final?

**INITIAL**
- Passed := Ø;
- Waiting := {{n0,Z0}}

**REPEAT**
- pick (n,Z) in Waiting
- if for some Z' \(\supseteq\) Z
  - (n,Z') in Passed then STOP
- else /explore/ add
  - \{ (m,U) : (n,Z) \rightarrow (m,U) \} to Waiting;
  - Add (n,Z) to Passed

UNTIL Waiting = Ø
or
Final is in Waiting
Issues

- Datastructures for Passed and Waiting
- Do we really need to *always* store in Passed?
- Which symbolic state to select from Waiting?
- Do we really need to add *all* successors?
- How to represent and manipulate zones?