Model-checking of Real-Time Systems

Cristina Seceleanu & Paul Pettersson
Embedded Systems, IDT, MDH

PROGRESS
A national Swedish Strategic Research Centre
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
  - Sensors
  - Actuators

- **Controller Program**
  - Discrete
  - Tasks
  - Model of tasks (automatic?)

- **UPPAAL Model**
  - Model of environment (user-supplied)
Model-Checking

**Model:** A

**Requirement Specification:** F

- **Invariant:** something bad will never happen,
something may happen
- **Liveness:** something will eventually happen

**UPPAAL**

A satisfies F

Yes!

No!

Diagnostic Information

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
Formal design and analysis

Modeling

Simulation

Verification
Example model-based verification

Max response time between reaching these two states?

Is this error state reachable?

Is this variable value always less than 64?

Is the system guaranteed to reach this state?

Is this component always operating in this state?
Model-checking of Real-Time Systems

- Modeling Formalism
- A Simple Example
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
  – Timed automata is another example (clocks)
    • Guards and resets over clock variables on transitions
    • Infinite state!

• Semantics: Transition Systems
Timed Automata  

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

- **Action**
  - Synchronization e.g. $a$
    (handshake: $a!$ for send, $a?$ for receive)

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

**Wanted Behaviour:**
- pressed once = light
- pressed twice quickly = light will get brighter
- pressed again = light off.
Timed Automata: Light Control with Timing

**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
$(n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)$

Delay Trans
$(n, x=2.4, y=3.1415) \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)$

Action used for synchronization

$x \leq 5 \& y > 3$

$x := 0$

Alur & Dill 1990
Timed Automata
with Invariants

Clocks: \( x, y \)

Transitions

\[
\begin{align*}
(n, x=2.4, y=3.1415) & \xrightarrow{e(3.2)} (n, x=2.4, y=3.1415) \\
(n, x=3.5, y=4.2415) & \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)
\end{align*}
\]

Location Invariants

\( x \leq 5 \)

\( x := 0 \)

\( y \leq 10 \)

Invariants

\( g_1 \)

\( g_2 \)

\( g_3 \)

\( g_4 \)

\textbf{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
Timed Automata: Example

\[ 2 \leq x \leq 3 \]

\[ X := 0 \]

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

\[ 2 \leq x \leq 3 \]

\[ X := 0 \]

\[ X := 0 \]

Diagram:
- Initial state: \( X := 0 \)
- Transition: \( X := 0 \)
- Invariant: \( 2 \leq x \leq 3 \)
Timed Automata: Example

\[ X := 0 \]

\[ X \leq 3 \]

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

\[ X \geq 2 \]

\[ X := 0 \]

\[ X \leq 3 \]

\[ X := 0 \]

\[ l \]

\[ X := 0 \]

\[ X \leq 3 \]

\[ X := 0 \]

\[ X \geq 2 \]

value of x

4

3

2

2 4 6 8 10

time

PROGRESS
Timed Automata: Example
(periodic task, period 20)

\[ T \]
\[ x \leq 20 \]
\[ x := 0 \]
\[ x = 20 \]
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 has elapsed since last “turn off”
- Light automatically switches off after 9 time units if it is not pressed.
Semantics Definition

- **Clock valuations**: \( V(C) \) \( v : C \rightarrow R \geq 0 \)

- **State**: \( (l, v) \) where \( l \in L \) and \( v \in V(C) \)

- **Action transition**: \( (l, v) \xrightarrow{a}(l', v') \) iff

  \[ g(v) \text{ and } v' = v[r] \text{ and } \text{Inv}(l')(v') \]

- **Delay transition**: \( (l, v) \xrightarrow{d}(l, v + d) \) iff

  \[ \text{Inv}(l)(v + d') \text{ whenever } d' \leq d \in R \geq 0 \]
Timed Automata: Example

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

Example transitions

(l1, m1, ..., x=2, y=3.5, i=3, ....) \(\mapsto\) (l2, m2, ..., x=0, y=3.5, i=7, ....)
Datastructure: Zones

From infinite to finite

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

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

Zone: conjunction of x-y<=n, x<=n

PROGRESS
Symbolic Transitions

Using Zones

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

• A zone \( Z \) is a conjunctive formula:
  \[ g_1 \& g_2 \& \ldots \& g_n \]
  where \( g_i \) is a clock constraint:
  \[ x_i \sim b_i \text{ or } x_i-x_j \sim b_{ij} \]

• Use a zero-clock \( x_0 \) (constant 0)

• A zone can be re-written as a set:
  \[ \{x_i-x_j \sim b_{ij} \mid \sim \text{ is } < \text{ or } \leq, \ i,j \leq n\} \]

• This can be represented as a MATRIX, DBM (Difference Bound Matrices)
Operations on Zones

• Delay: $SP(Z)$ or $Z^\uparrow$
  $\quad [Z^\uparrow] = \{u+d | d \in R, u \in [Z]\}$

• Weakest pre-condition: $WP(Z)$ or $Z^\downarrow$ (the dual of $Z^\uparrow$)
  $\quad [Z^\downarrow] = \{u | u+d \in [Z] \text{ for some } d \in R\}$

• Reset: $\{x\}Z$ or $Z(x:=0)$
  $\quad [{x}\{Z\}] = \{u[0/x] | u \in [Z]\}$

• Conjunction
  $\quad [Z\&g] = [Z] \cap [g]$
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: $[Z^\uparrow]$, $[Z^\downarrow]$, $[\{x\}Z]$
One-step reachability: \( S_i \rightarrow S_j \)

- **Delay**: \( (n, Z) \rightarrow (n, Z') \) where \( Z' = Z^\uparrow \land \text{inv}(n) \)

- **Action**: \( (n, Z) \rightarrow (m, Z') \) where \( Z' = \{x\}(Z \land g) \)

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

Stopable Area
[10,20]

[7,15]

Queue

Crossing
[3,5]

River

Gate
Train Crossing

Communication via channels and shared variable.

Stopable Area

Queue

Gate

[10,20]

[7,15]

[3,5]

app, stop
go

empty
nonempty
hd, add, rem
How to specify what to check

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

Model: A

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

UPPAAL

Yes!

No!

Diagnostic Information

A² F

A – Model: Network of Timed Automata
F – Requirement: temporal logical formula, e.g.
Specification of Requirements

• TCTL - Timed Computation Tree Logic

P: A → C → C → C → … a path

(A,v) → (C,v’) → …+ time = a timed path
Quantifiers in TCTL

- $E$ - exists a path ($\exists$).
- $A$ - for all paths ($\forall$).
- $[\ ]$ - all states in a path ($\Box$ or $G$).
- $<>$ - some state in a path ($\diamond$ or $F$).

We shall look at the following combinations:

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 inevitably 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 )
\( A[] (g \text{ imply } A \leftrightarrow p) \)

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

- In UPPAAL: \( g \rightarrow p \)
A Simple Example

- Uppaal uses a continuous time model.
- Concept of time:
  - a simple example that makes use of an observer.

First example with an observer.

(a) Test.  (b) Observer.  (c) Behaviour: one possible run.
A Simple Example (cont’d)

• Properties to be verified in Uppaal:
  
  • A [] Obs.taken imply x>=2
    all resets off x will happen when x is above 2
  
  • E<> Obs.idle and x>3
    this property requires, that it is possible to reach a state where Obs is in the location idle and x is bigger than 3.
A Simple Example: Invariant

- Add an invariant to the location loop, as shown in figure below
- The system is not allowed to stay in the state more than 3 time units, so that the transition has to be taken, and the clock reset in our example holds.

(a) Test.  (b) Updated behavior with an invariant.
A Simple Example (cont’d)

- Properties that hold in Uppaal:

  - $A[\text{Obs.taken}]$ imply $(x \geq 2 \text{ and } x \leq 3)$
    - shows that the transition is taken when $x$ is between 2 and 3, i.e., after a delay between 2 and 3.
  - $E<> \text{Obs.idle and } x > 2$
    - it is possible to take the transition when $x$ is between 2 and 3. The upper bound 3 is checked with the next property.
  - $A[\text{Obs.idle}]$ imply $x \leq 3$
    - to show that the upper bound is respected.

The former property $E<> \text{Obs.idle and } x > 3$ no longer holds.
References

1. A Tutorial on UPPAAL 4.0
   (Gerd Behrmann, Alexandre David, and Kim G. Larsen)
   
   http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

2. UPPAAL Tool:
   
   http://www.uppaal.org/