Model Checking in UPPAAL

Giovanni Battaglia

Pisa C.S. Dep.

April, 2009
Integrated tool environment for modeling, validation and verification of real-time systems.

Systems modeled as network of Timed Automata (TA) enriched with:
- Bounded integer.
- Arrays and structures.
- Urgency.

The query language is a subset of the Computational Tree Logic (CTL) that doesn’t allow nesting of path formulae.
Outline

1. The Modeling Language
2. The Query Language
3. Putting the ingredients together: the Train-Gate example
Outline

1. The Modeling Language
2. The Query Language
3. Putting the ingredients together: the Train-Gate example
Timed Automaton (TA)

Definition (Timed Automaton)
Is a tuple $\langle L, l_0, C, A, E, I \rangle$ where
- $L$ is a finite set of locations.
- $l_0 \in L$ is the initial location.
- $C$ is the set of clocks.

Simple Lamp Process

[Diagram of a Timed Automaton with three states: Off, Low, Bright. Edges with conditions and actions like press?, y=0, y>=5 press?, y<5 press?, counter++.]
Timed Automaton (TA)

Definition (Timed Automaton)

Is a tuple $\langle L, l_0, C, A, E, I \rangle$ where

- $A$ is a set of actions, co-actions and the internal $\tau$-action.
- $E \subseteq L \times A \times B(C) \times 2^C \times L$ is set of edges between locations with an action, a guard and a set of clocks to reset.

Simple Lamp Process

![Diagram of a simple lamp process with transitions and actions.]

- Off
  - $y = 0$ press?
  - $y \geq 5$ press?
  - $\text{counter++}$
  - $\text{counter++}$

- Low
  - $y < 5$ press?

- Bright
  - $\text{counter++}$
  - $\text{press?}$
  - $\text{press?}$
Timed Automaton (TA)

Definition (Timed Automaton)
Is a tuple $\langle L, l_0, C, A, E, I \rangle$ where
- $I : L \rightarrow B(C)$ assigns invariants to locations.

Simple Lamp Process

```
Off
  y>5 press?
  counter++
  y=0
  press?
  y<5 press?
  counter++

Low
  y<5 press?

Bright
```
The Modeling Language

Semantics of TA

Definition (Semantics of TA)
The semantics of $\langle L, l_0, C, A, E, I \rangle$ is an LTS $\langle S, s_0, \rightarrow \rangle$ where:
- $S \subseteq L \times \mathbb{R}^C$ is the set of states.
- $s_0 = (l_0, u_0)$ is the initial state.
- $\rightarrow \subseteq S \times \{\mathbb{R}_{\geq 0} \cup A\} \times S$ is the transition relation s.t.

Simple Lamp Process

```
Simple Lamp Process

Bright

Low

Off

y>=5 press?

y=0

counter++

y<5 press?

Giovanni Battaglia (Pisa C.S. Dep.)
Model Checking in UPPAAL
April, 2009 6 / 28
```
Semantics of TA

Definition (Semantics of TA)

- \((l, u) \rightarrow^d (l, u + d)\) if \(\forall d' \in \mathbb{R}_{\geq 0}: 0 \leq d' \leq d \Rightarrow u + d' \in I(l)\)
- \((l, u) \rightarrow^a (l', u')\) if \(\exists l \xrightarrow{a, g, r} l'\) s.t. \(u \in g\), \(u' = [r \mapsto 0]u\), \(u' \in I(l)\)

Simple Lamp Process

[Diagram of a simple lamp process with states Off, Low, and Bright, transitions triggered by 'press?', 'y=0', 'y>=5', 'y<5', and 'counter++'.]
Semantics of a network of TA

Definition (Semantics of a network of TA)

Let $A_i = \langle L_i, l_0^i, C, A, E_i, l_i \rangle$ a network of $n$ timed automata and $\overrightarrow{l_0} = \langle l_0^1, \ldots, l_0^n \rangle$ the initial location vector, the semantics is an unlabeled transition system $\langle S, s_0, \rightarrow \rangle$ where

- $S = \langle L_1 \times \ldots \times L_n \rangle \times \mathbb{R}^C$ is the set of states.
- $s_0 = (\overrightarrow{l_0}, u_0)$ is the initial state.
- $\rightarrow \subseteq S \times S$ is the transition relation defined by:

![Diagram](attachment:image.png)

- Off
- Low
- Bright
- Idle

- $y=0$ press?
- $y \geq 5$ press?
- $y < 5$ press?
- $\text{counter}++$
- press?
- $\text{counter}++$
- press!

- Idle

Giovanni Battaglia (Pisa C.S. Dep.)  Model Checking in UPPAAL  April, 2009 7 / 28
Definition (Semantics of a network of TA)

\( (l, u) \rightarrow (l, u + d) \) if \( \forall d' \in \mathbb{R}_{\geq 0} : 0 \leq d' \leq d \Rightarrow u + d' \in I(l) \)
Semantics of a network of TA

Definition (Semantics of a network of TA)

\[(\overrightarrow{l}, u) \rightarrow (\overrightarrow{l}[l'_i/l_i], u') \quad \text{if} \quad \exists l_i \xrightarrow{\tau \subseteq g} l'_i \text{ s.t. } u \in g, u' = [r \mapsto 0]u, u' \in l(\overrightarrow{l})\]
The Modeling Language

Semantics of a network of TA

Definition (Semantics of a network of TA)

\[ (\overrightarrow{l}, u) \rightarrow (\overrightarrow{l}[l'_j/l_j, l'_i/l_i], u') \] if

\[ \exists l_i \vdash c? g_i r_i \rightarrow l'_i, l_j \vdash c! g_j r_j \rightarrow l'_j \] s.t.

\[ u \in g_i \land g_j, u' = [r_i \cup r_j \rightarrow 0]u, u' \in l(\overrightarrow{l}) \]
The Modeling Language

Invariants and Guards

\[ P \parallel \text{Observer} \]

- Start
  - \( x \geq 2 \)
  - \( \text{reset!} \)

- Idle
  - \( \text{reset?} \)
  - \( x = 0 \)

- Taken

Giovanni Battaglia (Pisa C.S. Dep.)
Model Checking in UPPAAL
April, 2009
Add the constraint: “P is not allowed to stay in the “Start” location for more than 3 time units”.
Add the constraint: “P is not allowed to stay in the “Start” location for more than 3 time units”.
Invariants and Guards

\[ P \parallel \text{Observer (correct)} \]

Add the constraint: “\( P \) is not allowed to stay in the “Start” location for more than 3 time units”. 

\[ \text{Start} \]

\[ \text{Idle} \]

\[ \text{Taken} \]

x \leq 3

x \geq 2

reset!

reset?

x=0
Invariants: the semantics

\[ P \parallel \text{Observer (correct)} \]

\[ (\vec{l}, u) \rightarrow (\vec{l}, u + d) \text{ if } \forall d' \in \mathbb{R}_{\geq 0} : 0 \leq d' \leq d \Rightarrow u + d' \in I(\vec{l}) \]
**Invariants : the semantics**

\[ P \parallel \text{Observer (correct)} \]

\[
\begin{align*}
(\vec{l}, u) &\rightarrow (\vec{l}[l_i/l_i], u') \quad \text{if} \\
\exists \ l_i \quad &\tau \overset{g \ r}{\longrightarrow} \ l'_i \quad \text{s.t.} \\
&u \in g, \ u' = [r \mapsto 0]u, \ u' \in l(\vec{l})
\end{align*}
\]
Invariants: the semantics

\[ P \parallel \text{Observer (correct)} \]

\[ (\overrightarrow{f}, u) \rightarrow (\overrightarrow{f}[l'/l_j, l'_i/l_i], u') \quad \text{if} \]
\[ \exists \ l_i \rightarrow c?g_i r_i \quad l'_i, \quad l_j \rightarrow c!g_j r_j \quad l'_j \quad \text{s.t.} \]
\[ u \in g_i \land g_j, \quad u' = [r_i \cup r_j \mapsto 0]u, \quad u' \in l(\overrightarrow{f}) \]
Urgent and Committed States

Definition (Urgent State of the System)

A state is urgent if any of the processes is in an urgent location.

- An urgent state has no delay transition (i.e. time may not progress in an urgent state).
Urgent and Committed States

Definition (Urgent State of the System)

A state is urgent if any of the processes is in an urgent location.

- An urgent state has no delay transition (i.e. time may not progress in an urgent state).

Locations vs Urgent Locations
Urgent and Committed States

Definition (Urgent State of the System)

A state is urgent if any of the processes is in an urgent location.

- An urgent state has no delay transition (i.e. time may not progress in an urgent state).

Locations vs Urgent Locations
Urgent and Committed States

Definition (Urgent State of the System)
A state is urgent if any of the processes is in an urgent location.

- An urgent state has no delay transition (i.e. time may not progress in an urgent state).
- It is semantically equivalent to a location with incoming edges resetting a clock \( x \), labeled with the invariant \( x \leq 0 \) (i.e. the process cannot wait in that location).

Locations vs Urgent Locations

```
S0 -- S1
```

```
S0
\( x \leq 0 \) -- S1
```
Urgent and Committed States

Definition (Committed State of the System)

A state is committed if any of the processes is in a committed location.

- It cannot delay.
- The next transition **must** involve at least one outgoing edge of one of the committed locations.

Urgent Locations vs Committed Locations
Urgent channels

Definition (Urgent channel)

Urgent channel are similar to regular channel, except that it is not possible to delay in the source state if it is possible to trigger a synchronization over an urgent channel.

\[
\text{P || Observer}
\]

\[
\text{Start} \quad \text{reset!}
\]

\[
\text{Idle} \quad \text{reset?}
\]

\[
\text{Taken} \quad x=0
\]
Outline

1. The Modeling Language
2. The Query Language
3. Putting the ingredients together: the Train-Gate example
The main purpose of a model checker is to verify the model w.r.t. a requirement specification.

UPPAAL uses a simplified version of Computational Tree Logic.

State formulae describe properties of an individual state: $y<3$, deadlock.

Path formulae describe properties of traces of the model: $A[] y>=0$ and $counter<100$
State Property: deadlock

- A deadlock state is a state in which no action transition will ever be enabled again.
- \((l, u)\) is a deadlock state iff \(\forall d \geq 0, a \in A : (l, u + d) \not\xrightarrow{a}\)
- To check if the system is deadlock free
  \[ A[] \text{ not deadlock} \]
Reachability Properties

$E<> \psi$

$E<> \psi$ evaluates to true if and only if there exists at least one state satisfying $\psi$, that is reachable from the initial state.
Safety Properties

$\text{A}[] \ \psi$

$\text{A}[] \ \psi$ evaluates to true if and only if every reachable state satisfy $\psi$. 
A<> Ψ evaluates to true if each path eventually reaches a state satisfying Ψ.
Liveness Properties (2)

\[ \psi \rightarrow \phi \]

\[ \psi \rightarrow \phi \] evaluates to true if whenever \( \psi \) holds eventually \( \phi \) will hold as well.
Liveness Properties (3) : bounded liveness

\[ \Psi \rightarrow_{\leq t} \Phi \]

\( \Psi \rightarrow_{\leq t} \Phi \) evaluates to true if whenever \( \Psi \) becomes true, then \( \Phi \) becomes true within \( t \) time units.
Liveness Properties (3) : bounded liveness

\[ \Psi \rightarrow \leq_t \Phi \] evaluates to true if whenever \( \Psi \) becomes true, then \( \Phi \) becomes true within \( t \) time units.

- There is not a dedicated operator in UPPAAL.
- We can reduce \( \Psi \rightarrow \leq_t \Phi \) to an unbounded liveness property.
- Add a new clock \( x \) and reset it whenever \( \Psi \) becomes true.
- Check \( \Psi \rightarrow (\Phi \text{ and } x \leq t) \).

![Diagram](image-url)
Outline

1. The Modeling Language
2. The Query Language
3. Putting the ingredients together: the Train-Gate example
Train-Gate: description

- Railway control system which controls access to a bridge for several trains.
- A train can not be stopped instantly and restarting also takes time.
- After sending the `appr` signal each train has 10 time units to stop safely.
Train-Gate: description

- **Safety**: no more than one train crossing the bridge at the same time.
- **No starvation**: whenever train approaches the bridge, it will eventually cross.
- **Deadlock free.**
Train-Gate: the idea

- If a train is approaching and the gate is not being used, the train is allowed to cross.
- Otherwise the train is stopped and the request stored in the queue of pending requests.
Train-Gate: communication channels

- **chan appr[N]** used by trains to signal gate approaching.
- **chan stop[N]** used by gate to stop an approaching train.
- **chan leave[N]** used by trains to release the shared resource.
- **urgent chan go[N]** used by gate to restart a previously stopped train.
Train-Gate: the train

- Safe
  - appr[0]!
  - x=0

- Appr
  - x<=20
  - x<=10
  - stop[0]?

- Stop
  - x=0

- Start
  - x<=15
  - go[0]?

- Cross
  - x<=5
  - x>=3
  - leave[0]!

- x=0
  - x>=7
  - x=0

Putting the ingredients together: the Train-Gate example
Train-Gate: the gate

len > 0
go[front()]!

len == 0
e : id_t
appr[e]?
enqueue(e)

e : id_t
e == front()
leave[e]?
dequeue()

stop[tail()]!

e : id_t
appr[e]?
enqueue(e)

Occ

Free

C
Conclusions

**PROS**
- Graphical descriptions of systems.
- Diagnostic trace that explains why a property is not satisfied.

**CONS**
- Lot of cares with urgent/committed locations.
- Messages are not that informative:
  “The successors of this state are not well defined. This is most likely due to a range error, index error or division by zero in a guard, synchronization, update or invariant in one or more outgoing edges”.
- I made mistakes just modifying an existing model !!