Symbolic Real Time Model Checking

Kim G Larsen
Overview

- Timed Automata – Decidability Results
- The UPPAAL Verification Engine
  - Datastructures for zones
  - Liveness Checking Algorithm
- Abstraction and Compositionality
- Further Optimizations
Timed Automata – Decidability Results
Decidability?

OBSTACLE:
Uncountably infinite state space
Derived Relations and Reachability

\[(l, u) \xrightarrow{\delta} (l', u') \quad \text{iff} \quad \exists d > 0. (l, u) \xrightarrow{\epsilon(d)} (l', u').\]

\[(l, u) \xrightarrow{\alpha} (l', u') \quad \text{iff} \quad \exists a \in \text{Act}. (l, u) \xrightarrow{\alpha} (l', u').\]

\[(l, u) \xrightarrow{\sim} (l', u') \quad \text{iff} \quad (l, u)(\xrightarrow{\delta} \cup \xrightarrow{\alpha})* (l', u').\]

Definition

The set of reachable locations, \(\text{Reach}(A)\), of a timed automaton \(A\) is defined as:

\[l \in \text{Reach}(A) \equiv^\Delta \exists u. (l_0, u_0) \xrightarrow{\sim} (l, u)\]
Time Abstracted Bisimulation

Definition
Let $G \subseteq L$ be a set of goal locations. An equivalence relation $R$ on $L \times \mathcal{R}^G$ is a TAB wrt $G$ if whenever $(l, u)R(n, v)$ the following holds:

1. $l \in G$ iff $n \in G$,
2. whenever $(l, u) \xrightarrow{\delta} (l', u')$ then $(n, v) \xrightarrow{\delta} (n', v')$ with $(l', u')R(n', v')$
3. whenever $(l, u) \xrightarrow{\alpha} (l', u')$ then $(n, v) \xrightarrow{\alpha} (n', v')$ with $(l', u')R(n', v')$
Stable Quotient

Definition
Let $R$ be a TAB wrt $G$. The induced quotient has classes of $R$, $\pi \in (L \times \mathbb{R}C/R)$, as states. For classes $\pi, \pi'$ the transitions are

- $\pi \xrightarrow{\delta} \pi'$ iff $(l, u) \xrightarrow{\delta} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.
- $\pi \xrightarrow{a} \pi'$ iff $(l, u) \xrightarrow{a} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.

Theorem
Let $R$ be TAB wrt $G$. Then, a location from $G$ is reachable iff there exists an equivalence class $\pi$ of $R$ such that $\pi$ is reachable in the quotient and $\pi$ contains a state whose location is in $G$. 
Stable Quotient

Partitioning

Reachable?

\[ y := 0 \quad a \quad x := 0 \quad b \quad y \leq 2 \]

\[ x \leq 2 \quad y \leq 2, \quad x = 4 \quad c \]

\[ y \]

\[ x \]
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning

Reachable?

y:=0
a
y<=2
L0
x:=0
b
x<=2
c
y<=2, x=4
L1

y

x

y

x

BRICS
Basic Research in Computer Science

CISS
CENTER FOR INNOVATIVE SOFTWARE SYSTEMS
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Every TA has a finite TAB quotient (region-constr.)
Regions
Finite Partitioning of State Space

For each clock \( x \) let \( c_x \) be the largest integer with which \( x \) is compared in any guard or invariant of \( A \). \( u \) and \( u' \) are region equivalent, \( u \equiv u' \) iff the following holds:

1. For all \( x \in C \), either \( \lfloor u(x) \rfloor = \lfloor u'(x) \rfloor \) or \( u(x), u'(x) > c_x \);

2. For all \( x, y \in C \) with \( u(x) \leq c_x \) and \( u(y) \leq c_y \),
   \[ \text{fr}(u(x)) \leq \text{fr}(u(y)) \iff \text{fr}(u'(x)) \leq \text{fr}(u'(y)); \]

3. For all \( x \in C \) with \( u(x) \leq c_x \),
   \[ \text{fr}(u(x)) = 0 \iff \text{fr}(u'(x)) = 0. \]

An equivalence class (i.e. a region) in fact there is only a finite number of regions!!
Fundamental Results

- **Reachability** ☺ Alur, Dill
- **Trace-inclusion** Alur, Dill
  - Timed ☹; Untimed ☺
- **Bisimulation**
  - Timed ☻ Cerans; Untimed ☺
- **Model-checking** ☻
  - TCTL, $T_{\mu}$, $L_{\nu}$,...
Updatable Timed Automata

<table>
<thead>
<tr>
<th></th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := c, x := y$</td>
<td>Pspace complete</td>
<td>Pspace complete</td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x := y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x := x - 1$</td>
<td>Undecidable</td>
<td></td>
</tr>
<tr>
<td>$x &lt; c, x \leq c$</td>
<td>Pspace complete</td>
<td>Pspace complete</td>
</tr>
<tr>
<td>$x &gt; c, x \geq c$</td>
<td></td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(y+)c &lt;: x &lt;: (y+)d$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y + c &lt;: x &lt;: z + d$</td>
<td>Undecidable</td>
<td></td>
</tr>
</tbody>
</table>

With $\sim \in \{<, \leq, \geq, >\}$ and $c, d \in \mathbb{Q}_+$

<table>
<thead>
<tr>
<th></th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$:= c, x := y$</td>
<td></td>
<td>TA-bisimilar</td>
</tr>
<tr>
<td>$x := x + 1$</td>
<td>TA-bisimilar</td>
<td>Turing</td>
</tr>
<tr>
<td>$x := y + c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x &lt; c, x \leq c$</td>
<td></td>
<td>TA$_{\varepsilon}$</td>
</tr>
<tr>
<td>$x &gt; c, x \geq c$</td>
<td></td>
<td>TA$_{\varepsilon}$</td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td></td>
<td>Turing</td>
</tr>
<tr>
<td>$(y+)c &lt;: x &lt;: (y+)d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With $\sim \in \{<, \leq, \geq, >\}$ and $c, d \in \mathbb{Q}_+$
The UPPAAL Verification Engine
Overview

- Zones and DBMs
- Minimal Constraint Form
- Clock Difference Diagrams

- Distributed UPPAAL  [CAV2000, STTT2004]
- Unification & Sharing  [FTRTFT2002, SPIN2003]
- Acceleration  [FORMATS2002]
- Static Guard Analysis  [TACAS2003, TACAS2004]
- Storage-Strategies  [CAV2003]
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=\leq n, x=\Rightarrow n
Symbolic Transitions

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

- A zone $Z$ is a conjunctive formula:

$$g_1 \& g_2 \& \ldots \& g_n$$

where $g_i$ is a clock constraint $x_i \sim b_i$ 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

- Future delay $Z^\uparrow$:
  \[ [Z^\uparrow] = \{u+d | d \in \mathbb{R}, u \in [Z] \} \]

- Past delay $Z^\downarrow$:
  \[ [Z^\downarrow] = \{u | u+d \in [Z] \text{ for some } d \in \mathbb{R} \} \]

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

- Conjunction
  \[ [Z \& g] = [Z] \cap [g] \]
THEOREM

- The set of zones is closed under all constraint operations.

- 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: \([Z^\uparrow], [Z^\downarrow], [{x}Z], [Z\&g]\).
Symbolic Exploration

reachable?
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?

y := 0
y <= 2
x := 0
y <= 2
x <= 2
y <= 2, x >= 4

Left
Symbolic Exploration

Reachable?

Left
Symbolic Exploration

Reachable?

Delay
Symbolic Exploration

\begin{itemize}
  \item $y := 0$
  \item $y \leq 2$
  \item $x := 0$
  \item $x \leq 2$
  \item $y \leq 2, x = 4$
  \item Reachable?
\end{itemize}

Left
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?

Down
Forward Reachability

\[ \text{Init} \rightarrow \text{Final} \]

\[
\begin{align*}
\text{INITIAL} & \quad \text{Passed} := \emptyset; \\
& \quad \text{Waiting} := \{(n_0, Z_0)\}
\end{align*}
\]

\[
\text{REPEAT}
\]

\[
\begin{align*}
\text{UNTIL} & \quad \text{Waiting} = \emptyset \\
& \quad \text{or} \\
& \quad \text{Final} \text{ is in } \text{Waiting}
\end{align*}
\]
Forward Reachability

\[ \text{Init} \rightarrow \text{Final} \]

\[ \text{INITIAL} \quad \text{Passed} := \emptyset; \]
\[ \quad \text{Waiting} := \{(n0,Z0)\} \]

\[ \text{REPEAT} \]
- pick \((n,Z)\) in \(\text{Waiting}\)
- if for some \(Z' \supseteq Z\)
  \((n,Z')\) in \(\text{Passed}\) then STOP

\[ \text{UNTIL} \quad \text{Waiting} = \emptyset \]
  or
  \quad \text{Final is in Waiting} \]
Forward Reachability

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

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

UNTIL Waiting = Ø
or
Final is in Waiting

Init -> Final ?
Forward Reachability

$\text{INITIAL}$  $\text{Passed} := \emptyset$;
$\text{Waiting} := \{ (n0,Z0) \}$

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

UNTIL  $\text{Waiting} = \emptyset$
  or
  Final is in $\text{Waiting}$
Canonical Datastructures for Zones

*Difference Bounded Matrices*

Bellman 1958, Dill 1989

**Inclusion**

| D1 | x \leq 1  
|    | y-x \leq 2  
|    | z-y \leq 2  
|    | z \leq 9   
|    | D1 Graph |

| D2 | x \leq 2  
|    | y-x \leq 3  
|    | y \leq 3   
|    | z-y \leq 3  
|    | z \leq 7   
|    | D2 Graph |
Canonical Datastructures for Zones

Difference Bounded Matrices

Inclusion

D1

\[
\begin{align*}
x & \leq 1 \\
y - x & \leq 2 \\
z - y & \leq 2 \\
z & \leq 9
\end{align*}
\]

Graph

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9
\end{array}
\]

Shortest Path Closure

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9
\end{array}
\]

D2

\[
\begin{align*}
x & \leq 2 \\
y - x & \leq 3 \\
y & \leq 3 \\
z - y & \leq 3 \\
z & \leq 7
\end{align*}
\]

Graph

\[
\begin{array}{c}
0 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9
\end{array}
\]

Shortest Path Closure

\[
\begin{array}{c}
0 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9
\end{array}
\]
Canonical Datastructures for Zones

Difference Bounded Matrices

Emptiness

\[
\begin{align*}
  &x \leq 1 \\
  &y \geq 5 \\
  &y-x \leq 3
\end{align*}
\]

Graph

Negative Cycle
iff
empty solution set
Canonical Datastructures for Zones

Difference Bounded Matrices
Canonical Datastructures for Zones

Difference Bounded Matrices

Remove all bounds involving $y$ and set $y$ to 0
Canonical Datastructures for Zones

Difference Bounded Matrices

Shortest Path Closure $O(n^3)$
Canonical Datastructures for Zones

Minimal Constraint Form

\[ x_1 - x_2 \leq 4 \]
\[ x_2 - x_1 \leq 10 \]
\[ x_3 - x_1 \leq 2 \]
\[ x_2 - x_3 \leq 2 \]
\[ x_0 - x_1 \leq 3 \]
\[ x_3 - x_0 \leq 5 \]

Shortest Path Closure \( O(n^3) \)

Shortest Path Reduction \( O(n^3) \)

Space worst \( O(n^2) \) practice \( O(n) \)

RTSS 1997
SPACE PERFORMANCE

- Audio
- Audio w Col
- B&O
- Box Sorter
- M. Plant
- Fischer 2
- Fischer 3
- Fischer 4
- Fischer 5
- Train Crossing

- Minimal Constraint
- Global Reduction
- Combination
Shortest Path Reduction
1st attempt

Idea

An edge is REDUNDANT if there exists an alternative path of no greater weight.
THUS, remove all redundant edges!

Problem

\( v \) and \( w \) are both redundant.
Removal of one depends on presence of other.

Observation: If no zero- or negative cycles then SAFE to remove all redundancies.
Shortest Path Reduction Solution

G: weighted graph
Shortest Path Reduction

Solution

1. Equivalence classes based on 0-cycles.

G: weighted graph
Shortest Path Reduction

Solution

1. Equivalence classes based on 0-cycles.

2. Graph based on representatives.
   Safe to remove redundant edges
Shortest Path Reduction
Solution

1. Equivalence classes based on 0-cycles.

2. Graph based on representatives.
Safe to remove redundant edges

3. Shortest Path Reduction
   =
   One cycle pr. class
   +
   Removal of redundant edges between classes

G: weighted graph

Canonical given order of clocks
Earlier Termination

\[
\text{INITIAL } \text{Passed} := \emptyset; \\
\text{Waiting} := \{(n0,Z0)\}
\]

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

\[
\text{UNTIL } \text{Waiting} = \emptyset \\
or \\
\text{Final is in Waiting}
\]
Earlier Termination

Init -> Final?

**INITIAL**

Passed := Ø;
Waiting := {(n0,Z0)}

**REPEAT**

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

**UNTIL**

Waiting = Ø or Final is in Waiting
Earlier Termination

\[ \text{INITIAL } \text{Passed} := \emptyset; \]
\[ \text{Waiting} := \{(n0, Z0)\} \]

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

\[ \text{UNTIL } \text{Waiting} = \emptyset \]
\[ \text{or} \]
\[ \text{Final is in Waiting} \]
Clock Difference Diagrams
= Binary Decision Diagrams + Difference Bounded Matrices

- Nodes labeled with differences
- Maximal sharing of substructures (also across different CDDs)
- Maximal intervals
- Linear-time algorithms for set-theoretic operations.

- NDD’s Maler et. al
- DDD’s Møller, Lichtenberg
TIME PERFORMANCE

Percent

CDD
Reduced CDD
CDD+BDD

Philips
Philips col
B&O
BRP
PowerDown1
PowerDown2
Dacapo
GearBox
Fischer4
Fischer5
UPPAAL 1995 - 2001

Every 9 month
10 times better performance!
Liveness Properties

\[ F ::= E\Box P \mid P \]

\[ A\Diamond P \mid P \Rightarrow Q \]

**Possibly always** \( P \)

**Eventually** \( P \)

is equivalent to \((\neg E\Box \neg P)\)

**P leads to** \( Q \)

is equivalent to

\[ A\Diamond (P \Rightarrow A\Diamond Q) \]

Bouajjani, Tripakis, Yovine’97

On-the-fly symbolic model checking of TCTL
proc Liveness(s₀, φ, Passed) ≡
  pre(s₀ = delay(s₀))
  pre(s₀ ⊨ φ)
  pre(¬unbounded(s₀) ∧ ¬deadlocked(s₀))
  pre(∀s ∈ Passed. s ⊨ A◊¬φ)
  WS := {s₀};
  ST := ∅;
  while WS ≠ ∅ do
    s := pop(WS);
    while top(ST) ≠ parent(s) do
      Passed := Passed ∪ {pop(ST)};
    od
    push(ST, s);
    if ∀s' ∈ Passed. s ≈ s'
      then foreach t: s ≈ t do
        if t ⊨ φ then t := delay(t);
          if unbounded(t) then exit(true) fi
          if deadlocked(t) then exit(true) fi
          if ∃t' ∈ ST.t = t' then exit(true) fi
        fi
      od
      exit(false);
  end
**Liveness**

\[ E[\phi] \quad (A\Diamond \neg \phi) \]

---

```plaintext
proc Liveness(s_0, \phi, Passed) \equiv
pre(s_0 = delay(s_0))
pre(s_0 \models \phi)
pre(\neg unbounded s_0 \land \neg deadlocked(s_0))
pre(\forall s \in Passed. s \models A\Diamond \neg \phi)
WS := \{s_0\};
ST := \emptyset;
while WS \neq \emptyset do
  s := pop(WS);
  while top(ST) \neq parent(s) do
    Passed := Passed \cup \{pop(ST)\};
    od
  push(ST, s);
  if \forall s' \in Passed. s \nsubseteq s'
    then foreach t : s \Rightarrow t do
      if t \models \phi then t := delay(t);
        if unbounded(t) then exit(t);
        if deadlocked(t) then exit(t);
        if \exists t' \in ST. t = t' then exit(t);
        push(WS, t);
      fi
    od
  fi
  od
  exit(false);
end
```
proc Liveness($s_0, \varphi, \text{Passed}) \equiv$
\begin{align*}
pre &\left(s_0 = \text{delay}(s_0)\right) \\
pre &\left(s_0 \models \varphi\right) \\
pre &\left(\neg\text{unbounded}_s \land \neg\text{deadlocked}(s_0)\right) \\
pre &\left(\forall s \in \text{Passed}. \ s \models A\Diamond \neg \varphi\right) \\
\text{WS} &:= \{s_0\}; \\
\text{ST} &:= \emptyset; \\
\text{while} &\ WS \neq \emptyset \ \text{do} \\
\quad &\ s := \text{pop}(\text{WS}); \\
\quad &\ \text{while} \ \text{top}(\text{ST}) \neq \text{parent}(s) \ \text{do} \\
\quad &\quad \ \text{Passed} := \text{Passed} \cup \{\text{pop}(\text{ST})\}; \\
\quad &\ \text{od} \\
\quad &\ \text{push}(\text{ST}, s); \\
\quad &\ \text{if} \ \forall s' \in \text{Passed}. \ s \not\subseteq s' \\
\quad &\ \ \ \text{then} \ \text{foreach} \ t : \ s \not\rightarrow t \ \text{do} \\
\quad &\quad &\ \text{if} \ t \models \varphi \ \text{then} \ t := \text{delay}(t); \\
\quad &\quad &\ \text{if} \ \text{unbounded}(t) \ \text{then} \ \text{exit}(\text{true}); \\
\quad &\quad &\ \text{if} \ \text{deadlocked}(t) \ \text{then} \ \text{exit}(\text{true}); \\
\quad &\quad &\ \text{if} \ \exists t' \in \text{ST}. \ t = t' \ \text{then} \ \text{exit}(\text{true}); \\
\quad &\quad &\ \text{push}(\text{WS}, t); \\
\quad &\ \text{od} \\
\quad &\ \text{od} \\
\text{od} \\
\text{exit}(\text{false}); \\
\end{align*}
\textbf{proc Liveness}(s_0, \varphi, \text{Passed}) \equiv \\
\text{pre}(s_0 = \text{delay}(s_0)) \\
\text{pre}(s_0 \models \varphi) \\
\text{pre}(\neg \text{unbounded}(s_0) \land \neg \text{deadlocked}(s_0)) \\
\text{pre}(\forall s \in \text{Passed. } s \models A \Box \neg \varphi) \\
\text{WS} := \{s_0\}; \\
\text{ST} := \emptyset; \\
\textbf{while } \text{WS} \neq \emptyset \textbf{ do} \\
\quad s := \text{pop(WS)}; \\
\quad \textbf{while } \text{top(ST)} \neq \text{parent}(s) \textbf{ do} \\
\quad\quad \text{Passed} := \text{Passed} \cup \{\text{pop(ST)}\}; \\
\quad\quad \text{od} \\
\quad \text{push(ST, s);} \\
\quad \text{if } \forall s' \in \text{Passed. } s \not\subseteq s' \\
\quad\quad \textbf{then } \textbf{foreach } t: s \xrightarrow{a} t \textbf{ do} \\
\quad\quad\quad \text{if } t \models \varphi \textbf{ then } t := \text{delay}(t); \\
\quad\quad\quad\quad \text{if } \text{unbounded}(t) \textbf{ then } \text{exit(tr)}; \\
\quad\quad\quad\quad \text{if } \text{deadlocked}(t) \textbf{ then } \text{exit(tr)}; \\
\quad\quad\quad\quad \text{if } \exists t' \in \text{ST. } t = t' \textbf{ then } \text{exit(tr)}; \\
\quad\quad\quad \text{push(WS, t);} \\
\quad\quad \text{fi} \\
\quad \text{fi} \\
\quad \text{od} \\
\textbf{od} \\
\text{exit}(\text{false}); \\
\text{end}
proc Liveness(s₀, φ, Passed) ≡
pre(s₀ = delay(s₀))
pre(s₀ ⊨ φ)
pre(¬unbounded s₀ ∧ ¬deadlocked(s₀))
pre(∀s ∈ Passed. s ⊨ A◊¬φ)
WS := {s₀};
ST := ∅;
while WS ≠ ∅ do
  s := pop(WS);
  while top(ST) ≠ parent(s) do
    Passed := Passed ∪ {pop(ST)};
  od
  push(ST, s);
  if ∀s' ∈ Passed. s ⊈ s'
    then foreach t : s ↠ t do
      if t ⊨ φ then t := delay(t);
        if unbounded(t) then exit(t);
        if deadlocked(t) then exit(t);
        if ∃t' ∈ ST. t = t' then exit(t);
      fi
    od
  fi
od
exit(false);
end
proc Liveness(s₀, φ, Passed) ⇔
pre(s₀ = delay(s₀))
pre(s₀ ⊨ φ)
pre(¬unbounded_s₀ ∧ ¬deadlocked(s₀))
pre(∀s ∈ Passed. s ⊨ A♢¬φ)
WS := {s₀};
ST := ∅;
while WS ≠ ∅ do
    s := pop(WS);
    while top(ST) ≠ parent(s) do
        Passed := Passed ∪ {pop(ST)};
    od
    push(ST, s);
    if ∀s' ∈ Passed. s ⊈ s'
        then foreach t : s ⊨ t do
            if t ⊨ φ then t := delay(t);
            if unbounded(t) then exit(true);
            if deadlocked(t) then exit(true);
            if ∃t' ∈ ST. t = t' then exit(true);
            push(WS, t);
        fi
    fi
    od
exit(false);
proc Liveness(s₀, φ, Passed) ≡
pre(s₀ = delay(s₀))
pre(s₀ ⊨ φ)
pre(¬unbounded s₀ ∧ ¬deadlocked(s₀))
pre(∀s ∈ Passed. s ⊨ A¬φ)
WS := \{s₀\};
ST := ∅;
while WS \neq ∅ do
    s := pop(WS);
    while top(ST) \neq parent(s) do
        Passed := Passed ∪ \{pop(ST)\};
    od
    push(ST, s);
    if ∀s' ∈ Passed. s \not\subseteq s'
        then foreach t : s \Rightarrow t do
            if t \models φ then t := delay(t);
            if unbounded(t) then exit(true);
            if deadlocked(t) then exit(true);
            if \exists t' ∈ ST. t = t' then exit;
            push(WS, t);
        od
    fi
od
exit(false);
end
proc Liveness($s_0$, $\varphi$, Passed) \equiv \\
pre(s_0 = delay(s_0)) \\
pre(s_0 \models \varphi) \\
pre(\neg \text{unbounded} s_0 \land \neg \text{deadlocked}(s_0)) \\
pre(\forall s \in \text{Passed. } s \models A\diamond \neg \varphi) \\
WS := \{s_0\}; \\
ST := \emptyset; \\
while WS \neq \emptyset do \\
\hspace{1em} s := \text{pop}(WS); \\
\hspace{1em} while \text{top}(ST) \neq \text{parent}(s) do \\
\hspace{2em} \text{Passed} := \text{Passed} \cup \{\text{pop}(ST)\}; \\
\hspace{1em} od \\
\hspace{1em} \text{push}(ST, s); \\
\hspace{1em} if \forall s' \in \text{Passed. } s \not\subset s' \\
\hspace{2em} then foreach t : s \not\models t do \\
\hspace{3em} if t \models \varphi \text{ then } t := \text{delay}(t); \\
\hspace{3em} if \text{unbounded}(t) \text{ then exit(t); } \\
\hspace{3em} if \text{deadlocked}(t) \text{ then exit(t); } \\
\hspace{3em} if \exists t' \in ST. t = t' \text{ then exit; } \\
\hspace{2em} od \\
\hspace{2em} fi \\
fi \\
\hspace{1em} od \\
\hspace{1em} exit(false); \\
end

[FORMATS05]
Extensions allowing for automatic synthesis of smallest bound $t$ such that $A\diamond \preceq_t \varphi$ holds
Compositionality & Abstraction
The State Explosion Problem

Model-checking is either EXPTIME-complete or PSPACE-complete (for TA’s this is true even for a single TA)
Abstraction

$\textit{Sys}$

$\textit{Abs}$

REDUCE TO

$\textit{Abs}$ $\textit{sat} \, \varphi$

$\textit{Sys} \leq \textit{Abs}$

$\textit{Sys}$ $\textit{sat} \, \varphi$

Preserving safety properties
Compositionality

\[
\begin{align*}
Sys_1 &\leq Abs_1 \\
Sys_2 &\leq Abs_2 \\
Sys_1 | Sys_2 &\leq Abs_1 | Abs_2
\end{align*}
\]

\[
\begin{align*}
Sys_1 &\leq Abs_1 \\
Sys_2 &\leq Abs_2 \\
Abs_1 | Abs_2 &\leq Abs \\
Sys &\leq Abs
\end{align*}
\]
Abstraction

Example
Example Continued

process P1
(())

process P2
(())

P1P2

abstracted by
Proving abstractions
using reachability

Applied to

IEEE 1394a Root contention protocol
(Simons, Stoelinga)

B&O Power Down Protocol
(Ejersbo, Larsen, Skou, FTRTFT2k)

A[] not TestAbstPoP1.BAD

Henrik Ejersbo Jensen PhD Thesis 1999
Further Optimizations
Datastructures for Zones

UPPAAL DBM Library
The library used to manipulate DBMs in UPPAAL.

Welcome!
DBMs [dill89, rokic93, lw97, lw95, bengs02] are efficient data structures to represent clock constraints in timed automata [a90]. They are used in UPPAAL [lw97, lw95, bd04] as the core data structure to represent time. The library features all the common operations such as up (delay or future), down (past), general updates, different extrapolation functions, etc., on DBMs and federations. The library also supports subtractions. The API is in C and C++. The C++ part uses active clocks and hides (to some extent) memory management.

References
Zone Abstractions

- Abstraction taking maximum constant into account necessary for termination
- Utilization of distinction between lower and upper bounds
- Utilization of location-dependency

[TACAS03, TACAS04]
LU Abstraction

THEOREM
For any state in the LU- abstraction there is a state in the original set simulating it
→
LU abstraction is exact wrt reachability
## Zone abstractions

<table>
<thead>
<tr>
<th>Model</th>
<th>Classical</th>
<th>Loc. dep. Max</th>
<th>Loc. dep. LU</th>
<th>Convex Hull</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-n1</td>
<td>-n2</td>
<td>-n3</td>
<td>-A</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>States</td>
<td>Mem</td>
<td>Time</td>
</tr>
<tr>
<td>f5</td>
<td>4.02</td>
<td>82,685</td>
<td>5</td>
<td>0.24</td>
</tr>
<tr>
<td>f6</td>
<td>597.04</td>
<td>1,489,230</td>
<td>49</td>
<td>6.67</td>
</tr>
<tr>
<td>f7</td>
<td>352.67</td>
<td>1,620,542</td>
<td>46</td>
<td>0.47</td>
</tr>
<tr>
<td>f8</td>
<td>2.11</td>
<td>164,528</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>f9</td>
<td>8.76</td>
<td>598,662</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>f10</td>
<td>37.26</td>
<td>2,136,980</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>f11</td>
<td></td>
<td></td>
<td></td>
<td>152.44</td>
</tr>
<tr>
<td>c5</td>
<td>0.55</td>
<td>27,174</td>
<td>3</td>
<td>0.14</td>
</tr>
<tr>
<td>c6</td>
<td>19.39</td>
<td>287,109</td>
<td>11</td>
<td>3.63</td>
</tr>
<tr>
<td>c7</td>
<td></td>
<td></td>
<td></td>
<td>195.35</td>
</tr>
<tr>
<td>c8</td>
<td></td>
<td></td>
<td></td>
<td>0.98</td>
</tr>
<tr>
<td>c9</td>
<td></td>
<td></td>
<td></td>
<td>2.90</td>
</tr>
<tr>
<td>c10</td>
<td></td>
<td></td>
<td></td>
<td>8.42</td>
</tr>
<tr>
<td>c11</td>
<td></td>
<td></td>
<td></td>
<td>24.13</td>
</tr>
<tr>
<td>c12</td>
<td></td>
<td></td>
<td></td>
<td>68.20</td>
</tr>
<tr>
<td>bus</td>
<td>102.28</td>
<td>6,727,443</td>
<td>303</td>
<td>66.54</td>
</tr>
<tr>
<td>philips</td>
<td>0.16</td>
<td>12,823</td>
<td>3</td>
<td>0.09</td>
</tr>
<tr>
<td>sched</td>
<td>17.01</td>
<td>929,726</td>
<td>78</td>
<td>15.09</td>
</tr>
</tbody>
</table>
Symmetry Reduction

- Exploitation of full symmetry may give factorial reduction.

- Many timed systems are inherently symmetric.

- Computation of canonical state representative using swaps.
Symmetry Reduction

- Exploitation of full symmetry may give factorial reduction

- Many timed systems are inherently symmetric

- Computation of canonical state representative using swaps.
Symmetry Reduction

[Formats 2003]
Symmetry Reduction

UPPAAL 3.6

- Iterators
  for (i: int[0,4]) { }

- Quantifiers
  forall (i: int[0,4]) a[i]==0

- Selection
  select i: int[0,4]; guard...

- Template sets
  process P[4](...) { }

- Scalar set based symmetry reduction

- Compact state-space representations

- Priorities

Martijn Henriks, Nijmegen U
Informationsteknologi

UCb

D-UPPAAL

Gerd Behrman

"Distributed implementation of UPPAAL on PC-cluster [CAV'00, PDMC'02, STTT'03]."

Applications
- Synthesis of Dynamic Voltage Scaling strategies (CISS).
- Ad-hoc mobile real-time protocol (Leslie Lamport) - 25GB in 3 min!

Running on NorduGrid. Local cluster: 50 CPUs and 50GB of RAM

To be used as inspiration for verification GRID platform within ARTIST2 NoE.