Timed Automata – Decidability Results
Decidability?

OBSTACLE: Uncountably infinite state space
Derived Relations and Reachability

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

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

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

**Definition**

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

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

Definition
Let $G \subseteq L$ be a set of goal locations. An equivalence relation $R$ on $L \times R^G$ 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 $(l, u) \xrightarrow{a} (l', u')$ then $(n, v) \xrightarrow{a} (n', v')$ 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
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning
Stable Quotient

Partitioning

Reachable?

0 → ε → 1 → a → 2 → ε → 3 → a → 4 → ε → 5 → c → 6
Region Equivalence

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, $u \approx u'$ 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 $u(y) \leq c_y$,
   
   \[ fr(u(x)) \leq fr(u(y)) \text{ iff } fr(u'(x)) \leq fr(u'(y)) \];

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

3. For all $x \in C$ with $u(x) \leq c_x$, $fr(u(x)) = 0$ iff $fr(u'(x)) = 0$.

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

Each region may be represented by specifying

1. for every clock $x$ a constraint from

   $$\{x = c \mid c = 0, 1, \ldots, c_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots, c_x\} \cup \{x > c_x\}$$

2. for every pair of clocks $x, y$ such that $c - 1 < x < c$ and $d - 1 < y < d$ appears in 1., whether $fr(x)$ is $<$, $=$ or $>$ than $fr(y)$.

**Theorem**
The number of regions is $n! \cdot 2^n \cdot \prod_{x \in C}(2c_x + 2)$. 
Stability of Regions

Lemma

1. If $u \cong u'$ then $\forall d. \exists d'. u + d \equiv u' + d'$;
2. If $u \cong u'$ then $\forall g \in B(X). u \models g \iff u' \models g$;
3. If $u \cong u'$ then $\forall r \subseteq C. u[r] \cong u'[r]$.

Theorem

Let $(l, u) \cong (l', u')$ iff $l = l'$ and $u \cong u'$. Then $\cong$ is a TAB with respect to any set of goal locations $G$.

*Here $\cong$ and $g$ should agree on the maximal constants.
Regions

Successor Operation (wrt delay)

An equivalence class (i.e. a region)
Regions
Reset Operation

An equivalence class (i.e. a region) $r$
An Example Region Graph

(a) $\frac{x \geq 2}{\{x\}}$

(b) 

- A: $l$, $x = 0$
- B: $l$, $0 < x < 1$
- C: $l$, $x = 1$
- D: $l$, $1 < x < 2$
- E: $l$, $x = 2$
- F: $l$, $x > 2$
Modified light switch

- $x \geq 1 \quad \{x, y\}$
- $y = 3 \quad \{x\}$
- $x \geq 2 \quad \{x\}$
- $\text{inv}(\text{off}) = \text{true}$
- $\text{inv}(\text{on}) = y \leq 3$

$\text{AG}(x \leq y)$
$\text{AG}(\text{on} \Rightarrow \text{AF} \text{off})$
$\text{AG}(\text{on} \Rightarrow \text{AF}_{\leq 9} \text{off})$
Reachable part of region graph

Properties

AG(x ≤ y)
AG(on ⇒ AF\text{off})
AG(on ⇒ AF_{x \leq 9}\text{off})
Finite Representation of Regions

Let $\Gamma$ be a triple of the form:

$$\Gamma = \left< C_>, \mathcal{H}, [C_0, C_1, \ldots, C_k] \right>$$

where

- $C_>, C_0, C_1, \ldots, C_k$ is a partitioning of $C$ ($C_>$ and $C_0$ are allowed to be empty).

- $\mathcal{H} : (C \setminus C_>) \rightarrow \mathbb{N}$ with $\mathcal{H}(x) \leq c_x$. 
Representation of Regions (cont)

\( u \in \llbracket \Gamma \rrbracket \) if and only if

- \( \forall x \in C_. \ u(x) > c_x; \)
- \( \forall x \in C\setminus C_. \ [u(x)] = \mathcal{H}(x); \)
- \( \forall x \in C_0. \ fr(u(x)) = 0; \)
- \( \forall x, y \in C_i. \ fr(u(x)) = fr(u(y)); \)
- \( \forall x \in C_i, y \in C_j. \ fr(u(x)) < fr(u(y)) \iff i < j; \)

\( \llbracket \Gamma \rrbracket \) is a region, and any region is representable as a triplet.
Decidability & Complexity

**Theorem**
The transition relations $\delta$ and $\rightarrow a$ between regions may be computed effectively using the triplet or the logical characterization of regions.

**Theorem**
Let $A$ be a timed automaton with location $l$.
It is decidable whether $l \in \text{Reach}(A)$.

**Theorem**
Let $A$ be a timed automaton with location $l$.
Deciding $l \in \text{Reach}(A)$ is PSPACE-complete.
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>( x := c, x := y )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pspace complete</td>
<td>Pspace complete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := x + 1 )</th>
<th>Diagonal-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pspace complete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := y + c )</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undecidable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := x - 1 )</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undecidable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x &lt; c, x \leq c )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pspace complete</td>
<td>Pspace complete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x &gt; c, x \geq c )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pspace complete</td>
<td>Pspace complete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( y + c &lt; x &lt; (y + d) )</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undecidable</td>
<td></td>
</tr>
</tbody>
</table>

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

---

<table>
<thead>
<tr>
<th>( := c, x := y )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA-bisimilar</td>
<td>TA-bisimilar</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := x + 1 )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td>Turing</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := y + c )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_{\varepsilon}</td>
<td>TA_{\varepsilon}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x &gt; c, x \geq c )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_{\varepsilon}</td>
<td>TA_{\varepsilon}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x := y + c )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td>Turing</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( (y+)c &lt; x &lt; (y+)d )</th>
<th>Diagonal-free</th>
<th>W Diagonals</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_{\varepsilon}</td>
<td>TA_{\varepsilon}</td>
<td></td>
</tr>
</tbody>
</table>

With \( \sim \in \{<, \leq, \geq, >\} \) and \( c, d \in \mathbb{Q}_+ \)
TCTL:
Timed Computational Tree Logic
TCTL = CTL + Time

\[ \phi ::= p \mid \alpha \mid \neg \phi \mid \phi \lor \phi \mid z \text{ in } \phi \mid E[\phi U \phi] \mid A[\phi U \phi] \]

- \( p \in AP \), *atomic propositions*,
- \( z \in D \), *formula clocks*,
- \( \alpha \) – *constraints over formula clocks and automata clock*.
- \( z \text{ in } \phi \) – “freeze operator” introduces new *formula clock* \( z \)
- \( E[\phi U \phi], A[\phi U \phi] \) - like in CTL
- No \( EX \phi \)
Derived Operators

\[ A [\phi \bigcup_{7} \psi] = z \text{ in } A [(\phi \land z \leq 7) \bigcup \psi]. \]

Along any path \( \phi \) holds continuously until within 7 time units \( \psi \) becomes valid.

\[ \text{EF}_{5} \phi = z \text{ in } \text{EF} (z < 5 \land \phi) \]

The property \( \phi \) becomes valid within 5 time units.
Paths

A path is an infinite sequence $s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ of states alternated by transition labels such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \geq 0$.

Example:

$$(\text{off}, x = y = 0) \xrightarrow{3.5} (\text{off}, x = y = 3.5) \xrightarrow{\text{push}}$$

$$(\text{on}, x = y = 0) \xrightarrow{\pi} (\text{on}, x = y = \pi) \xrightarrow{\text{push}}$$

$$(\text{on}, x = 0, y = \pi) \xrightarrow{3} (\text{on}, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)}$$

$$(\text{on}, x = 9-(\pi+3), y = 9) \xrightarrow{\text{click}} (\text{off}, x = 0, y = 9) \ldots$$
Elapsed time in path

\[
\Delta(\sigma,0) = 0 \\
\Delta(\sigma,i+1) = \Delta(\sigma,i) + \begin{cases} 
0 & \text{if } a_i = * \\
 a_i & \text{if } a_i \in \mathbb{R}^+. 
\end{cases}
\]

Example:

\[ \sigma = (off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{\text{push}} \]
\[ (on, x = y = 0) \xrightarrow{\pi} (on, x = y = \pi) \xrightarrow{\text{push}} \]
\[ (on, x = 0, y = \pi) \xrightarrow{3} (on, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)} \]
\[ (on, x = 9-(\pi+3), y = 9) \xrightarrow{\text{click}} (off, x = 0, y = 9) \ldots \]

\[ \Delta(\sigma,1) = 3.5, \Delta(\sigma,6) = 3.5 + 9 = 12.5 \]
TCTL Semantics

\[
s, w \models p \quad \text{iff} \quad p \in \text{Label}(s)
\]

\[
s, w \models \alpha \quad \text{iff} \quad v \cup w \models \alpha
\]

\[
s, w \models \neg \phi \quad \text{iff} \quad \neg (s, w \models \phi)
\]

\[
s, w \models \phi \lor \psi \quad \text{iff} \quad (s, w \models \phi) \lor (s, w \models \psi)
\]

\[
s, w \models z \text{ in } \phi \quad \text{iff} \quad s, \text{reset } z \text{ in } w \models \phi
\]

\[
s, w \models E [\phi U \psi] \quad \text{iff} \quad \exists \sigma \in P^\infty_M(s). \exists (i, d) \in \text{Pos}({\sigma}).
\]

\[
\quad \quad \quad \quad (\sigma(i, d), w+\Delta(\sigma, i) \models \psi \land
\quad \quad \quad \quad (\forall (j, d') \ll (i, d). \sigma(j, d'), w+\Delta(\sigma, j) \models \phi \lor \psi))
\]

\[
s, w \models A [\phi U \psi] \quad \text{iff} \quad \forall \sigma \in P^\infty_M(s). \exists (i, d) \in \text{Pos}({\sigma}).
\]

\[
\quad \quad \quad \quad ((\sigma(i, d), w+\Delta(\sigma, i)) \models \psi \land
\quad \quad \quad \quad (\forall (j, d') \ll (i, d). (\sigma(j, d'), w+\Delta(\sigma, j)) \models \phi \lor \psi)).
\]

\[
(i, d) \ll (i', d') \quad \text{iff} \quad (i < j) \text{ or } ((i = j) \text{ and } (d < d'))
\]
Timeliness Properties

\[ AG[send(m) \Rightarrow AF_{\leq 5} \text{receive}(r_m)] \]

\text{receive}(m) \text{ occurs within } 5 \text{ time units after } send(m)

\[ EG[send(m) \Rightarrow AF_{=11} \text{receive}(r_m)] \]

\text{receive}(m) \text{ occurs exactly } 11 \text{ time units after } send(m)

\[ AG[AF_{\geq 25} \text{putbox}] \]

\text{putbox} \text{ occurs periodically (exactly) every } 25 \text{ time units}

\text{(note: other putbox’s may occur in between)}
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)
\((n, 1 \leq x \leq 4, 1 \leq y \leq 3)\)

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

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:

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

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

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

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

- Conjunction
  $$[Z\&g] = [Z] \cap [g]$$
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?

Left
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?
Symbolic Exploration

Reachable?

\[ y := 0 \]
\[ x := 0 \]
\[ y <= 2 \]
\[ x <= 2 \]
\[ y <= 2, x = 4 \]

Delay
Symbolic Exploration

Reachable?
Fischer’s Protocol

**Init**

\[ V = 1 \]

**Analysis using zones**

```
X < 10 \quad X := 0 \quad X > 10
```

```
Y < 10 \quad Y := 0 \quad Y > 10
```

```
A1 \xrightarrow{V := 1} B1 \xrightarrow{V = 1} CS1
```

```
A2 \xrightarrow{V := 2} B2 \xrightarrow{V = 2} CS2
```

Critical Section
Fischers cont.

Untimed case
Fischers cont.

Untimed case

A1, A2, v=1 → A1, B2, v=2 → A1, CS2, v=2 → B1, CS2, v=1 → CS1, CS2, v=1

Taking time into account

\[
\begin{align*}
X \leq 10 & \quad X = 0 & \quad X > 10 \\
V = 1 & & V = 1 \\
A1 & \rightarrow B1 & \rightarrow CS1 \\
A2 & \rightarrow B2 & \rightarrow CS2
\end{align*}
\]
Fischers cont.

Untimed case

A1, A2, v=1 → A1, B2, v=2 → A1, CS2, v=2 → B1, CS2, v=1 → CS1, CS2, v=1

Taking time into account

\[
\begin{align*}
V &:= 1 \\
X &< 10 \\
X &> 10
\end{align*}
\]

\[
\begin{align*}
V &:= 2 \\
Y &< 10 \\
Y &> 10
\end{align*}
\]
Fischers cont.

Untimed case

Taking time into account
Fischers cont.

**Untimed case**

\[
\begin{align*}
A1, A2, \nu = 1 & \rightarrow A1, B2, \nu = 2 \\
A1, B2, \nu = 2 & \rightarrow A1, CS2, \nu = 2 \\
A1, CS2, \nu = 2 & \rightarrow B1, CS2, \nu = 1 \\
B1, CS2, \nu = 1 & \rightarrow CS1, CS2, \nu = 1
\end{align*}
\]

**Taking time into account**

\[
\begin{align*}
X < 10 & \rightarrow V = 1 \\
X = 0 & \rightarrow V = 1 \\
X > 10 & \rightarrow V = 1
\end{align*}
\]
Fischers cont.

Untimed case

Taking time into account
Forward Reachability

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

\[
\begin{align*}
\text{INITIAL} & \quad \text{Passed} := \emptyset; \\
& \quad \text{Waiting} := \{(n0,z0)\}
\end{align*}
\]

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

\[
\text{INITIAL } \text{Passed} := \emptyset; \\
\text{Waiting} := \{(n_0, Z_0)\}
\]

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

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

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

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

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

**UNTIL**
- **Waiting** = \( \emptyset \)
  or
- Final is in **Waiting**
Forward Reachability

Initial $\rightarrow$ Final?

**INITIAL**

- $\text{Passed} := \emptyset$
- $\text{Waiting} := \{(n_0, Z_0)\}$

**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**

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

Difference Bounded Matrices

Bellman 1958, Dill 1989

Inclusion

\[
\begin{align*}
D1 & : & x \leq 1 \\
 & & y-x \leq 2 \\
 & & z-y \leq 2 \\
 & & z \leq 9
\end{align*}
\]

\[
\begin{align*}
D2 & : & x \leq 2 \\
 & & y-x \leq 3 \\
 & & y \leq 3 \\
 & & z-y \leq 3 \\
 & & z \leq 7
\end{align*}
\]
Canonical Datastructures for Zones

Difference Bounded Matrices

Inclusion

D1

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

D2

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

**Difference Bounded Matrices**

**Emptiness**

$$x \leq 1$$
$$y \geq 5$$
$$y - x \leq 3$$

**Compact**

```
D
x<=1
y>=5
y-x<=3
```

**Graph**

- **Negative Cycle**
  - iff empty solution set
Canonical Datastructures for Zones

Difference Bounded Matrices

Future D

1 <= x <= 4
1 <= y <= 3

Future

1 <= x, 1 <= y
-2 <= x-y <= 3

Shortest Path Closure
Remove upper bounds on clocks
Canonical Datastructures for Zones

Difference Bounded Matrices

Remove all bounds involving y and set y to 0

y=0, 1<=x
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 \( O(n^3) \)

\( \text{Space worst } O(n^2) \)
\( \text{practice } O(n) \)
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.
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

\[ \text{INITIAL} \quad \text{Passed} := \emptyset; \]
\[ \text{Waiting} := \{(n_0, Z_0)\} \]

\[ \text{REPEAT} \]
- 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}

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

Init \rightarrow Final ?
Earlier Termination

\textbf{INITIAL} \hspace{1em} \textbf{Passed} := \emptyset; \\
\hspace{1em} \textbf{Waiting} := \{(n0, Z0)\}

\textbf{REPEAT}
- \textbf{pick} \ (n, Z) \textbf{in} \ \textbf{Waiting}
- \textbf{if} \ \textbf{for some} \ Z' \ Z \\
\hspace{1em} (n, Z') \textbf{in} \ \textbf{Passed} \ \textbf{then} \ \textbf{STOP}
- \textbf{else} /explore/ \ \textbf{add} \\
\hspace{1em} \{(m, U) : (n, Z) \Rightarrow (m, U)\} \textbf{to} \ \textbf{Waiting}; \\
\hspace{1em} \textbf{Add} \ (n, Z) \ \textbf{to} \ \textbf{Passed}

\textbf{UNTIL} \ \textbf{Waiting} = \emptyset \\
\hspace{1em} \textbf{or} \\
\hspace{2em} \textbf{Final} \ \textbf{is} \ \textbf{in} \ \textbf{Waiting}
Earlier Termination

\[ \text{INITIAL Passed} := \emptyset; \]
\[ \text{Waiting} := \{(n_0, Z_0)\} \]

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

\[ \text{UNTIL Waiting} = \emptyset \]
\[ \text{or} \]
\[ \text{Final is in Waiting} \]
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
UPPAAL 1995 - 2001

Every 9 month
10 times better performance!

Dec’96
Liveness Properties

\[ F ::= E \Box P \mid A \Diamond P \mid P \Rightarrow Q \]

\begin{align*}
\text{Possibly always} & \quad P \\
\text{Eventually} & \quad P \\
\text{leads to} & \quad Q
\end{align*}

\[ P \leadsto Q \quad \text{is equivalent to} \quad A \Box (P \Rightarrow A \Diamond Q) \]

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
  fi
  exit(false);
end
proc Liveness\(s_0, \varphi, \text{Passed}\) \equiv
\begin{align*}
\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}\(\forall s \in \text{Passed}. s \models A\Diamond \neg \varphi\)
\end{align*}
\begin{align*}
\text{WS} & := \{s_0\}; \\
\text{ST} & := \emptyset; \\
\text{while } & \text{WS} \neq \emptyset \text{ do} \\
& s := \text{pop(WS)}; \\
& \text{while } \text{top(ST)} \neq \text{parent}(s) \text{ do} \\
& \quad \text{Passed} := \text{Passed} \cup \{\text{pop(ST)}\}; \\
& \text{od} \\
& \text{push(ST, s)}; \\
& \text{if } \forall s' \in \text{Passed}. s \not\subseteq s' \text{ then} \\
& \quad \text{foreach } t : s \not\Rightarrow t \text{ do} \\
& \quad \text{if } t \models \varphi \text{ then} t := \text{delay}(t); \\
& \quad \text{if } \text{unbounded}(t) \text{ then } \text{exit(true)}; \\
& \quad \text{if } \text{deadlocked}(t) \text{ then } \text{exit(true)}; \\
& \quad \text{if } \exists t' \in \text{ST}. t = t' \text{ then } \text{exit(true)}; \\
& \quad \text{push(WS, t)}; \\
& \text{fi} \\
& \text{fi} \\
& \text{od} \\
& \text{exit(false)}; \\
\text{end}
\end{align*}
\begin{align*}
\text{proc } \text{Liveness}(s_0, \varphi, \text{Passed}) & \equiv \\
& \quad \text{pre}(s_0 = \text{delay}(s_0)) \\
& \quad \text{pre}(s_0 \models \varphi) \\
& \quad \text{pre}(\neg \text{unbounded}(s_0) \land \neg \text{deadlocked}(s_0)) \\
& \quad \text{pre}(\forall s \in \text{Passed}, s \models A \Diamond \neg \varphi) \\
& \quad \text{WS} := \{s_0\}; \\
& \quad ST := \emptyset; \\
& \quad \text{while } WS \neq \emptyset \text{ do} \\
& \quad \quad s := \text{pop}(WS); \\
& \quad \quad \text{while } \text{top}(ST) \neq \text{parent}(s) \text{ do} \\
& \quad \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' \text{ then} \\
& \quad \quad \text{foreach } t : s \not\Rightarrow t \text{ do} \\
& \quad \quad \quad \text{if } t \models \varphi \text{ then } t := \text{delay}(t); \\
& \quad \quad \quad \text{if } \text{unbounded}(t) \text{ then } \text{exit}(t); \\
& \quad \quad \quad \text{if } \text{deadlocked}(t) \text{ then } \text{exit}(t); \\
& \quad \quad \quad \text{if } \exists t' \in ST, t = t' \text{ then } \text{exit}(t); \\
& \quad \quad \quad \text{push}(WS, t); \\
& \quad \quad \text{fi} \\
& \quad \text{od} \\
& \quad \text{fi} \\
& \quad \text{od} \\
& \quad \text{exit}(false); \\
& \end{align*}
proc Liveness \(s_0, \varphi, \text{Passed} \) ≡
pre \(s_0 = \text{delay}(s_0)\)
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 ≠ \(\emptyset\) do
  \(s := \text{pop}(WS)\);
  \(\text{while} \top(ST) \neq \text{parent}(s) \text{ do}
    \text{Passed} := \text{Passed} \cup \{\text{pop}(ST)\};
  \text{od}
  \text{push}(ST, s);
  \text{if} \forall s' \in \text{Passed}. s \not\subseteq s'\)
  \text{then foreach} \(t : s \not\Rightarrow t\) do
    \text{if} t \models \varphi \text{ then } t := \text{delay}(t);
    \text{if} \text{unbounded}(t) \text{ then } \text{exit}(t);
    \text{if} \text{deadlocked}(t) \text{ then } \text{exit}(t);
    \text{if} \exists t' \in ST. t = t' \text{ then } \text{exit}(t);
  \text{fi}
  \text{od}
  \text{od}
\text{exit}(false);
\textbf{Liveness} \hspace{5cm} E \square \phi \quad (A \diamond \neg \phi)

\begin{pro}
\textbf{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 \wedge \neg \text{deadlocked}(s_0)) \\
\text{pre}(\forall s \in \text{Passed. } s \models A \diamond \neg \varphi) \\
\text{WS} := \{s_0\}; \\
\text{ST} := \emptyset; \\
\text{while } WS \neq \emptyset \text{ do} \\
\quad s := \text{pop}(WS); \\
\quad \text{while top}(ST) \neq \text{parent}(s) \text{ do} \\
\quad \quad \text{Passed} := \text{Passed} \cup \{\text{pop}(ST)\}; \\
\quad \text{od} \\
\quad \text{push}(ST, s); \\
\quad \text{if } \forall s' \in \text{Passed. } s \notin s' \text{ then} \text{ foreach } t : s \triangleleft 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}(trm); \\
\quad \quad \text{if } \text{deadlocked}(t) \text{ then } \text{exit}(trm); \\
\quad \quad \text{if } \exists t' \in ST. t = t' \text{ then } \text{exit} \\
\quad \quad \text{push}(WS, t); \\
\quad \text{fi} \\
\quad \text{od} \\
\quad \text{od} \\
\quad \text{exit}(false); \text{ end}
\end{pro}
\textbf{Liveness}

\[ \mathbb{E} \square \phi \quad (A \lozenge \neg \phi) \]

\begin{verbatim}
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 Passed. s \models A \lozenge \neg \varphi)
  WS := \{s_0\};
  ST := \emptyset;
  while WS \neq \emptyset do
    s := pop(WS);
    while \text{top}(ST) \neq \text{parent}(s) do
      Passed := Passed \cup \{\text{pop}(ST)\};
    od
    push(ST, s);
    if \forall s' \in Passed. s \not\subseteq s'
      then foreach t : s \not\Rightarrow t do
        if t \models \varphi 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
    fi
    od
  exit(false);
end
\end{verbatim}
\[
\text{proc Liveness}(s_0, \varphi, \text{Passed}) \equiv \\
\text{pre}(s_0 = \text{delay}(s_0)) \\
\text{pre}(s_0 \models \varphi) \\
\text{pre}(\neg \text{unbounded}(s_0) \land \neg \text{deadlocked}(s_0)) \\
\text{pre}(\forall s \in \text{Passed}. s \models A\Box \neg \varphi) \\
\text{WS} := \{s_0\}; \\
\text{ST} := \emptyset; \\
\text{while } \text{WS} \neq \emptyset \text{ do} \\
\quad s := \text{pop(WS)}; \\
\quad \text{while } \text{top(ST)} \neq \text{parent}(s) \text{ do} \\
\quad\quad \text{Passed} := \text{Passed} \cup \{\text{pop(ST)}\}; \\
\quad\quad \text{od} \\
\quad \text{push(ST, s);} \\
\quad \text{if } \forall s' \in \text{Passed}. s \not\subseteq s' \\
\quad\quad \text{then foreach } t : s \models t \text{ do} \\
\quad\quad\quad \text{if } t \models \varphi \text{ then } t := \text{delay}(t); \\
\quad\quad\quad\quad \text{if } \text{unbounded}(t) \text{ then } \text{exit}(t); \\
\quad\quad\quad\quad \text{if } \text{deadlocked}(t) \text{ then } \text{exit}(t); \\
\quad\quad\quad\quad \text{if } \exists t' \in \text{ST}. t = t' \text{ then } \text{exit}(t'); \\
\quad\quad\quad\quad \text{push(WS, t);} \\
\quad\quad \text{od} \\
\quad \text{od} \\
\quad \text{exit}(\text{false}); \\
\text{end}
\]
proc Liveness(s₀, φ, Passed) ≡
pre(s₀ = delay(s₀))
pre(s₀ \models φ)
pre(\neg \text{unbounded}(s₀) \land \neg \text{deadlocked}(s₀))
pre(\forall s \in Passed. s \models A\Diamond \neg φ)
WS := \{s₀\};
ST := \emptyset;
while WS \neq \emptyset do
  s := pop(WS);
  while top(ST) \neq parent(s) do
    Passed := Passed \cup \{\text{pop}(ST)\};
  od
  push(ST, s);
  if \forall s' \in Passed. s \not\subseteq s'
    then foreach t : t \models φ \Rightarrow t do
      if t \models φ then t := delay(t);
      if unbounded(t) then exit(trm);
      if deadlocked(t) then exit(trm);
      if \exists t' \in ST. t = t' then exit(trm);
      push(WS, t);
    od
  fi
  exit(false);
end

Extensions allowing for automatic synthesis of smallest bound \( t \) such that \( A\Diamond s.t \models φ \) holds
Verification Options

**Search Order**
- Depth First
- Breadth First

**State Space Reduction**
- None
- Conservative
- Aggressive

**State Space Representation**
- DBM
- Compact Form
- Under Approximation
- Over Approximation

**Diagnostic Trace**
- Some
- Shortest
- Fastest

**Extrapolation**

**Hash Table size**

**Reuse**
State Space Reduction

However, **Passed** list useful for efficiency

No Cycles: **Passed** list not needed for *termination*
State Space Reduction

Cycles:
Only symbolic states involving loop-entry points need to be saved on **Passed** list
To Store or Not To Store

117 states<sub>total</sub> → 81 states<sub>entrypoint</sub> → 9 states

Time OH less than 10%

Audio Protocol

Behrmann, Larsen, Pelanek 2003
To Store or Not to Store

<table>
<thead>
<tr>
<th>Entry Points</th>
<th>Covering Set</th>
<th>Successors</th>
<th>Random $p = 0.1$</th>
<th>Distance $k = 10$</th>
<th>Combination $k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fischer 3,077</td>
<td>27.1%</td>
<td>42.1%</td>
<td>47.9%</td>
<td>53.7%</td>
<td>67.6%</td>
</tr>
<tr>
<td>BRP 6,060</td>
<td>70.5%</td>
<td>16.5%</td>
<td>19.8%</td>
<td>18.3%</td>
<td>15.8%</td>
</tr>
<tr>
<td>Token Ring 15,103</td>
<td>33.0%</td>
<td>10.3%</td>
<td>20.7%</td>
<td>17.2%</td>
<td>17.5%</td>
</tr>
<tr>
<td>Train-gate 16,666</td>
<td>71.1%</td>
<td>27.4%</td>
<td>24.2%</td>
<td>31.8%</td>
<td>24.2%</td>
</tr>
<tr>
<td>Dacapo 30,502</td>
<td>29.4%</td>
<td>24.3%</td>
<td>24.9%</td>
<td>12.2%</td>
<td>12.7%</td>
</tr>
<tr>
<td>CSMA 47,857</td>
<td>94.0%</td>
<td>75.9%</td>
<td>81.2%</td>
<td>105.9%</td>
<td>114.9%</td>
</tr>
<tr>
<td>BOCDP 203,557</td>
<td>25.2%</td>
<td>22.5%</td>
<td>6.5%</td>
<td>10.2%</td>
<td>9.3%</td>
</tr>
<tr>
<td>BOPDP 1,013,072</td>
<td>14.7%</td>
<td>13.2%</td>
<td>42.1%</td>
<td>15.2%</td>
<td>11%</td>
</tr>
<tr>
<td>Buscoupler 3,595,108</td>
<td>53.2%</td>
<td>13.6%</td>
<td>40.5%</td>
<td>31.7%</td>
<td>24.6%</td>
</tr>
</tbody>
</table>
Over-approximation

Convex Hull

TACAS04: An **EXACT** method performing as well as Convex Hull has been developed based on abstractions taking max constants into account distinguishing between clocks, locations and $\leq$ & $\geq$
Under-approximation

Bitstate Hashing

Diagram showing states and transitions: Init, Passed, Waiting, and Final states with transitions labeled $n, Z$, $m, U$, and $n, Z'$.
Under-approximation
*Bitstate Hashing*

```
Passed = Bitarray

UPPAAL
8 Mbits
```

```
Hashfunction F
```

```
Passed
```

```
Waiting
```

```
Final
```

```
1
0
1
0
1
```

```
m, U
```

```
\[ m, U \]
```

```
m, Z
```

```
\[ m, Z \]
```

```
\[ \mathbf{1} \]
```

```
\[ 0 \]
```

```
\[ 0 \]
```

```
\[ 1 \]
```

---

```
\[ \mathbf{1} \]
```

---

```
\[ 0 \]
```

---

```
\[ 1 \]
```

---

```
\[ 0 \]
```

---

```
\[ 1 \]
```

---
Modelling Patterns
Variable Reduction

- Reduce size of state space by explicitly resetting variables when they are not used!

- Automatically performed for clock variables (active clock reduction)

```c
// Remove the front element of the queue
void dequeue()
{
    int i = 0;
    len -= 1;
    while (i < len)
    {
        list[i] = list[i + 1];
        i++;
    }
    list[i] = 0;
}
```
Clock Reduction

x is only active in location S1

Definition
x is inactive at S if on all path from S, x is always reset before being tested.
# Synchronous Value Passing

<table>
<thead>
<tr>
<th>Unconditional</th>
<th>Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-way</strong></td>
<td></td>
</tr>
<tr>
<td><code>c!</code></td>
<td><code>c!</code></td>
</tr>
<tr>
<td><code>var := out</code></td>
<td><code>var := out</code></td>
</tr>
<tr>
<td><code>in := var, var := 0</code></td>
<td><code>in := var, var := 0</code></td>
</tr>
<tr>
<td><img src="diagram1.png" alt="Diagram" /></td>
<td><img src="diagram2.png" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Asymmetric two-way</strong></td>
<td></td>
</tr>
<tr>
<td><code>c!</code></td>
<td><code>c!</code></td>
</tr>
<tr>
<td><code>var := out</code></td>
<td><code>var := out</code></td>
</tr>
<tr>
<td><code>d?</code></td>
<td><code>d?</code></td>
</tr>
<tr>
<td><code>in := var, var := 0</code></td>
<td><code>in := var, var := 0</code></td>
</tr>
<tr>
<td><img src="diagram3.png" alt="Diagram" /></td>
<td><img src="diagram4.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Atomicity

- To allow encoding of control structure (for-or while-loops, conditionals, etc.) without erroneous interleaving

- To allow encoding of multicasting.

- Heavy use of committed locations.

\[\begin{align*}
\text{len} &> 0 \\
\text{go}[\text{front()}]!
\end{align*}\]

\[\begin{align*}
\text{e : id_t} \\
\text{len} &== 0 \\
\text{appr}[e]? \\
\text{enqueue}(e)
\end{align*}\]

\[\begin{align*}
\text{e : id_t} \\
\text{e} &== \text{front()} \\
\text{leave}[e]?
\text{enqueue}(e)
\end{align*}\]

\[\begin{align*}
\text{e : id_t} \\
\text{stop}[\text{tail()}]!
\end{align*}\]
Bounded Liveness

- **Intent:** Check for properties that are guaranteed to hold eventually within some upper (time) bound.
  - Provide additional information (with a valid bound).
  - More efficient verification.
  - \( \phi \leadsto_{\leq t} \psi \) reduced to \( A(\Box(b \Rightarrow z \leq t)) \) with bool \( b \) set to \textit{true} and clock \( z \) reset when \( \phi \) starts to hold. When \( \psi \) starts to hold, set \( b \) to \textit{false}. 
Bounded Liveness

- The truth value of $b$ indicates whether or not $\psi$ should hold in the future.

\[ b = \text{true, check } z \leq t \]

\[ b = \text{false} \]

\[ A[] (b \text{ imply } z \leq t) \]

\[ b \rightarrow \neg b \text{ (for non-zenoness)} \]

\[ E<> b \text{ (for meaningful check)} \]
Timers

Parametric timer:
- (re-)start(value)
  start! var=value
- expired?
  active (bool)
- active go?
  (bool+urgent chan)
- time-out event
  timeout?

Declare 'to' with a tight range.
Zenoness

Problem: UPPAAL does not check for zenoness directly.
- A model has “zeno” behavior if it can take an infinite amount of actions in finite time.
- That is usually not a desirable behavior in practice.
- Zeno models may wrongly conclude that some properties hold though they logically should not.
- Rarely taken into account.

Solution: Add an observer automata and check for non-zenoness, i.e., that time will always pass.
Zenoness

Detect by:
• adding the observer:

Constant (10) can be anything (>0), but choose it well w.r.t. your model for efficiency. Clocks ‘x’ are local.

• and check the property
ZenoCheck.A --> ZenoCheck.B
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

\[ \text{sat } \phi \]

REDUCE TO

\[ \text{Abs sat } \phi \]

\[
\begin{array}{c}
\text{Abs sat } \phi \\
\hline
\text{Sys sat } \phi
\end{array}
\]

\[
\text{Sys } \leq \text{Abs}
\]
Compositionality

Sys₁ ≤ Abs₁
Sys₂ ≤ Abs₂

Sys₁ | Sys₂ ≤ Abs₁ | Abs₂

Sys₁ ≤ Abs₁
Sys₂ ≤ Abs₂

Abs₁ | Abs₂ ≤ Abs

Sys ≤ Abs
Abstraction & Compositionality

dealing w stateexplosion
Abstraction Example
Example Continued

process P1
()

S0
a1 ?
S1
x := 0
S2
x >= 2
S3
a2 !
S4

process P2
()

S0
a2 ?
S1
x := 0
S2
x >= 2
S3
a3 !
S4

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

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form
  [RTSS97]
- Clock Difference Diagrams
  [CAV99]
- PW List
  [SPIN03]
Datastructures for Zones

- Difference Bounded

UPPAAL DBM Library
The library used to manipulate DBMs in UPPAAL

Welcome!
DBMs [dill99, rolicki93, lawr95, bengtsson02] are efficient data structures to represent clock constraints in timed automata [a90]. They are used in UPPAAL [lpy97, by04, 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 memory management.

References

Elegant RUBY bindings for easy implementations

[SPIN03]
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>Time</td>
<td>States</td>
<td>Mem</td>
</tr>
<tr>
<td>f5</td>
<td>4.02</td>
<td>82,685</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>f6</td>
<td>597.04</td>
<td>1,489,230</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>f7</td>
<td>352.67</td>
<td>1,620,542</td>
<td>46</td>
<td></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>37.26</td>
<td>2,136,980</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>f10</td>
<td>152.44</td>
<td>7,510,382</td>
<td>268</td>
<td></td>
</tr>
<tr>
<td>f11</td>
<td>0.55</td>
<td>27,174</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>c5</td>
<td>19.39</td>
<td>287,109</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>c6</td>
<td>195.35</td>
<td>813,924</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>c7</td>
<td>2.90</td>
<td>132,623</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>c8</td>
<td>8.42</td>
<td>341,452</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>c9</td>
<td>24.13</td>
<td>859,265</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>c10</td>
<td>68.20</td>
<td>2,122,286</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>bus</td>
<td>102.28</td>
<td>6,727,443</td>
<td>303</td>
<td></td>
</tr>
<tr>
<td>philips</td>
<td>0.16</td>
<td>12,823</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>sched</td>
<td>17.01</td>
<td>929,726</td>
<td>78</td>
<td></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
  
  \[
  \text{for } (i: \text{int}[0,4]) \{ \}
  \]

- Quantifiers
  
  \[
  \text{forall } (i: \text{int}[0,4]) \ a[i]==0
  \]

- Selection
  
  \[
  \text{select } i: \text{int}[0,4]; \text{ guard...}
  \]

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

- Scalar set based symmetry reduction

- Compact state-space representations

- Priorities

Gerd Behrmann

Martijn Henriks, Nijmegen U

Processes
D-UPPAAL
Gerd Behrmann

- Distributed implementation of UPPAAL on PC-cluster [CAV'00, PDMC'02, STTT'03].
- WWW frontend available
- 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.
D-UPPAAL
Gerd Behrmann

- Distributed implementation of UPPAAL on PC-cluster [CAV'00, PDMC'02, STTT'03].
- WWW frontend available.
- 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 500GB RAM
- To be used as inspiration for verification GRID platform with ARTIST2 NoE.
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.