Symbolic Model Checking

Binary Decision Diagrams
Combinatorial Circuits
Combinatorial Problems

Sudoku

Eight Queen
Control Programs
A Train Simulator, visualSTATE (VVS)

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: $10^{476}$

“Ideal” presentation: 1 bit/state will clearly NOT work!

BUGS?
Reduced Ordered Binary Decision Diagrams [Bryant’86]

• Compact representation of boolean functions allowing effective manipulation (satisfiability, validity, ....)

or

• Compact representation of sets over finite universe allowing effective manipulations.
Binary Decision Diagrams
[Randal Bryant’86]

A short review
If-Then-Else Normal Form

Definition

A boolean expression is in If-Then-Else normal form (INF) iff it is given by the following abstract syntax

\[ t, t_1, t_2 ::= 0 \mid 1 \mid x \rightarrow t_1, t_2 \]

where \( x \) ranges over boolean variables.

Example: \( x_1 \rightarrow (x_2 \rightarrow 1, 0), 0 \) (equivalent to \( x_1 \land x_2 \))

Boolean expressions in INF can be drawn as decision trees.
Shannon Expansion

Let $t$ be a boolean expression and $x$ a variable. We define boolean expressions

- $t[0/x]$ where every occurrence of $x$ in $t$ is replaced with 0, and
- $t[1/x]$ where every occurrence of $x$ in $t$ is replaced with 1.

**Shannon's Expansion Law**

Let $x$ be an arbitrary boolean variable. Any boolean expressions $t$ is equivalent to

$$x \rightarrow t[1/x], t[0/x].$$

**Corollary**

For any boolean expression there is an equivalent one in \text{INF}.
Binary Decision Trees

\[(x_1 \Leftrightarrow y_1) \land (x_2 \Leftrightarrow y_2)\]

- Variable is set to 0
- Variable is set to 1

Each path determines a partial (set of) truth assignments.

Result of the boolean expression under the given assignment found in value of terminal.
Orderedness & Redundant TESTS
Orderedness & Reducedness

x \lessdot y \quad x \lessdot z

\[ x \lessdot y \quad x \lessdot z \]
ROBDDs formally

A Binary Decision Diagram is a rooted, directed, acyclic graph \((V, E)\). \(V\) contains (up to) two terminal vertices, \(0, 1 \in V\). \(v \in V \setminus \{0, 1\}\) are non-terminal and has attributes \(\text{var}(v)\), and \(\text{low}(v), \text{high}(v) \in V\).

A BDD is ordered if on all paths from the root the variables respect a given total order.

A BDD is reduced if for all non-terminal vertices \(u, v,\)

1) \(\text{low}(u) \neq \text{high}(u)\)
2) \(\text{low}(u) = \text{low}(v), \text{high}(u) = \text{high}(v), \text{var}(u) = \text{var}(v)\) implies \(u = v\)
Reduced Ordered Binary Decision Diagrams

\((x_1 \Leftrightarrow y_1) \land (x_2 \Leftrightarrow y_2)\)
Ordering DOES matter

\[(x_1 \Leftrightarrow x_2) \land (x_3 \Leftrightarrow x_4) \land (x_5 \Leftrightarrow x_6) \land (x_7 \Leftrightarrow x_8)\]

\[x_1 < x_2 < \cdots < x_8\]
Canonicity of ROBDDs

\begin{align*}
    t_0 &= 0 \\
    t_1 &= 1 \\
    t_u &= x \rightarrow t_h, t_l, \text{ if } u \text{ is a node } (x, l, h)
\end{align*}

**Lemma 1 (Canonicity lemma)** For any function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ there is exactly one ROBDD $b$ with variables $x_1 < x_2 < \ldots < x_n$ such that

\[ t_b[v_1/x_1, \ldots, v_n/x_n] = f(v_1, \ldots, v_n) \]

for all $(v_1, \ldots, v_n) \in \mathbb{B}^n$.

**Consequences:**

- $b$ is a tautology, if and only if, $b = 1$
- $b$ is satisfiable, if and only if, $b \neq 0$
Build

Let $t$ be a boolean expression and $x_1 < x_2 < \cdots < x_n$.

Build($t, 1$) builds a corresponding ROBDD and returns its root.

\[
\text{Build}(t, i): \text{Node} = \\
\text{if } i > n \text{ then} \\
\quad \text{if } t \text{ is true then return 0 else return 1} \\
\text{else} \\
\quad \text{low := Build}(t[0/x_i], i + 1) \\
\quad \text{high := Build}(t[1/x_i], i + 1) \\
\quad \text{var := i} \\
\quad \text{return Makenode(var, low, high)} \\
\text{end if}
\]
APPLY operation

Apply\((op, b_1, b_2)\)

4: function \(app(u_1, u_2) = \)
6: \[ \text{if } u_1 \in \{0, 1\} \text{ and } u_2 \in \{0, 1\} \text{ then } res \leftarrow op(u_1, u_2) \]
7: \[ \text{else if } u_1 \in \{0, 1\} \text{ and } u_2 \geq 2 \text{ then} \]
8: \[ res \leftarrow \text{makenode}\left(\text{var}(u_2), app(u_1, \text{low}(u_2)), app(u_1, \text{high}(u_2))\right) \]
9: \[ \text{else if } u_1 \geq 2 \text{ and } u_2 \in \{0, 1\} \text{ then} \]
10: \[ res \leftarrow \text{makenode}\left(\text{var}(u_1), \text{app}(\text{low}(u_1), u_2), \text{app}(\text{high}(u_1), u_2)\right) \]
11: \[ \text{else if } \text{var}(u_1) = \text{var}(u_2) \text{ then} \]
12: \[ res \leftarrow \text{makenode}\left(\text{var}(u_1), \text{app}(\text{low}(u_1), \text{low}(u_2)), \text{app}(\text{high}(u_1), \text{high}(u_2))\right) \]
13: \[ \text{else if } \text{var}(u_1) < \text{var}(u_2) \text{ then} \]
14: \[ res \leftarrow \text{makenode}\left(\text{var}(u_1), \text{app}(\text{low}(u_1), u_2), \text{app}(\text{high}(u_1), u_2)\right) \]
15: \[ \text{else (*) var}(u_1) > \text{var}(u_2) (*) \]
16: \[ res \leftarrow \text{makenode}\left(\text{var}(u_2), \text{app}(u_1, \text{low}(u_2)), \text{app}(u_1, \text{high}(u_2))\right) \]
18: \[ \text{return } res \]
20: \[ b.\text{root} \leftarrow app(b_1.\text{root}, b_2.\text{root}) \]
22: \[ \text{return } b \]
APPLY example
APPLY operation with dynamic programming

```latex
\begin{align*}
\text{Apply}(\text{op}, b_1, b_2) & \\
4: & \quad \text{function } \text{app}(u_1, u_2) = \\
5: & \quad \quad \text{if } G(u_1, u_2) \neq \text{empty} \text{ then return } G(u_1, u_2) \\
6: & \quad \quad \text{else if } u_1 \in \{0, 1\} \text{ and } u_2 \in \{0, 1\} \text{ then } res \leftarrow \text{op}(u_1, u_2) \\
7: & \quad \quad \text{else if } u_1 \in \{0, 1\} \text{ and } u_2 \geq 2 \text{ then} \\
8: & \quad \quad \quad \quad res \leftarrow \text{makenode}(\text{var}(u_2), \text{app}(u_1, \text{low}(u_2)), \text{app}(u_1, \text{high}(u_2))) \\
9: & \quad \quad \text{else if } u_1 \geq 2 \text{ and } u_2 \in \{0, 1\} \text{ then} \\
10: & \quad \quad \quad \quad res \leftarrow \text{makenode}(\text{var}(u_1), \text{app}(\text{low}(u_1), u_2), \text{app}(\text{high}(u_1), u_2)) \\
11: & \quad \quad \text{else if } \text{var}(u_1) = \text{var}(u_2) \text{ then} \\
12: & \quad \quad \quad \quad res \leftarrow \text{makenode}(\text{var}(u_1), \text{app}(\text{low}(u_1), \text{low}(u_2)), \text{app}(\text{high}(u_1), \text{high}(u_2))) \\
13: & \quad \quad \text{else if } \text{var}(u_1) < \text{var}(u_2) \text{ then} \\
14: & \quad \quad \quad \quad res \leftarrow \text{makenode}(\text{var}(u_1), \text{app}(\text{low}(u_1), u_2), \text{app}(\text{high}(u_1), u_2)) \\
15: & \quad \quad \text{else } (* \text{var}(u_1) > \text{var}(u_2) *) \\
16: & \quad \quad \quad \quad res \leftarrow \text{makenode}(\text{var}(u_2), \text{app}(u_1, \text{low}(u_2)), \text{app}(u_1, \text{high}(u_2))) \\
17: & \quad G(u_1, u_2) \leftarrow res \\
18: & \quad \text{return } res \\
2: & \quad \text{forall } i \leq \text{max}(b_1), j \leq \text{max}(b_2) : G(i, j) \leftarrow \text{empty} \\
21: & \quad b.\text{root} \leftarrow \text{app}(b_1.\text{root}, b_2.\text{root}) \\
22: & \quad \text{return } b
\end{align*}
```
Other operations

Let $t$ be a boolean expression with its ROBDD representation.

The following operations can be done efficiently:

- **Restriction** $t[0/x_i]$ ($t[1/x_i]$): restricts the variable $x_i$ to 0 (1)
- **SatCount($t$)**: returns the number of satisfying assignments
- **AnySat($t$)**: returns some satisfying assignment
- **AllSat($t$)**: returns all satisfying assignments
- **Existential quantification** $\exists x_i \cdot t$: equivalent to $t[0/x_i] \lor t[1/x_i]$
- **Composition** $t[t'/x_i]$: equivalent to $t' \rightarrow t[1/x_i], t[0/x_i]$
Constraint Solving using BDDs
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4 x 4 Sudoku**

288 solutions!
Encoding

Boolean variables \( x_{i,j,k} \) for all \( i, j, k \in \{1,2,3,4\} \).

Idea:

\[ x_{i,j,k} = 1 \quad \text{if the number } k \text{ is in position } (i,j) \text{ in the solution} \]
\[ x_{i,j,k} = 0 \quad \text{otherwise} \]
Constraints

Precisely one value in each position \(i, j\):
\[
x_{1,j,1} + x_{i,j,2} + x_{i,j,3} + x_{i,j,4} = 1
\]
for each \(i, j\)

Each value \(k\) appears in each row \(i\) exactly ones:
\[
x_{i,1,k} + x_{i,2,k} + x_{i,3,k} + x_{i,4,k} = 1
\]
for each \(i, k\)

Each value \(k\) appears in each column \(j\) exactly ones:
\[
x_{1,j,k} + x_{2,j,k} + x_{3,j,k} + x_{4,j,k} = 1
\]
for each \(j, k\)

Each value \(k\) appears in each 2x2 box exactly ones:
\[
x_{1,1,k} + x_{1,2,k} + x_{2,1,k} + x_{2,2,k} = 1
\] (e.g.)
Solving Sudoku
ROBDDs and Verification
[... McMillan’90, ... VVS’97]
ROBDD encoding of transition system

Encoding of states using binary variables (here $x_1$ and $x_2$).

Encoding of transition relation using source and target variables (here $x_1$, $x_2$, $y_1$, and $y_2$)

$$\text{Trans}(x_1,x_2,y_1,y_2) :=$$
$$\text{(!x}_1 \land \text{!x}_2 \land \text{!y}_1 \land y_2) + \text{(!x}_1 \land \text{!x}_2 \land y_1 \land y_2) + \text{x}_1 \land \text{!x}_2 \land \text{!y}_1 \land y_2 + \text{x}_1 \land \text{!x}_2 \land y_1 \land y_2 + \text{x}_1 \land \text{x}_2 \land y_1 \land \text{!y}_2;$$
ROBDD representation (cont.)

Trans(x1,x2,y1,y2) := 
!x1 & !x2 & !y1 & y2 
+ !x1 & !x2 & y1 & y2 
+ x1 & !x2 & !y1 & y2 
+ x1 & !x2 & y1 & y2 
+ x1 & x2 & y1 & !y2;
ROBDD for parallel composition

Asynchronous composition

\[
\text{Trans}(x, y, u, v) = \left( \text{ATrans}(x, y) \land v = u \right) + \left( \text{BTrans}(u, v) \land y = x \right)
\]

Synchronous composition

\[
\text{Trans}(x, y, u, v) = \left( \text{ATrans}(x, y) \land \text{BTrans}(u, v) \right)
\]

Which ordering to choose?
Ordering?

45 nodes
x1,x2,u1,u2, y1,y2 ,v1,v2

23 nodes
x1,x2,y1,y2,u1,u2,v1,v2

20 nodes
x1,y1,x2,y2,u1,v1,u2,v2

Polynomial size BDDs guaranteed in size of argument BDDs [Enders,Filkorn, Taubner’91]
Reachable States

Reach(x) := Init(x);

REPEAT
  Old(x) := Reach(x);
  New(y) := Exists x. (Reach(x) & Trans(x, y));
  Reach(x) := Old(x) + New(x)
UNTIL Old(x) = Reach(x)

Relational Product:
May be constructed without building intermediate (often large) &-BDD.
A MUTEX Algorithm
Clarke & Emerson

P1 :: while True do
    T1 : wait(turn=1)
    C1 : turn:=0
endwhile
||
P2 :: while True do
    T2 : wait(turn=0)
    C2 : turn:=1
endwhile

Mutual Exclusion Program
Global Transition System
A MUTEX Algorithm
Clarke & Emerson

vars x1 x2;
vars y1 y2;
vars u1 u2;
vars v1 v2;
vars t s;

ATrans := (!x1 & !x2 & !y1 & y2 & (s=t))
  + (!x1 & x2 & !y1 & y2 & !t & !s)
  + (!x1 & x2 & y1 & !y2 & t & s)
  + (x1 & !x2 & !y1 & !y2 & !s);

BTrans := (!u1 & !u2 & !v1 & v2 & (s=t))
  + (!u1 & u2 & !v1 & v2 & t & s)
  + (!u1 & u2 & v1 & !v2 & !t & !s)
  + (u1 & !u2 & !v1 & !v2 & s);

TT := (ATrans & (u1=v1) & (u2=v2))
  + (BTrans & (x1=y1) & (x2=y2));
BDDs for Transition Relations

ATrans

TT
Reachable States

Reach(x) := Init(x);
REPEAT
    Old(x) := Reach(x);
    New(y) := Exists x. (Reach(x) & Trans(x, y));
    Reach(x) := Old(x) + New(x)
UNTIL Old(x) = Reach(x)
Reachable States

Reach(x) := Init(x);
REPEAT
  Old(x) := Reach(x);
  New(y) := Exists x. (Reach(x) & Trans(x, y));
  Reach(x) := Old(x) + New(x)
UNTIL Old(x) = Reach(x)
Reachable States

Reachable States

Reach(x) := Init(x);
REPEAT
  Old(x) := Reach(x);
  New(y) := Exists x. (Reach(x) & Trans(x, y));
  Reach(x) := Old(x) + New(x)
UNTIL Old(x) = Reach(x)

Reach &
  x1 & !x2 &
  u1 & !u2
Bisimulation

vars $x$ ($y$)

vars $u$ ($v$)

Bis($x,u$) := 1;
REPEAT
    Old($x,u$) := Bis($x,u$);
    Bis($x,u$) :=
        Forall $y$. Trans($x,y$) =>
            (Exists $v$. Trans($u,v$) & Bis($y,v$))
        &
        Forall $v$. Trans($u,v$) =>
            (Exists $y$. Trans($x,y$) & Bis($y,v$));
UNTIL Bis($x,u$) = Old($x,u$)
Bisimulation (cont.)

3 equivalence classes
= 6 pairs in final bisimulation
Model Checking

define Trans(x1, x2, y1, y2) :=
  !x1 & !x2 & !y1 & y2
+ !x1 & !x2 & y1 & y2
+ ............ ;

define P(x1, x2) := !x1 & !x2
+ !x1 & x2
+ x1 & !x2;

define Q(x1, x2) := ........ ;
Model Checking

 Exists y1, y2.
 Trans(x1, x2, y1, y2) &
 P(y1, y2);
Model Checking

Exists y1, y2.
   Trans(x1, x2, y1, y2) &
   P(y1, y2);
Model Checking

For all $y_1, y_2$.
$\text{Trans}(x_1, x_2, y_1, y_2) \Rightarrow P(y_1, y_2)$;
Model Checking

For all $y_1, y_2$.
\[ \text{Trans}(x_1, x_2, y_1, y_2) \implies P(y_1, y_2); \]
Model Checking

\[ \text{AG } P \]

max fixpoint

\[ \mathbf{A}(x_1, x_2) = P(x_1, x_2) \& \]
\[ \text{Forall } y_1, y_2. \]
\[ \text{Trans}(x_1, x_2, y_1, y_2) \Rightarrow \]
\[ \mathbf{A}(y_1, y_2); \]
Model Checking

\[ \text{AG } P \]

\[ A(x_1, x_2) = P(X_1, x_2) \land \forall y_1, y_2. \text{Trans}(x_1, x_2, y_1, y_2) \Rightarrow A(y_1, y_2); \]

\[ \text{max fixpoint} \]
Model Checking

\[ A( P \text{ UNTIL } Q ) \]

\[
U(x_1, x_2) = Q(x_1, x_2) + \\
\{ P(x_1, x_2) \land \forall y_1, y_2. Trans(x_1, x_2, y_1, y_2) \Rightarrow U(y_1, y_2) \};
\]
Model Checking

\[ A( P \text{ UNTIL } Q ) \]

\[ U(x_1, x_2) = Q(X_1, x_2) + \{ P(x_1, x_2) \& \]
\[ \quad \text{Forall } y_1, y_2. \]
\[ \quad \text{Trans}(x_1, x_2, y_1, y_2) \Rightarrow \]
\[ \quad U(y_1, y_2) \]
\[ ]; \]
Partitioned Transition Relation

Relational Product

\[ T(xu, yv) = \]

Asynchronous

\[ \exists yv. (T(xu, yv) \land S(yv)) \]

Synchronous

\[ \begin{align*}
T(xu, yv) &= \exists y. ATrans(x, y) \land S(yu) \\
&+ \exists v. BTrans(u, v) \land S(xv)
\end{align*} \]

\[ \begin{align*}
T(xu, yv) &= \exists y. ATrans(x, y) \\
&\quad \land (\exists v. BTrans(u, v) \land S(yv))
\end{align*} \]

\[ \begin{align*}
T(xu, yv) &= \exists yv. ATrans(x, y) \\
&\quad \land BTrans(u, v) \land S(yv)
\end{align*} \]
Beologic’s Products: salesPLUS visualSTATE

1980-95: Independent division of B&O
1995+: Independent company
    B&O, 2M Invest,
    Danish Municipal Pension Ins. Fund
1998: BAAN
2000: IAR Systems A/S

Customers
    ABB
    B&O
    Daimler-Benz
    Ericson DIAX
    ESA/ESTEC
    FORD
    Grundfos
    LEGO
    PBS
    Siemens …… (approx. 200)

Verification Problems:
    • 1,400 components
    • $10^{400}$ states

Our techniques has reduced verification by an order of magnitude
(from 14 days to 6 sec)

• Embedded Systems
• Simple Model
• Verification of Std. Checks
• Explicit Representation
  (STATEEXPLOSION)
• Code Generation
visualSTATE

Embedded World
Nürnberg, 2005
Control Programs
A Train Simulator, visualSTATE (VVS)

1421 machines
11102 transitions
2981 inputs
2667 outputs
3204 local states
Declare state sp.: $10^{476}$

"Ideal" presentation: 1 bit/state will clearly NOT work!
## Experimental Breakthroughs

**Patented**

<table>
<thead>
<tr>
<th>System</th>
<th>Mach.</th>
<th>State Space</th>
<th>Checks</th>
<th>Visual ST</th>
<th>St-of-Art</th>
<th>ComBack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCR</td>
<td>7</td>
<td>$10^5$</td>
<td>1279</td>
<td>50</td>
<td>&lt;1</td>
<td>&lt;1</td>
</tr>
<tr>
<td>JVC</td>
<td>8</td>
<td>$10^4$</td>
<td>352</td>
<td>22</td>
<td>&lt;1</td>
<td>&lt;1</td>
</tr>
<tr>
<td>HI-FI</td>
<td>9</td>
<td>$10^7$</td>
<td>1416384</td>
<td>120</td>
<td>1200</td>
<td>1.0</td>
</tr>
<tr>
<td>Motor</td>
<td>12</td>
<td>$10^7$</td>
<td>34560</td>
<td>123</td>
<td>32</td>
<td>&lt;1</td>
</tr>
<tr>
<td>AVS</td>
<td>12</td>
<td>$10^7$</td>
<td>1438416</td>
<td>173</td>
<td>3780</td>
<td>6.7</td>
</tr>
<tr>
<td>Video</td>
<td>13</td>
<td>$10^8$</td>
<td>1219440</td>
<td>122</td>
<td>---</td>
<td>1.1</td>
</tr>
<tr>
<td>Car</td>
<td>20</td>
<td>$10^{11}$</td>
<td>9.2 $10^9$</td>
<td>83</td>
<td>---</td>
<td>3.8</td>
</tr>
<tr>
<td>N6</td>
<td>14</td>
<td>$10^{10}$</td>
<td>6399552</td>
<td>443</td>
<td>---</td>
<td>32.3</td>
</tr>
<tr>
<td>N5</td>
<td>25</td>
<td>$10^{12}$</td>
<td>5.0 $10^{10}$</td>
<td>269</td>
<td>---</td>
<td>56.2</td>
</tr>
<tr>
<td>N4</td>
<td>23</td>
<td>$10^{13}$</td>
<td>3.7 $10^8$</td>
<td>132</td>
<td>---</td>
<td>622</td>
</tr>
<tr>
<td>Train1</td>
<td>373</td>
<td>$10^{136}$</td>
<td>---</td>
<td>1335</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Train2</td>
<td>1421</td>
<td>$10^{476}$</td>
<td>---</td>
<td>4708</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

Machine: 166 MHz Pentium PC with 32 MB RAM

---: Out of memory, or did not terminate after 3 hours.
Experimental Breakthroughs
Patented

<table>
<thead>
<tr>
<th>System</th>
<th>Mach.</th>
<th>State Space</th>
<th>Checks</th>
<th>Visual ST</th>
<th>Strength</th>
<th>ComBack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Declared</td>
<td>Reach</td>
<td></td>
<td>Sec</td>
<td>MB</td>
</tr>
<tr>
<td>VCR</td>
<td>7</td>
<td>$10^5$</td>
<td>1279</td>
<td>50</td>
<td>---</td>
<td>7</td>
</tr>
<tr>
<td>JVC</td>
<td>8</td>
<td>$10^4$</td>
<td>352</td>
<td>---</td>
<td>---</td>
<td>6</td>
</tr>
<tr>
<td>HI - FI</td>
<td>9</td>
<td>$10^7$</td>
<td>1416384</td>
<td>---</td>
<td>---</td>
<td>6</td>
</tr>
<tr>
<td>Motor</td>
<td>12</td>
<td>$10^7$</td>
<td>120</td>
<td>1200</td>
<td>6.7</td>
<td>6</td>
</tr>
<tr>
<td>AVS</td>
<td>12</td>
<td>$10^7$</td>
<td>32</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Video</td>
<td>13</td>
<td>$10^7$</td>
<td>43</td>
<td>1.1</td>
<td>6</td>
<td>1.5</td>
</tr>
<tr>
<td>Car</td>
<td>20</td>
<td>$10^7$</td>
<td>443</td>
<td>3.8</td>
<td>9</td>
<td>1.8</td>
</tr>
<tr>
<td>N6</td>
<td>14</td>
<td>$10^7$</td>
<td>269</td>
<td>32.3</td>
<td>7</td>
<td>218</td>
</tr>
<tr>
<td>N5</td>
<td>25</td>
<td>$10^7$</td>
<td>132</td>
<td>56.2</td>
<td>7</td>
<td>9.1</td>
</tr>
<tr>
<td>N4</td>
<td>23</td>
<td>$10^7$</td>
<td>132</td>
<td>622</td>
<td>7</td>
<td>6.3</td>
</tr>
<tr>
<td>Train1</td>
<td>373</td>
<td>$10^{13}$</td>
<td>---</td>
<td>1335</td>
<td>---</td>
<td>25.9</td>
</tr>
<tr>
<td>Train2</td>
<td>1421</td>
<td>$10^{13}$</td>
<td>---</td>
<td>4708</td>
<td>---</td>
<td>739</td>
</tr>
</tbody>
</table>

Machine: 166 MHz Pentium PC with 32 MB RAM
---: Out of memory, or did not terminate after 3 hours.

Our technique have reduced verification time by several orders of magnitude (eg. From 14 days to 6 sec)
Compositional Backwards Reachability

[ TACAS’98 ]
Example
The Small Train

Train
Gate
Signal

STOP
UP
DOWN
GR
RED

BW
FW

l, t
r
t
l, t
r
t
l

DOWNDOWN
RED
u
re
g
d

57
State-Event Model

Syntax

\[ M_i = (S_i, s_i^0, T_i) \]

where

\[ T_i \subseteq S_i \times E \times G_i \times M(O) \times S_i \]

Semantics

\[(s_1, \ldots, s_n) - e, \cup o_i \Rightarrow (s'_1, \ldots, s'_n)\]

iff

\[ s_i - e, g_i, o_i \Rightarrow s'_i \text{ with } g_i(s_1, \ldots, s_n) = \text{true} \]

or

\[ s_i = s'_i \text{ and } o_i = \emptyset \text{ and } \forall i \leq n, \text{ whenever } s_i - e, g_i \Rightarrow \text{ then } g_i(s_1, \ldots, s_n) = \text{false} \]
Small Train (cont)
Small Train (cont)
Generic Checks

**visualSTATE** offers checks for a number of predefined properties:

- Reachability of states
- Firing of transitions
- Input without interpretation
- Output without generation
- Conflicting Rules
- Local Deadlock
- Global Deadlock

Not a single CHECK but several thousands!
Guard dependencies

Let

\[ g_1, g_2, g_3, \ldots, g_N \ (N \text{ big}) \]

be the guards we want to show reachable

If

\[ g_i \Rightarrow g_j \quad (\text{e.g. } g_i=\text{DOWN} \land \text{UP}, g_j=\text{DOWN}) \]

then it suffices to show that \( g_i \) reachable, i.e. there is a reachable global state satisfying \( g_i \).

Sort \( g_1, g_2, g_3, \ldots, g_N \) according to size and check only if NOT implied by a previously shown reachable guard

\[ \rightarrow \]

40-70-90 % reduction.
Machine Dependencies

A guard \( g \) in machine \( M_i \) that depends on/references to a state in \( M_j \) introduces a dependency from \( M_i \) to \( M_j \).

Backwards statespace iterations can be restricted to dependency closed sets, e.g. \( \text{DC}(\text{GATE}) = \{ \text{GATE}, \text{SIGNAL} \} \)
Compositional Backwards

Is FW reachable?

IDEA:
Compute backward reachable states as much as possible with minimal set of machines. Increase set of considered machines when necessary!

Consider TRAIN

STOP BW FW

TRAIN
Compositional Backwards

Is FW reachable?

IDEA:
Compute backward reachable states as much as possible with minimal set of machines. Increase set of considered machines when necessary!

Consider TRAIN

Ignoring INPUT event
Compositional Backwards

Is FW reachable?

IDEA:
Compute backward reachable states as much as possible with minimal set of machines. Increase set of considered machines when necessary!

Consider TRAIN

Ignoring INPUT event
Compositional Backwards

Include GATE!
Compositional Backwards

Include GATE!

Any state projecting to can reach a state projecting to

Ignoring transitions with source in

TRAIN
Compositional Backwards

Include SIGNAL!

Thus FW is reachable!
Hierarchical Systems

IDEA

Reuse already known reachability properties of superstates to conclude reachability of substates!
Experimental results