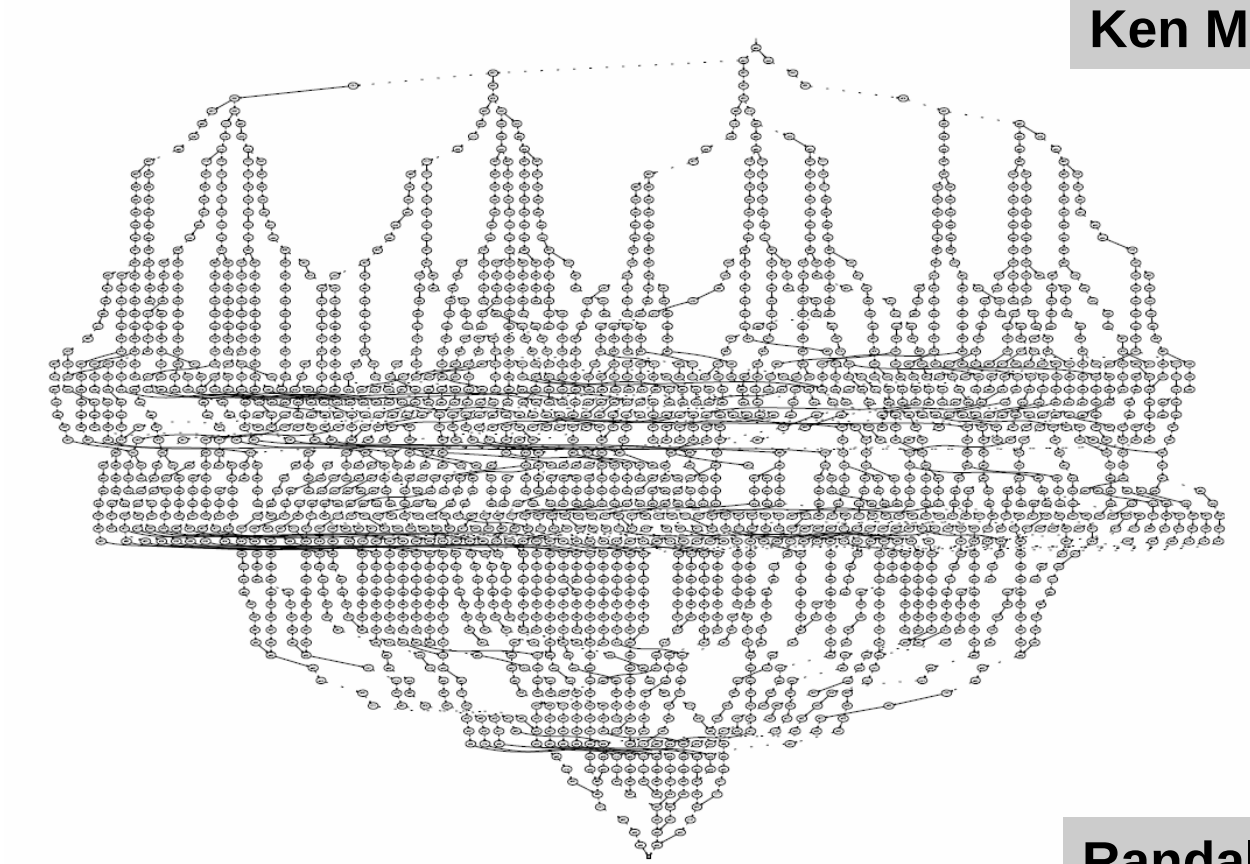
Symbolic Model Checking

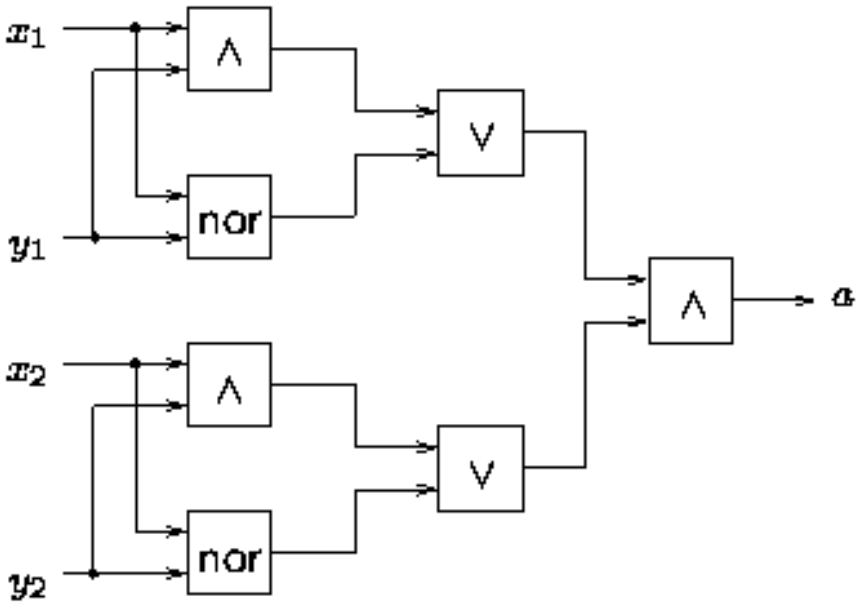
Ken McMillan ’90

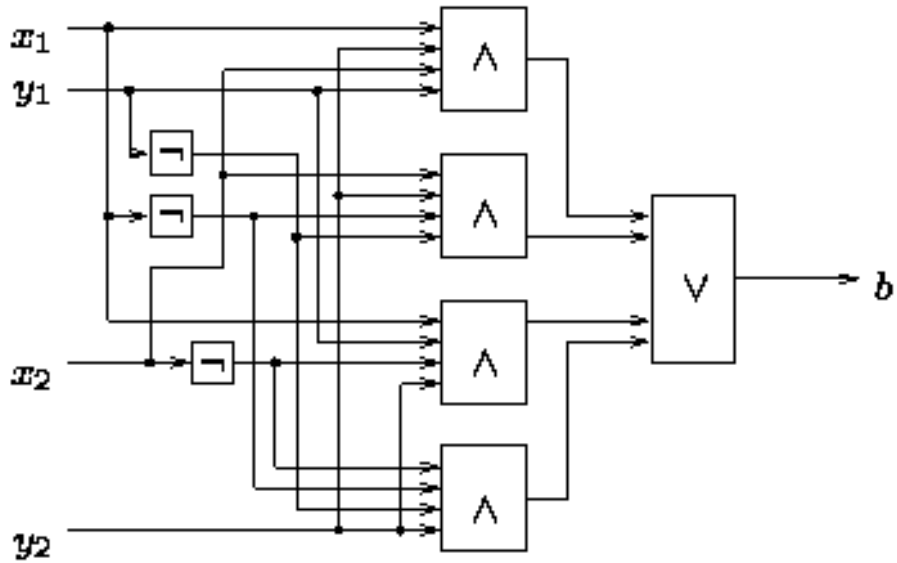


Randal Bryant ’86

Binary Decision Diagrams

Combinatorial Circuits

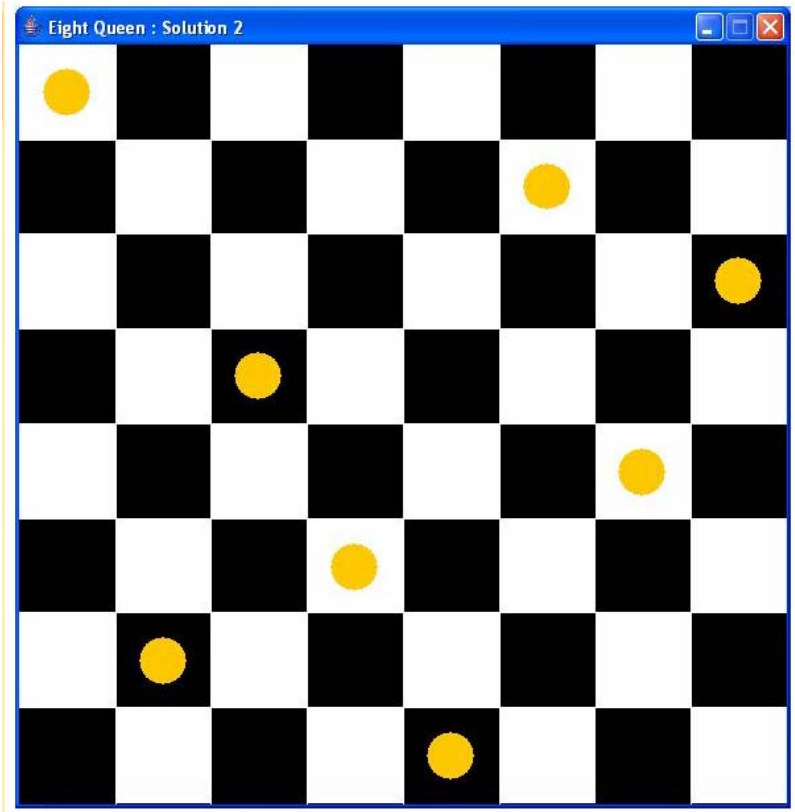




Combinatorial Problems

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Sudoku

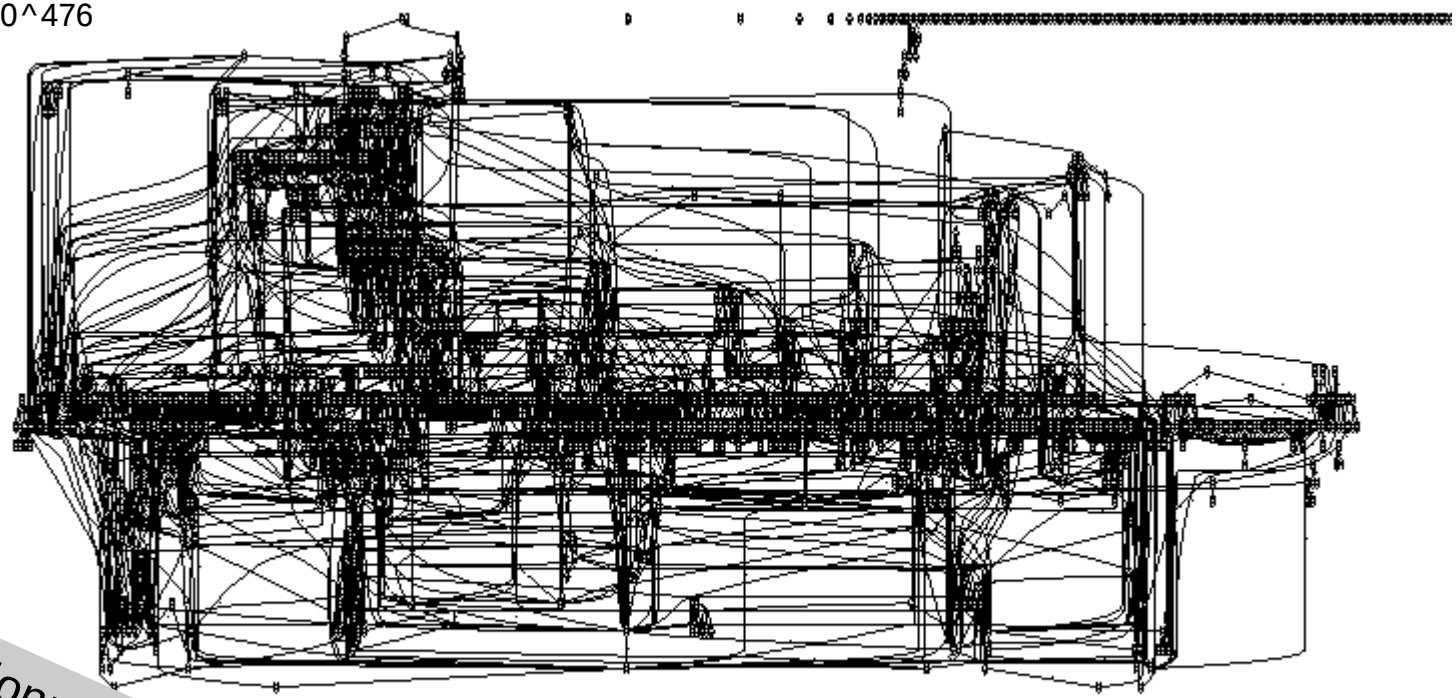


Eight Queen

Control Programs
A Train Simulator, visualSTATE (VVS)

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

BUGS ?



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

Reduced Ordered Binary Decision Diagrams [Bryant'86]

- Compact represetation of *boolean functions* allowing effective manipulation (satifiability, 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 $x_1 \wedge x_2$)

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

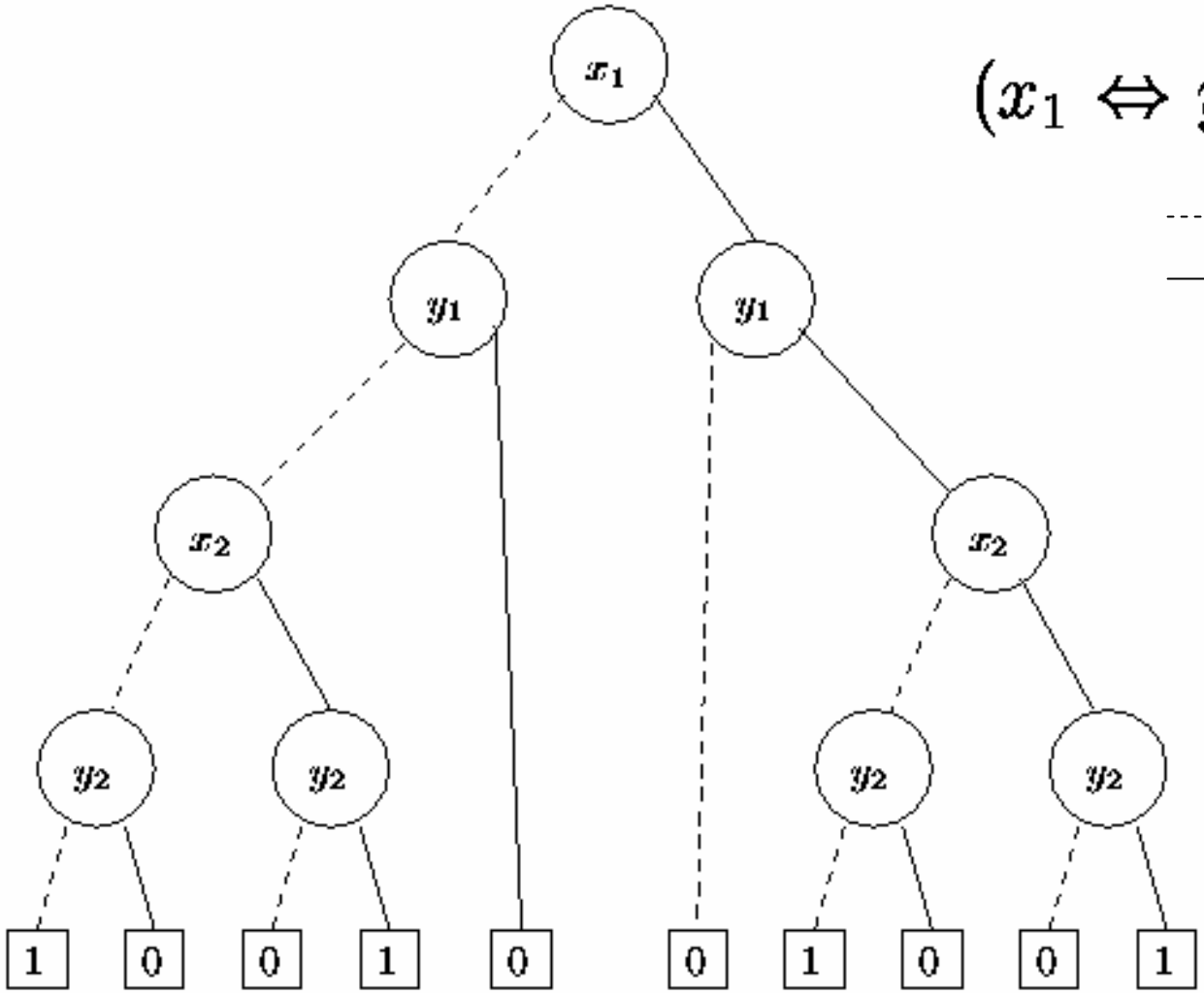
Let x be an arbitrary boolean variable. Any boolean expressions t is equivalent to

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

Corollary

For any boolean expression there is an equivalent one in INF.

Binary Decision *Trees*



$$(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$$

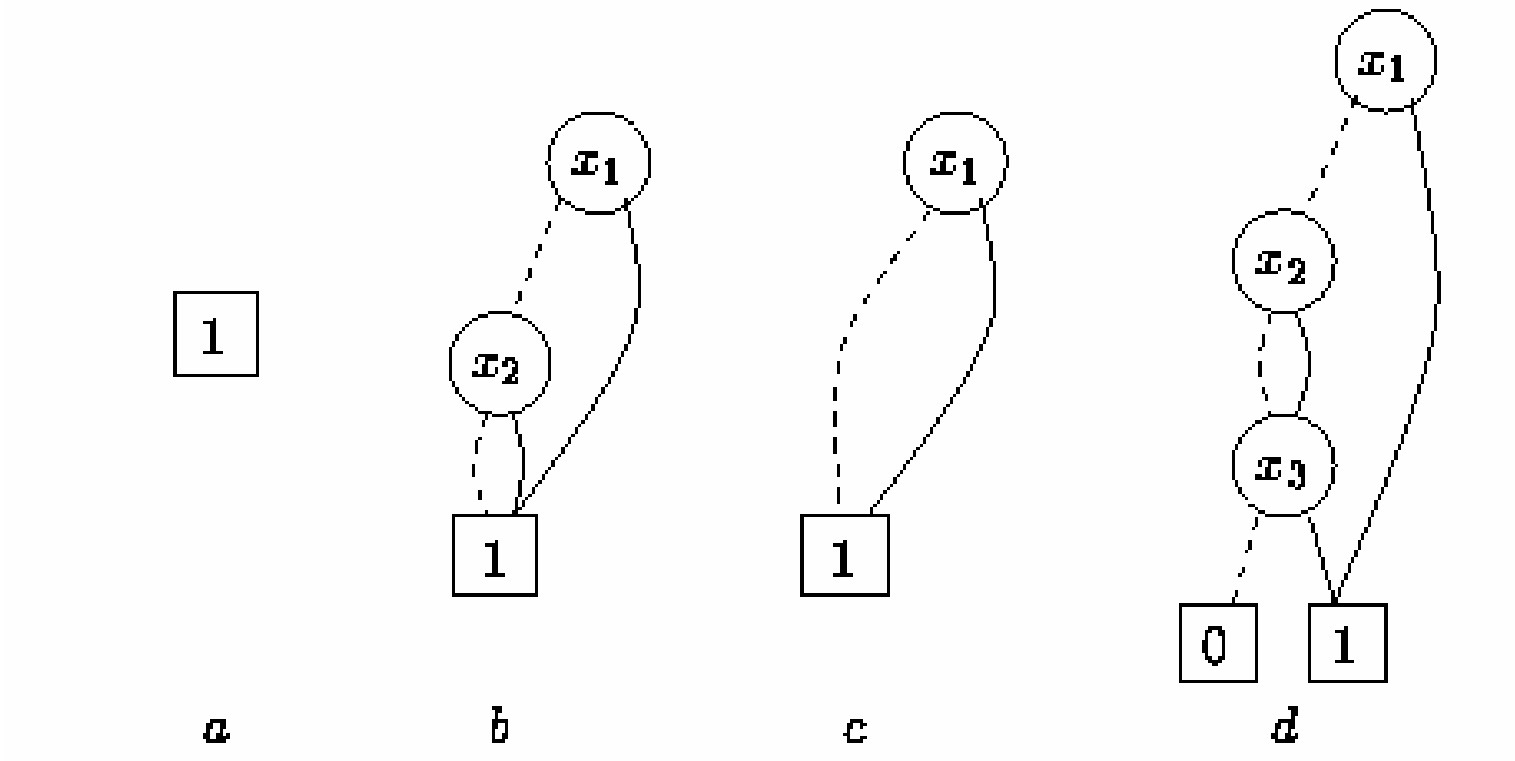
Variable is set to 0

Variable is set to 1

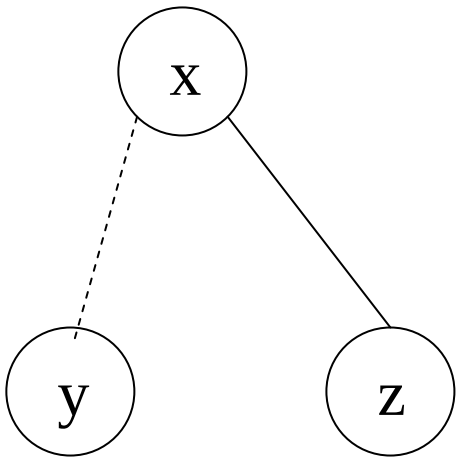
Each path determines a partial (set of) truth assignments.

Result of the boolean expression under the given assigment found in value of terminal.

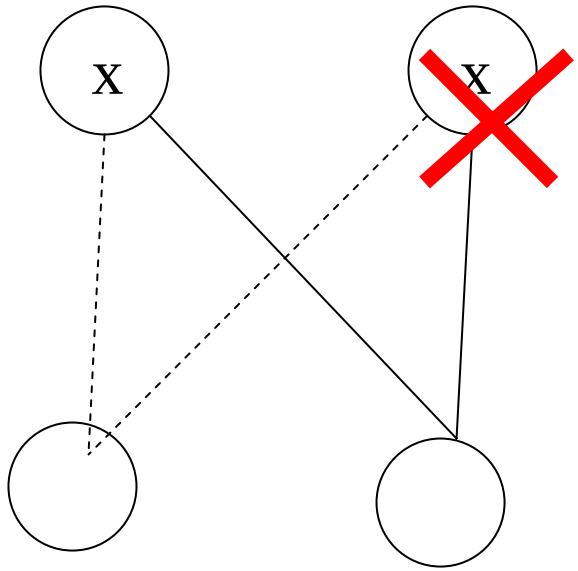
Orderedness & Redundant *TESTS*

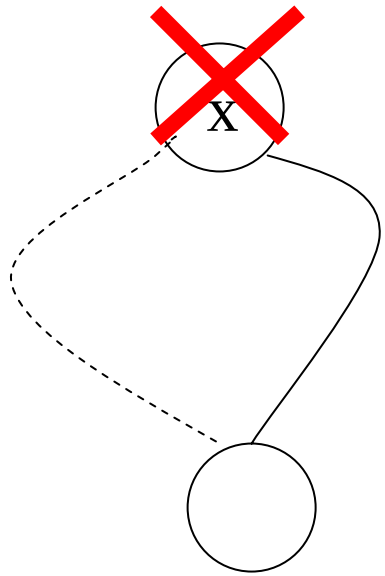


Orderedness & Reducedness



x<y x<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 $\mathit{var}(v)$, and $\mathit{low}(v), \mathit{high}(v) \in V$.

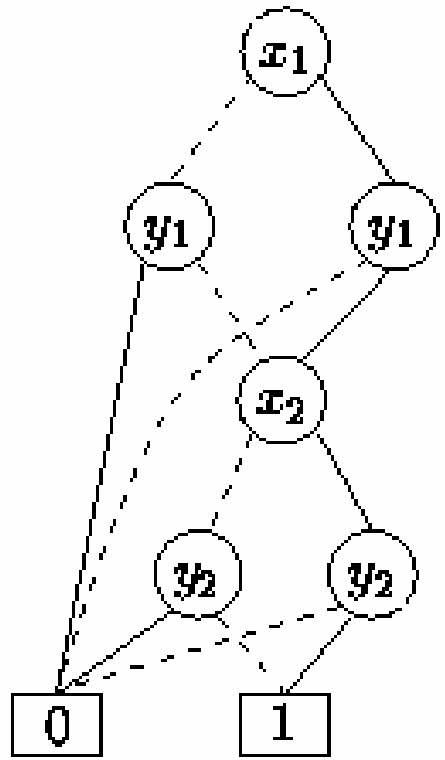
A BDD is *ordered* if on all paths from the root the variables respect a given total order.

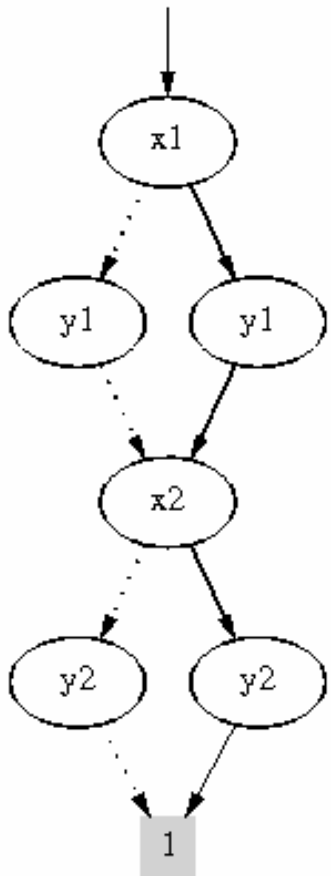
A BDD is *reduced* if for all non-terminal vertices u, v,

1) $\mathit{low}(u) \neq \mathit{high}(u)$
2) $\mathit{low}(u) = \mathit{low}(v), \mathit{high}(u) = \mathit{high}(v), \mathit{var}(u) = \mathit{var}(v)$ implies $u = v$

Reduced Ordered Binary Decision Diagrams

$$(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$$



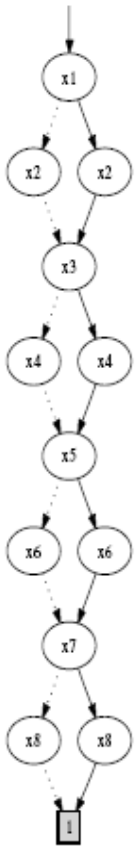


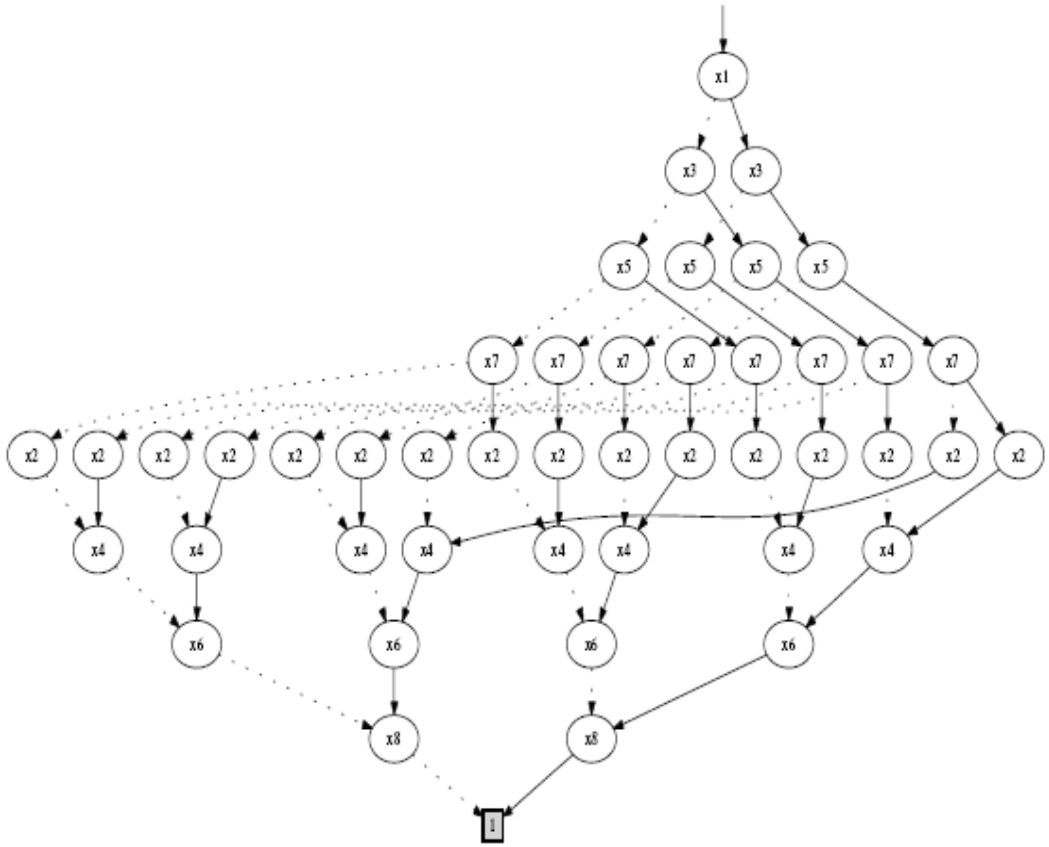
Iben

Edges to 0 implicit

Ordering DOES matter

$$(x_1 \Leftrightarrow x_2) \wedge (x_3 \Leftrightarrow x_4) \wedge (x_5 \Leftrightarrow x_6) \wedge (x_7 \Leftrightarrow x_8)$$





$$x_1 < x_2 < \cdots < x_8$$

$$x_1 < x_3 < x_5 < x_7 < x_2 < x_4 < x_6 < x_8$$

Canonicity of ROBDDs

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

Lemma 1 (Canonicity lemma) *For any function $f : \mathbb{B}^n \to \mathbb{B}$ 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, $b = \boxed{1}$
- b is satisfiable, if and only if, $b \neq \boxed{0}$

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.

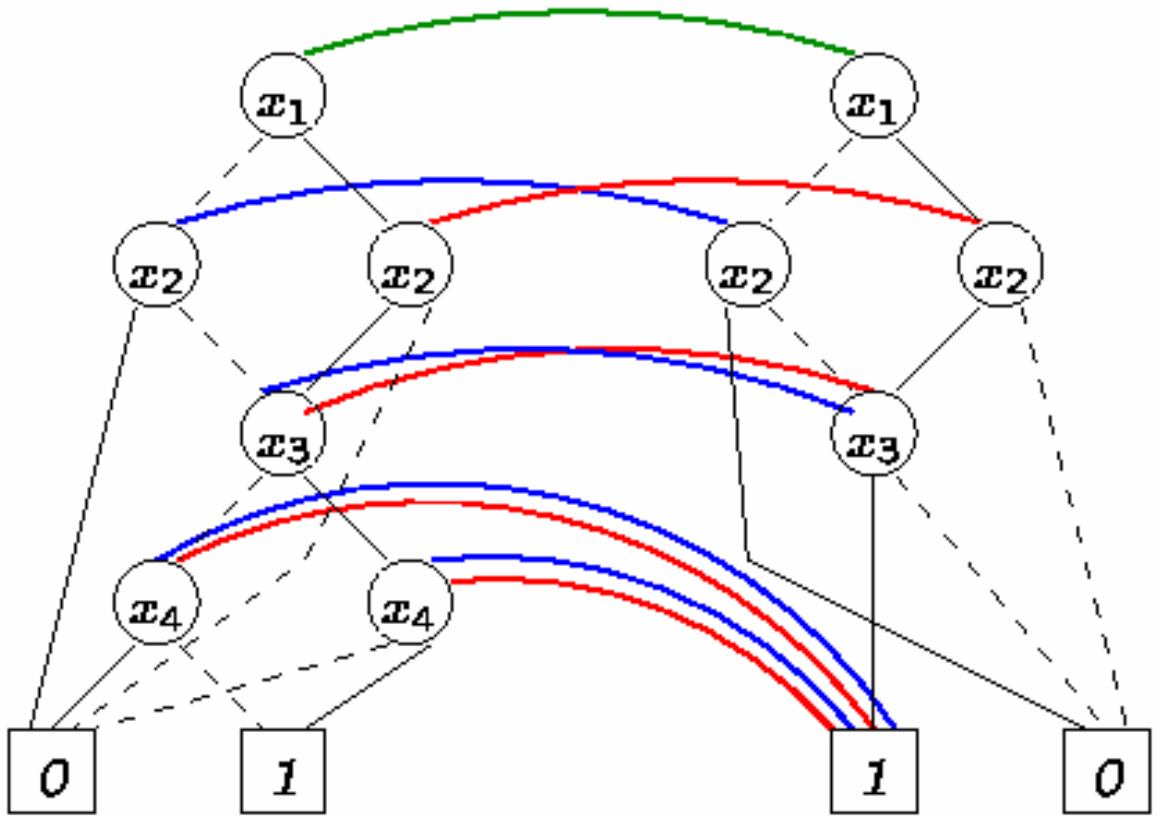
Build(t, i): Node $=$
if $i > n$ **then**
  if t is true **then return** 0 **else return** 1
else
  low := Build$(t[0/x_i], i + 1)$
  $high$:= Build$(t[1/x_i], i + 1)$
  $var := i$
  return Makenode$(var, low, high)$
end if

Complexity ??

APPLY operation

```
Apply(op, b1, b2)
4:    function app(u1, u2) =
6:              if u1 ∈ {0,1} and u2 ∈ {0,1} then res ← op(u1, u2)
7:              else if u1 ∈ {0,1} and u2 ≥ 2 then
8:                   res ← makenode(var(u2), app(u1, low(u2)), app(u1, high(u2))
9:              else if u1 ≥ 2 and u2 ∈ {0,1} then
10:                  res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
11:             else if var(u1) = var(u2) then
12:                  res ← makenode(var(u1), app(low(u1), low(u2)),
                                               app(high(u1), high(u2)))
13:             else if var(u1) < var(u2) then
14:                  res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
15:             else (* var(u1) > var(u2) *)
16:                  res ← makenode(var(u2), app(u1, low(u2)), app(u1, high(u2)))
18:             return res
20:
21:   b.root ← app(b1.root, b2.root)
22:   return b
```

APPLY example



APPLY operation
with dynamic programming

```
Apply(op, b1, b2)
4:   function app(u1, u2) =
5:       if G(u1, u2) ≠ empty then return G(u1, u2)
6:       else if u1 ∈ {0, 1} and u2 ∈ {0, 1} then res ← op(u1, u2)
7:            else if u1 ∈ {0, 1} and u2 ≥ 2 then
8:                 res ← makenode(var(u2), app(u1, low(u2)), app(u1, high(u2))
9:            else if u1 ≥ 2 and u2 ∈ {0, 1} then
10:                res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
11:           else if var(u1) = var(u2) then
12:                res ← makenode(var(u1), app(low(u1), low(u2)),
                                           app(high(u1), high(u2)))
13:           else if var(u1) < var(u2) then
14:                res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
15:           else (* var(u1) > var(u2) *)
16:                res ← makenode(var(u2), app(u1, low(u2)), app(u1, high(u2)))
17:           G(u1, u2) ← res
18:           return res
20:
2:   forall i ≤ max(b1), j ≤ max(b2) :  G(i, j) ← empty
21:  b.root ← app(b1.root, b2.root)
22:  return b
```

Other operations

Let t be a boolean expression with its ROBDD representation.

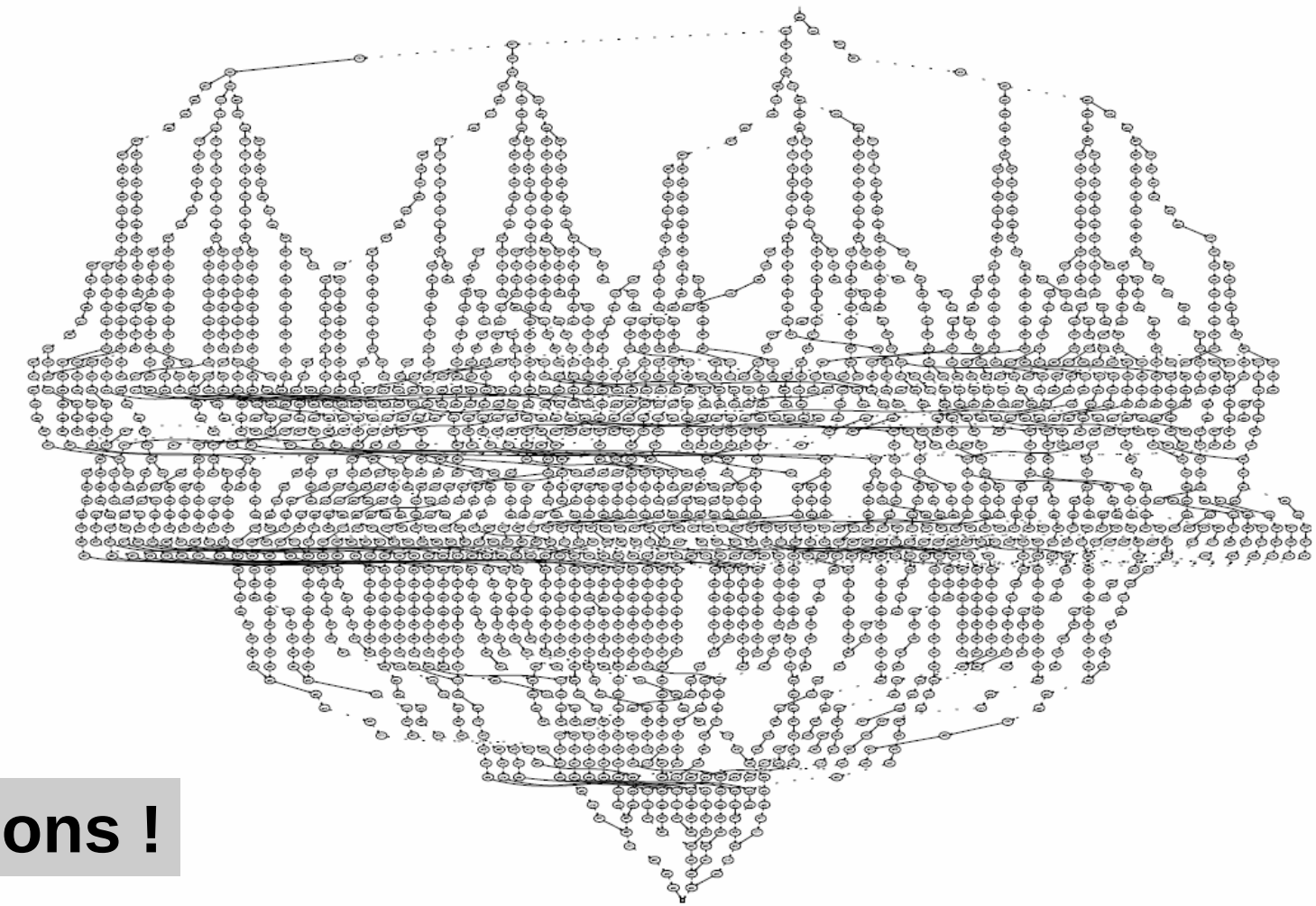
The following operations can be done efficiently:

- 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 $\exists x_i.t$: equivalent to $t[0/x_i] \vee t[1/x_i]$
- Composition $t[t'/x_i]$: equivalent to $t' \rightarrow t[1/x_i], t[0/x_i]$

Constraint Solving using BDDs

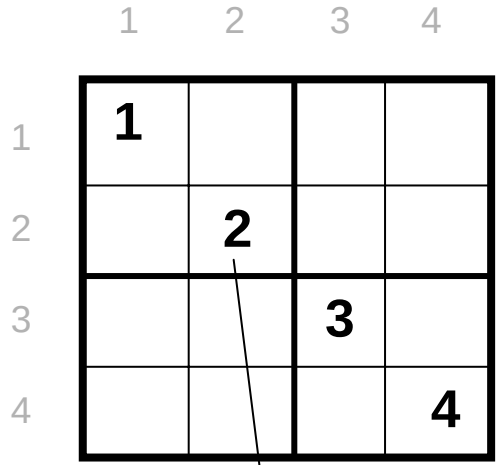
1			
	2		
		3	
			4

4 x 4 Sudoku



288 solutions !

Encoding



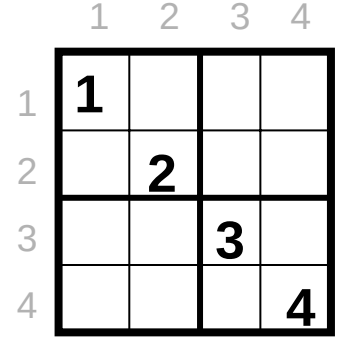
$x_{2,2,2} = 1$
$x_{4,4,4} = 1$
$x_{2,2,1} = 0$

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

Idea:

$x_{i,j,k} = 1$; if the number k is in position (i,j) in the solution

0 ; otherwise



Constraints

Precisely one value in **each position** i, j:

$$x_{1,j,1} + x_{i,j,2} + x_{i,j,3} + x_{i,j,4} = 1 \qquad \text{for each } i, j$$

Each value k appears in **each row** i exactly ones:

$$x_{i,1,k} + x_{i,2,k} + x_{i,3,k} + x_{i,4,k} = 1 \qquad \text{for each } i, k$$

Each value k appears in **each colomn** j exactly ones:

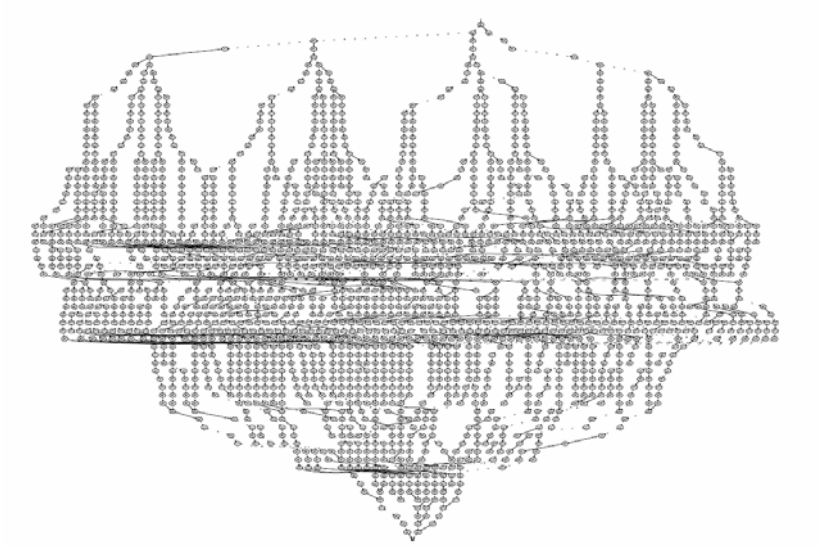
$$x_{1,j,k} + x_{2,j,k} + x_{3,j,k} + x_{4,j,k} = 1 \qquad \text{for each } j, k$$

Each value k appears in **each 2x2 box** exactly ones:

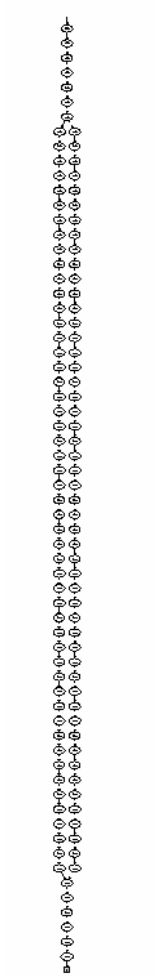
$$x_{1,1,k} + x_{1,2,k} + x_{2,1,k} + x_{2,2,k} = 1 \qquad \text{(e.g.)}$$

Solving Sudoku

	1	2	3	4
1				
2				
3				
4				



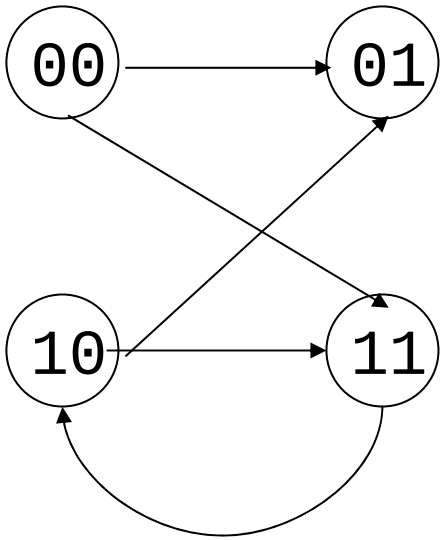
	1	2	3	4
1	**1**			
2		**2**		
3			**3**	
4				**4**



ROBDDs and Verification

[…,McMillan'90,…..,VVS'97]

ROBDD encoding of transition system

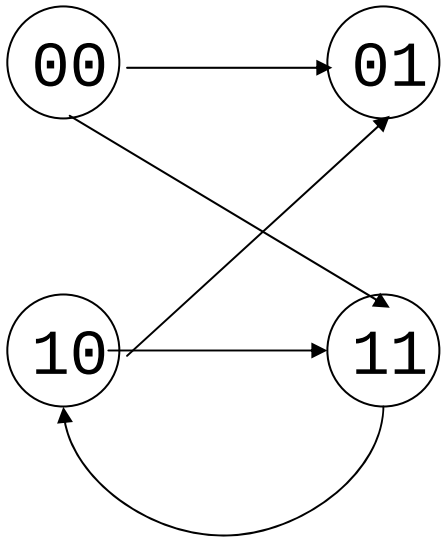


Encoding of states using binary variables (here `x1` and `x2`).

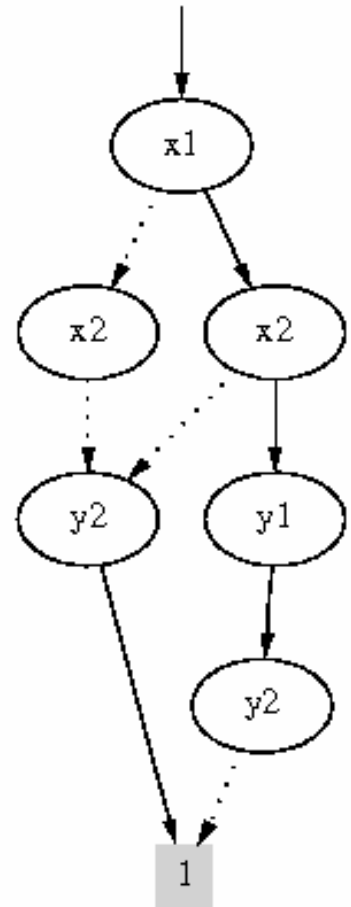
Encoding of transition relation using source and target variables (here `x1`, `x2`, `y1`, and `y2`)

```
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 representation (cont.)

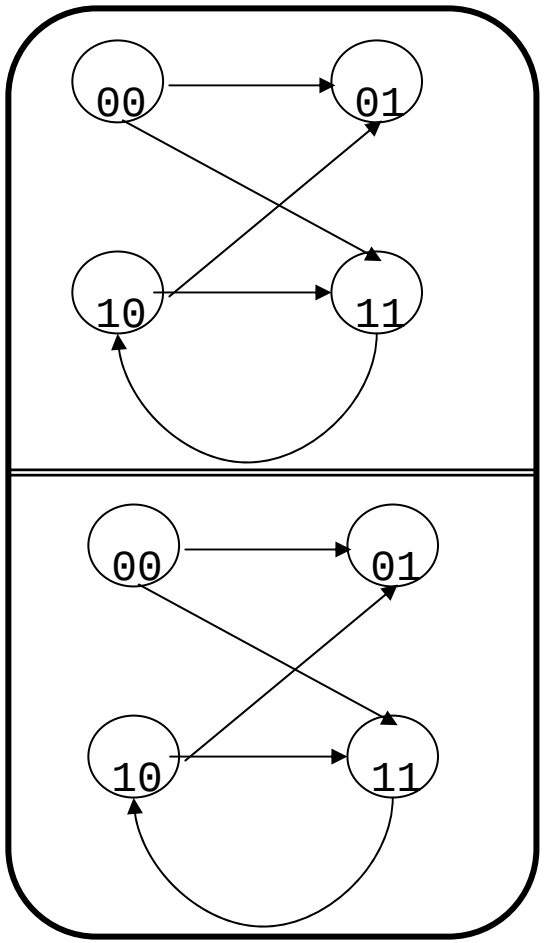



```
Trans(x1,x2,y1,y2):=
      !x1 & !x2 & !y1 & y2
    + !x1 & !x2 &  y1 & y2
    +  x1 & !x2 & !y1 & y2
    +  x1 & !x2 &  y1 & y2
    +  x1 &  x2 &  y1 & !y2;
```



ROBDD for parallel composition

ATrans(**x,y**)



BTrans(**u,v**)

Asynchronous composition

```
Trans(x,y,u,v) =
        (ATrans(x,y) & v=u)
      + (BTrans(u,v) & y=x)
```

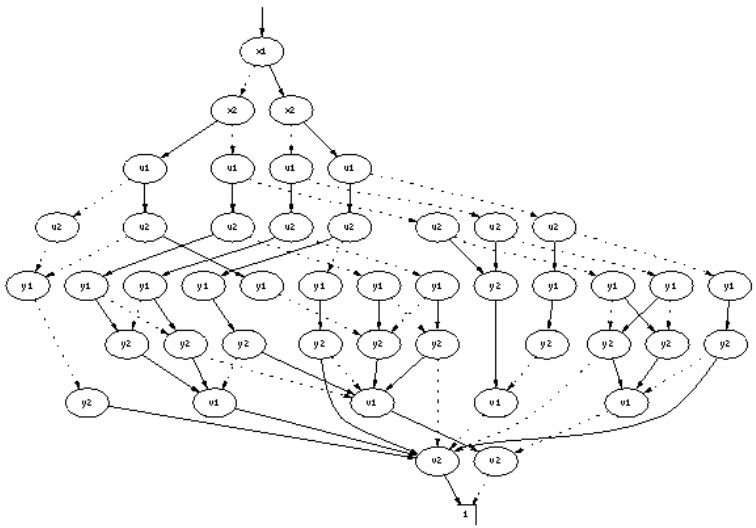
Synchronous composition

```
Trans(x,y,u,v) =
     (ATrans(x,y) & BTrans(u,v))
```

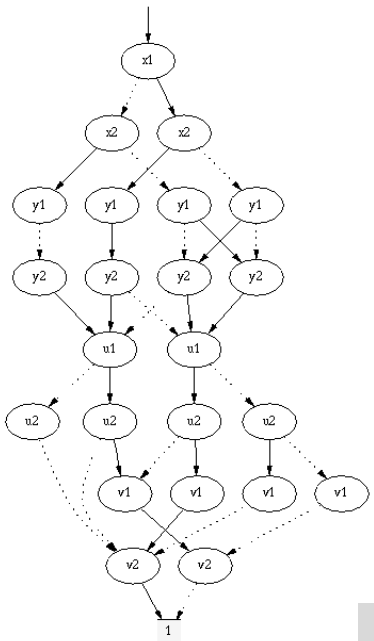
Which ordering to choose?

Ordering?

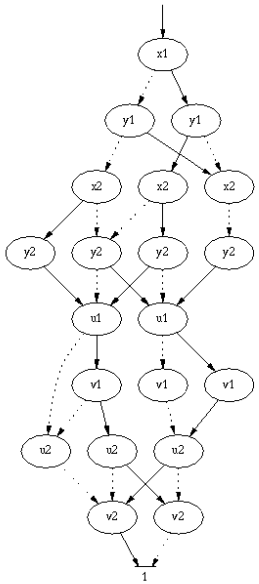
45 nodes
x1,x2,u1,u2, y1,y2 ,v1,v2



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



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

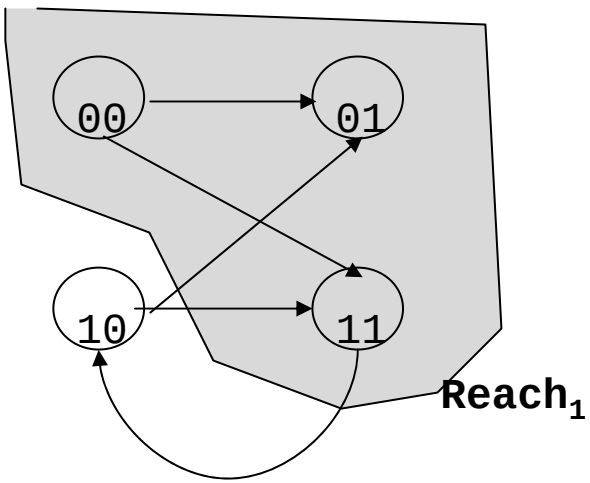


Polynomial size BDDs guaranteed in size of argument BDDs [Enders,Filkorn, Taubner'91]

Reachable States

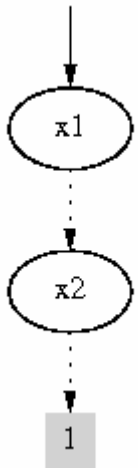
```
Reach(x) := Init(x);
REPEAT
   Old(x) := Reach(x);
   New(y) := Exists x.(Reach(x) & Trans(x,y));
   Reach(x) := Old(x) + New(x)
UNTIL Old(x) = Reach(x)
```

Relational Product:
May be constructed without building intermediate (often large) &-BDD.

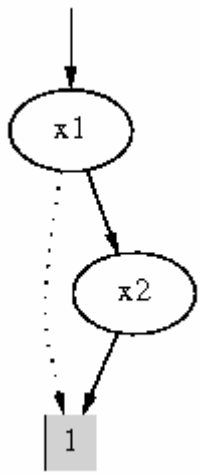


Reach$_1$

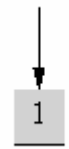
Reach$_0$



Reach$_1$



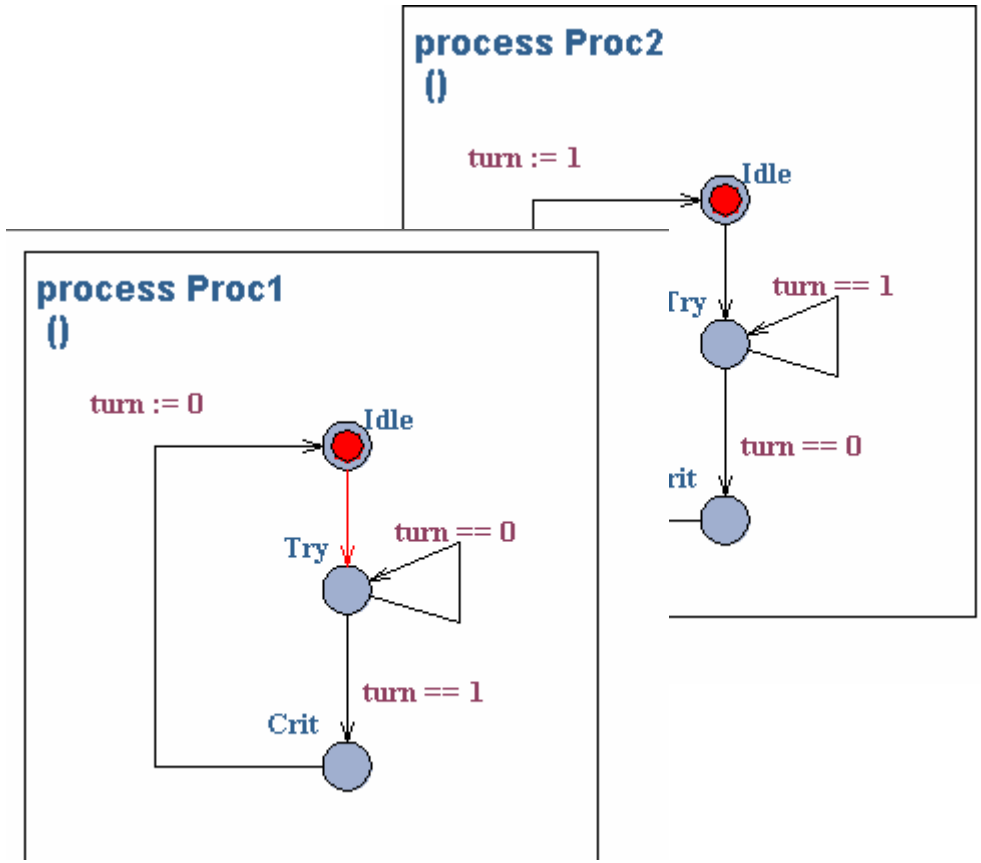
Reach$_2$



A MUTEX Algorithm

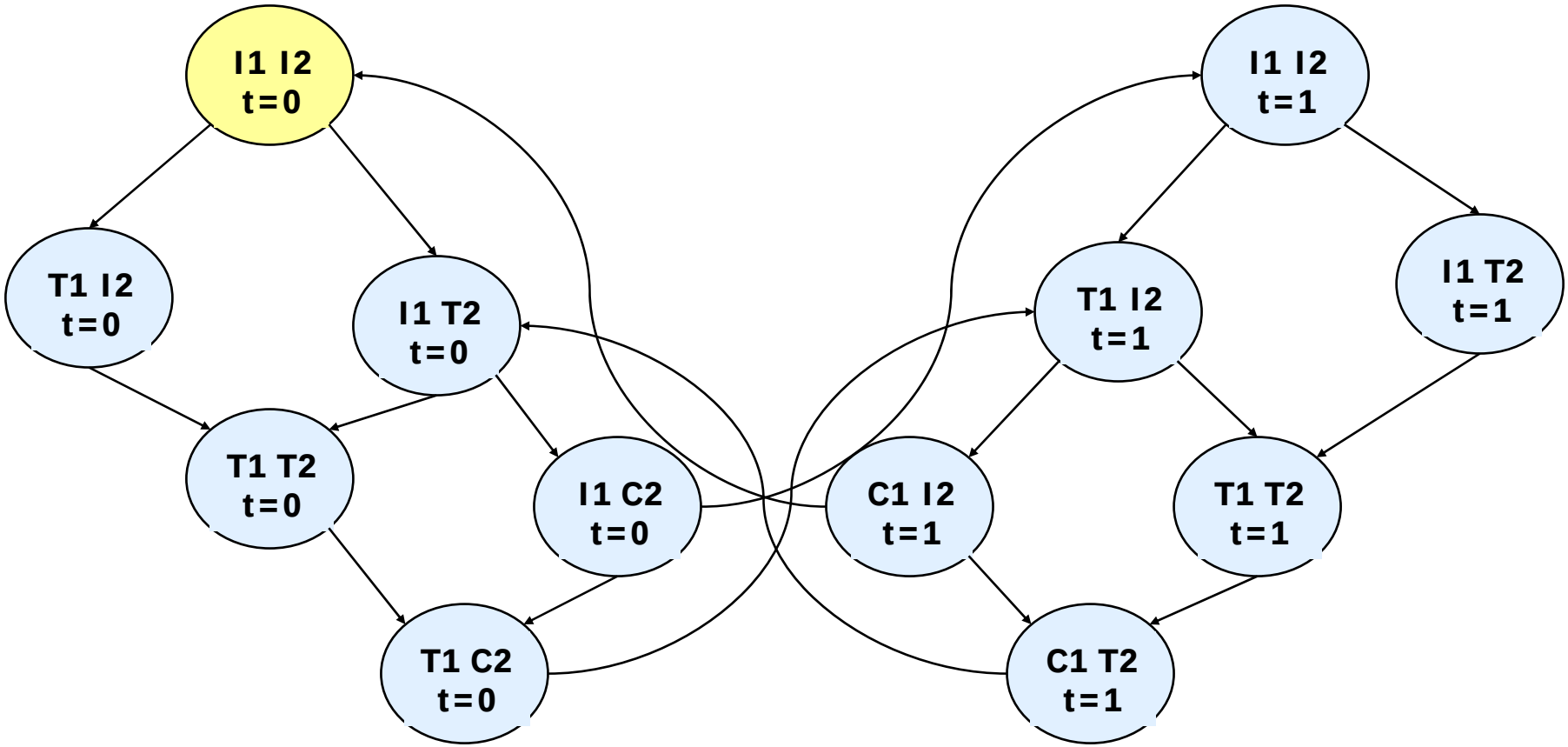
Clarke & Emerson

```
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

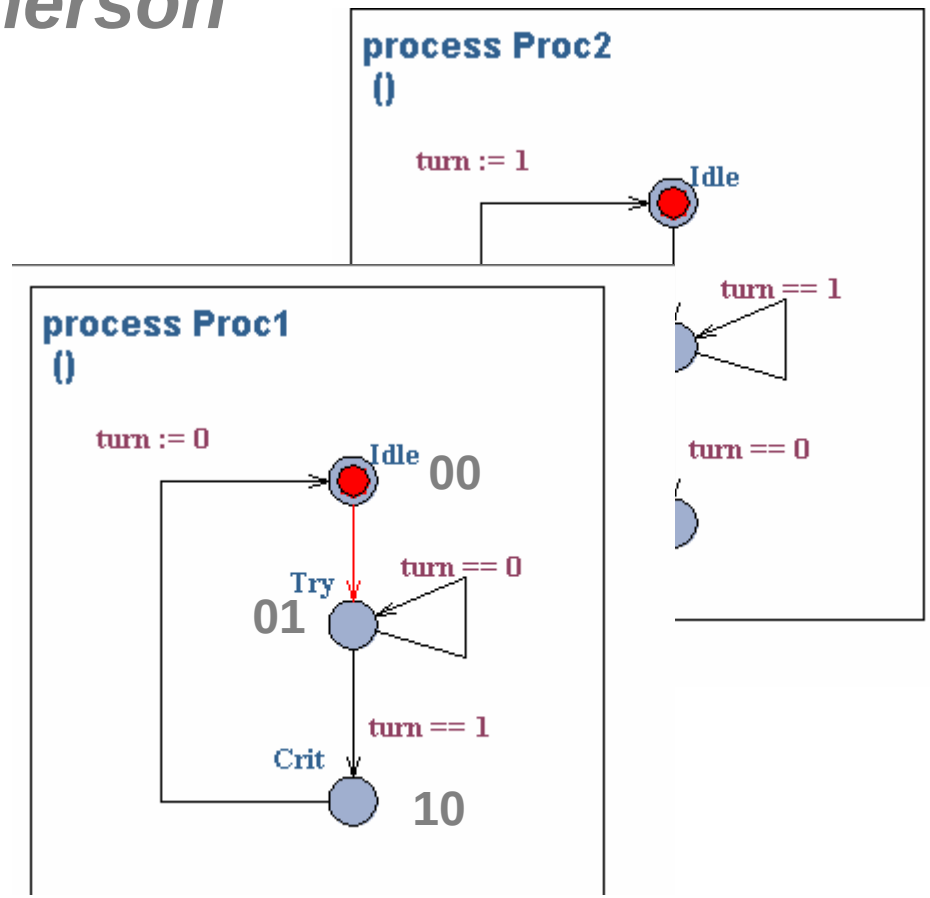
Global Transition System



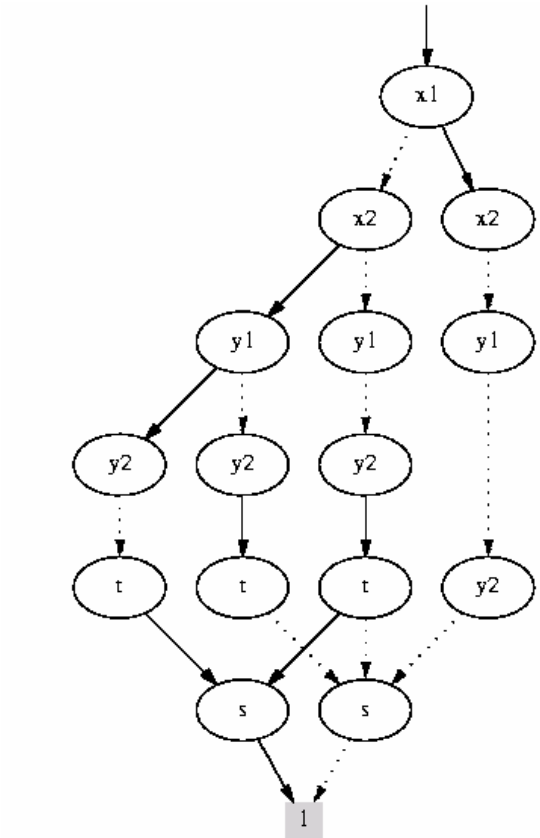
A MUTEX Algorithm

Clarke & Emerson

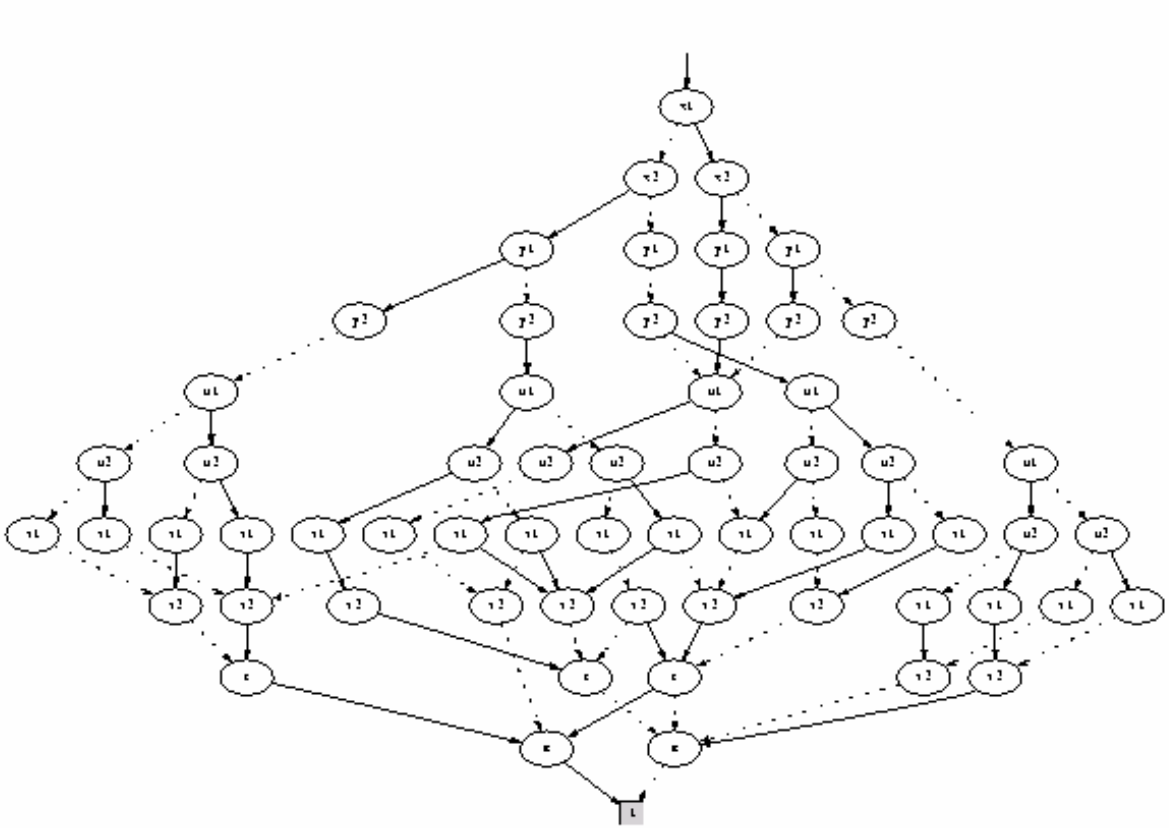
```
vars x1 x2;
vars y1 y2;
vars u1 u2;
vars v1 v2;
vars t s;

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



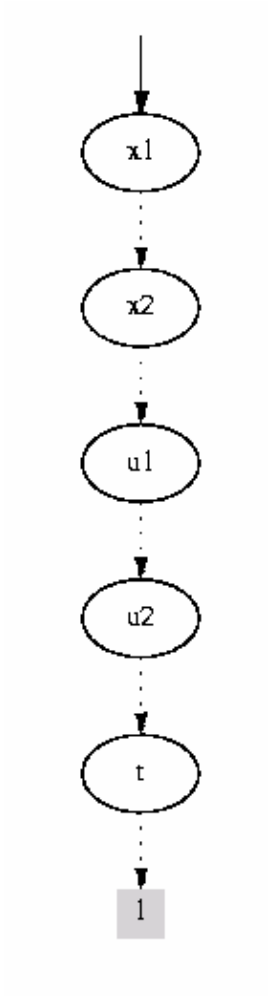
ATrans

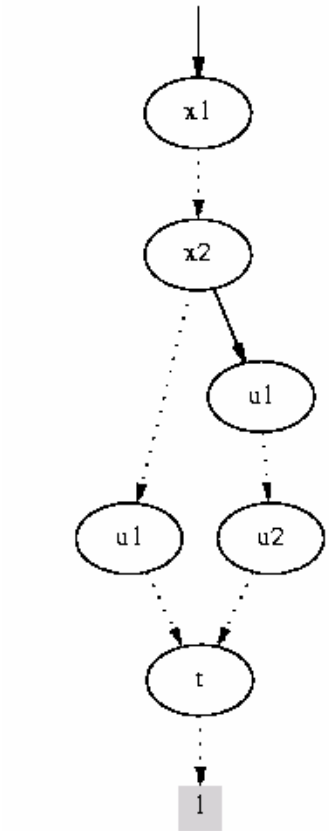


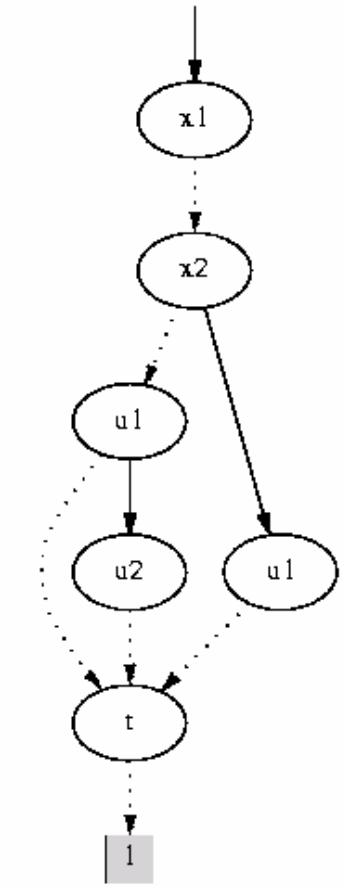
TT

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)
```

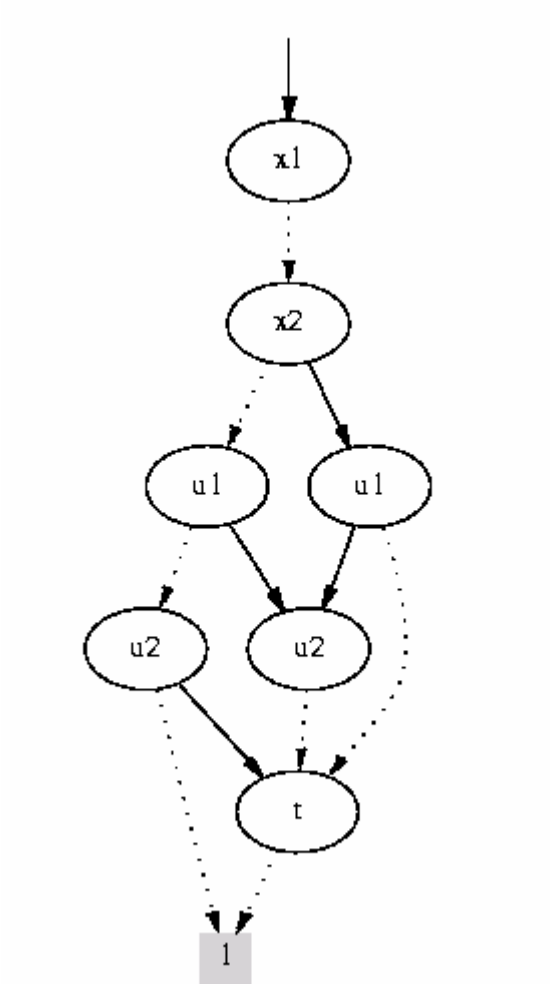


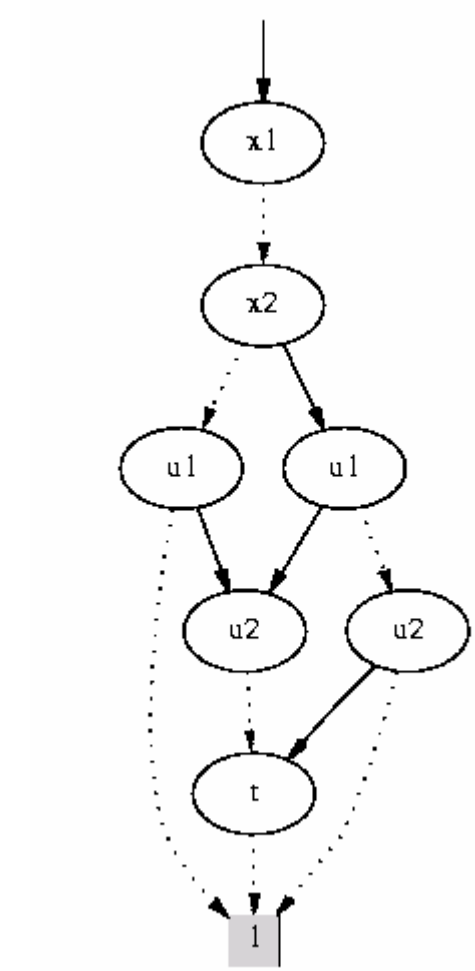


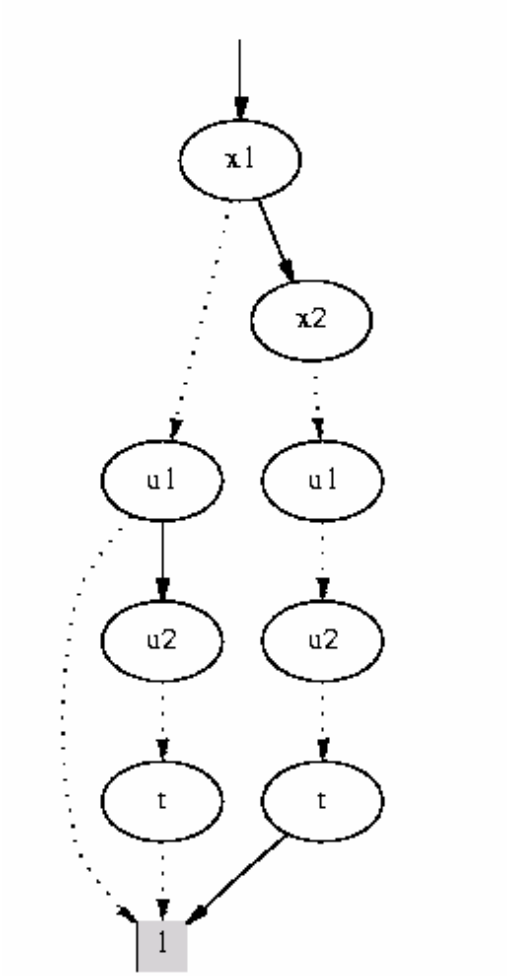


Reachable States

```
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)
```

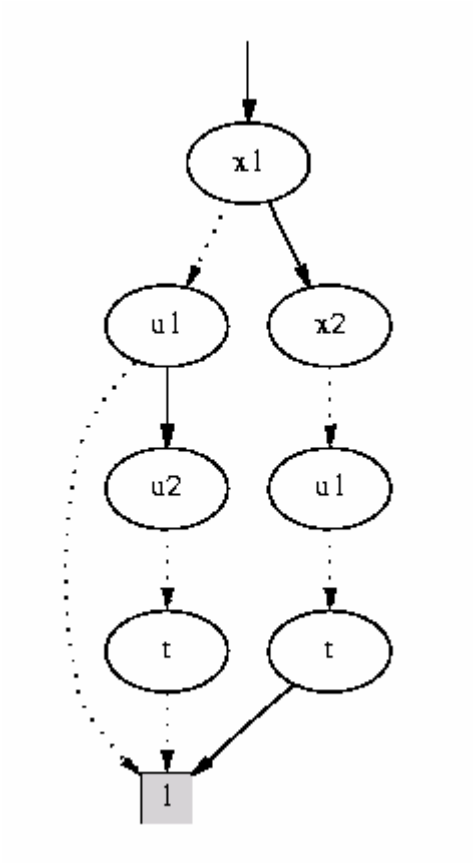


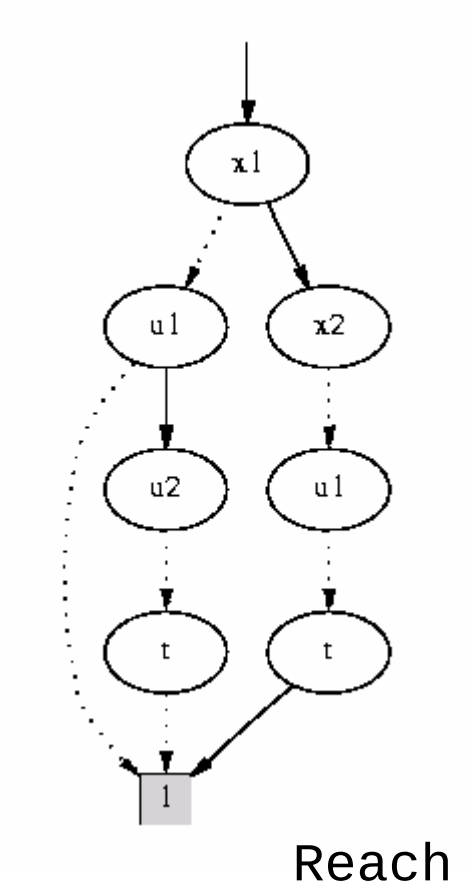




Reachable States

```
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

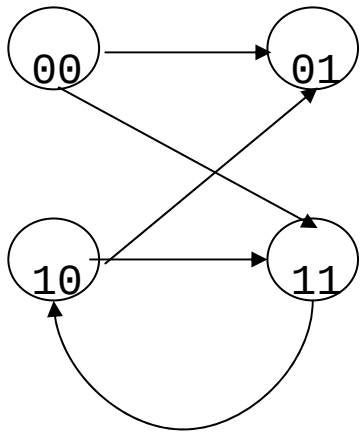
MUTEX ?

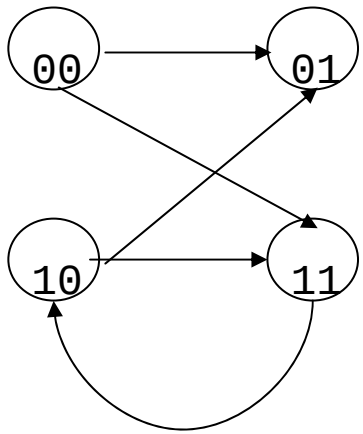
```
Reach &
    x1 & !x2 &
    u1 & !u2
```

1

Bisimulation

vars **x (y)**

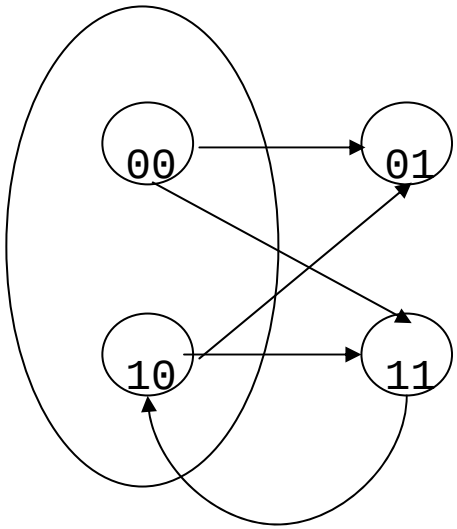




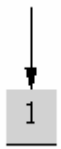
vars **u (v)**

```
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)
```

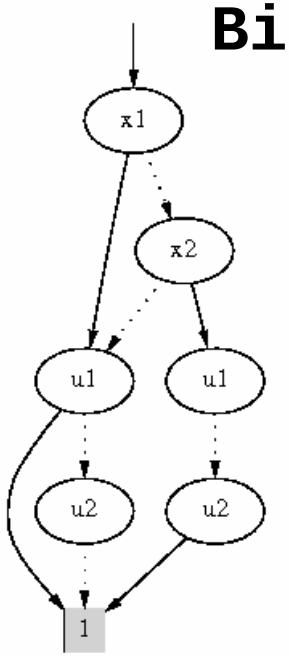
Bisimulation (cont.)



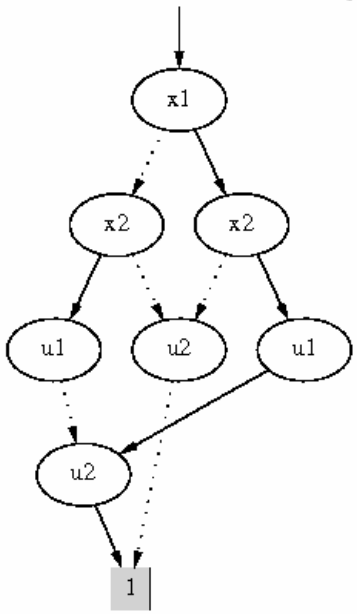
$\mathbf{Bis_0}$



$\mathbf{Bis_1}$

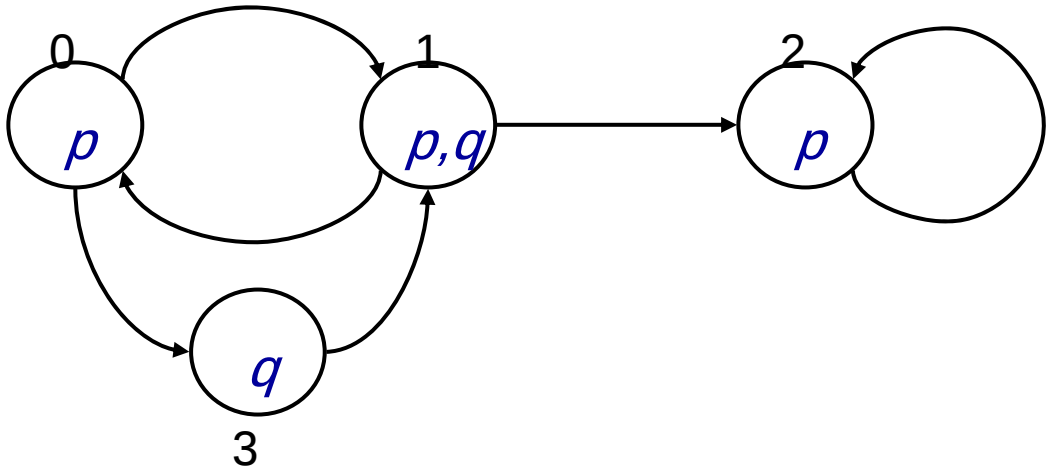


$\mathbf{Bis_2}$



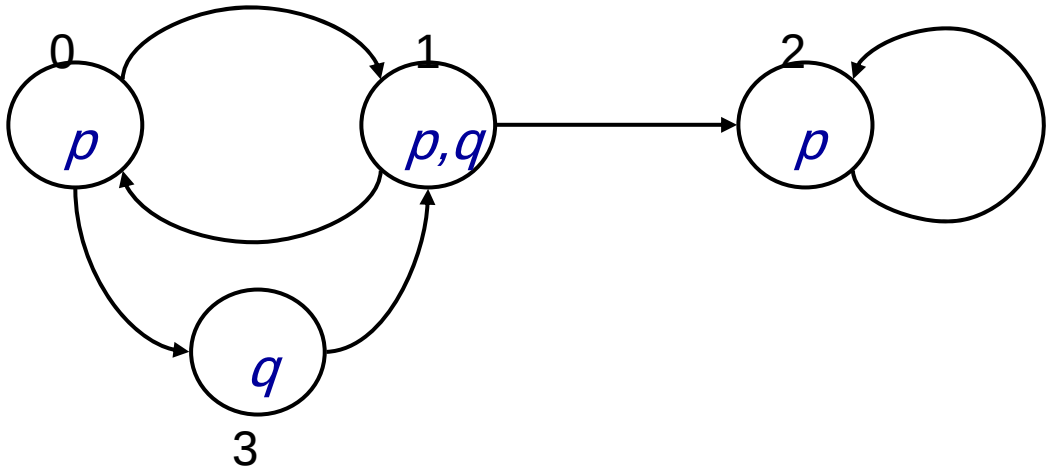
3 equivalence classes
= 6 pairs in final bisimulation

Model Checking




```
vars x1 x2;
vars y1 y2;

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

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

Q(x1,x2) := …… ;
```

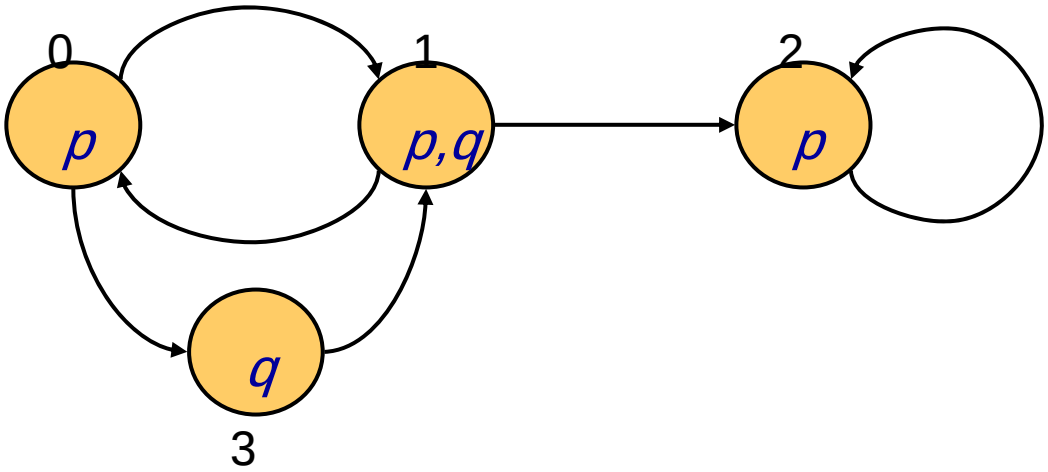
Model Checking



EX P

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

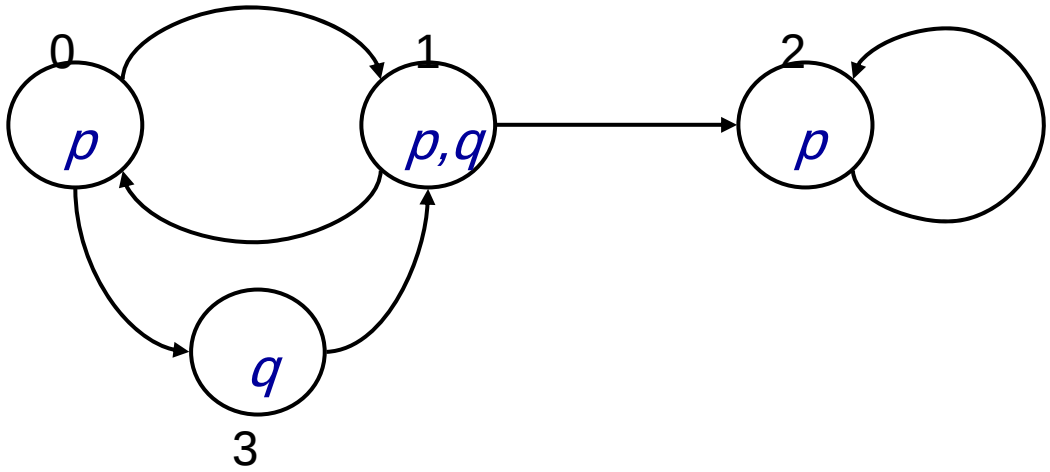
Model Checking



EX P

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

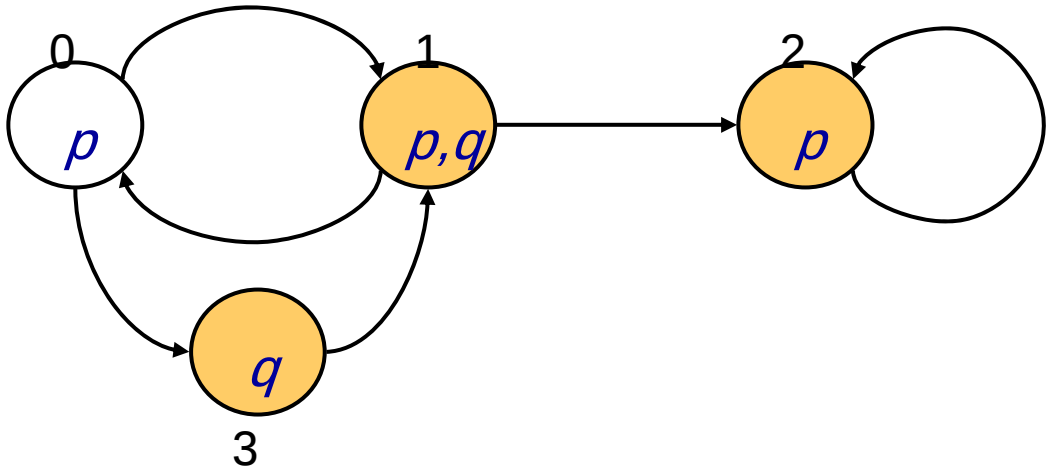
Model Checking



AX P

```
Forall y1,y2.
   Trans(x1,x2,y1,y2) =>
   P(y1,y2);
```

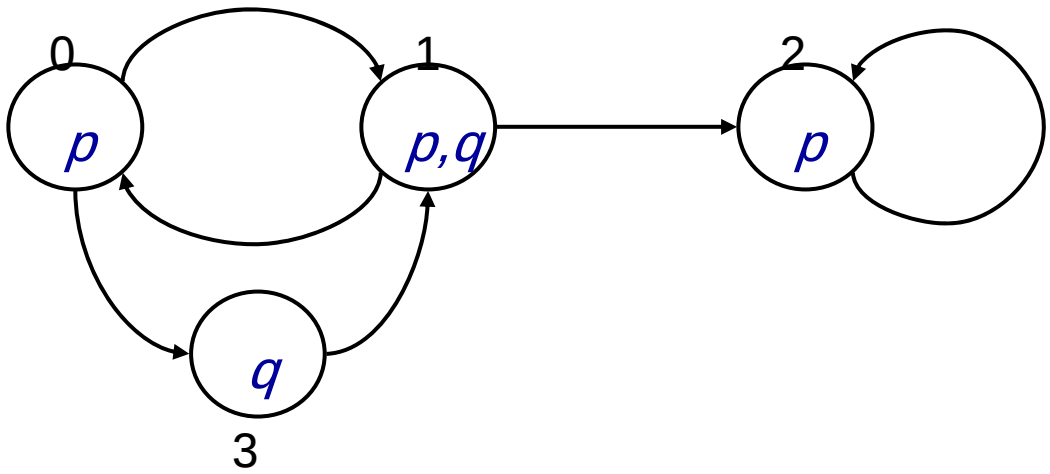
Model Checking



AX P

```
Forall y1,y2.
    Trans(x1,x2,y1,y2) =>
    P(y1,y2);
```

Model Checking

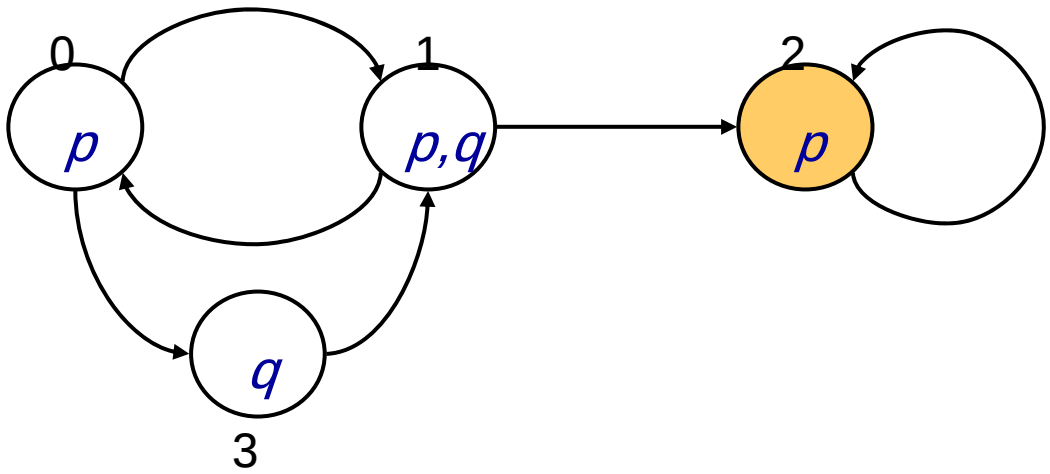


AG P

max fixpoint

```
A(x1,x2) =
  P(x1,x2) &
  Forall y1,y2.
     Trans(x1,x2,y1,y2) =>
     A(y1,y2);
```

Model Checking

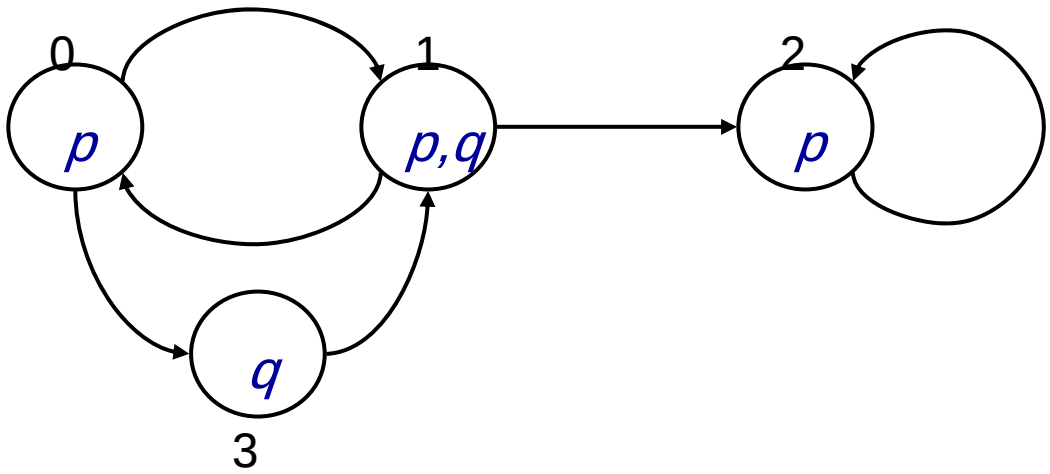


AG P

max fixpoint

```
A(x1,x2) =
  P(X1,x2) &
  Forall y1,y2.
     Trans(x1,x2,y1,y2) =>
     A(y1,y2);
```

Model Checking

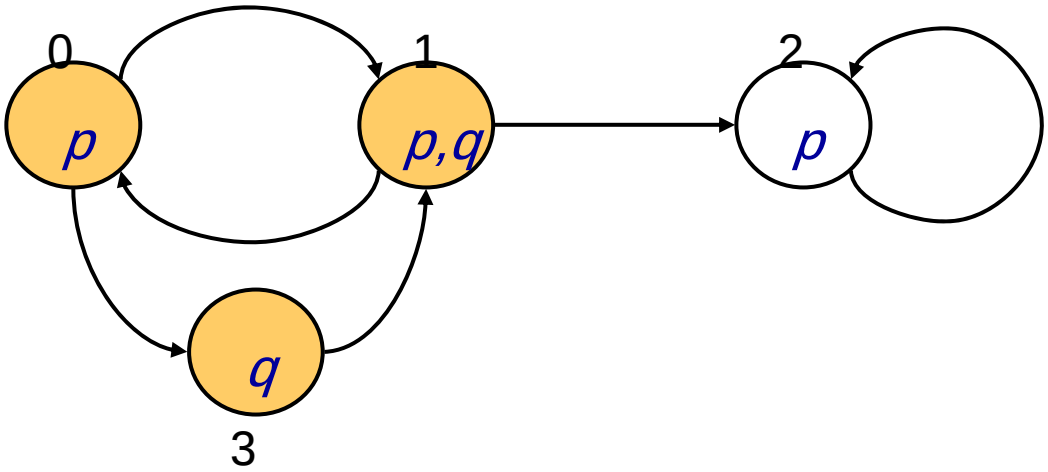


A(P UNTIL Q)

min fixpoint

```
U(x1,x2) =
  Q(x1,x2) +
  { P(x1,x2) &
     Forall y1,y2.
     Trans(x1,x2,y1,y2) =>
       U(y1,y2)
   };
```

Model Checking



A(P UNTIL Q)

min fixpoint

```
U(x1,x2) =
  Q(X1,x2) +
  { P(x1,x2) &
     Forall y1,y2.
     Trans(x1,x2,y1,y2) =>
       U(y1,y2)
   };
```

Partitioned Transition Relation

Relational Product

LARGE

```
Exists yv. (T(xu,yv) & S(yv))
```

Asynchronous

```
T(xu,yv) =
      (ATrans(x,y) & v=u)
    + (BTrans(u,v) & y=x)
```

```
Exists y. ATrans(x,y) & S(yu)
+
Exists v. BTrans(u,v) & S(xv)
```

Synchronous

```
T(xu,yv) =
     ATrans(x,y)
   & BTrans(u,v)
```

```
Exists yv.
  Atrans(x,y)
& Btrans(u,v) & S(yv)
```

```
Exists y.
  Atrans(x,y)
& (Exists v. Btrans(u,v) & S(yv))
```

visualSTATE

CIT project VVS (w DTU)

Beologic's Products: salesPLUS visualSTATE

1980-95: *Independent division of B&0*
1995- : *Independent company*
B&O, 2M Invest,
Danish Municipal Pension Ins. Fund
1998: *BAAN*
2000: *IAR Systems A/S*

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

Customers

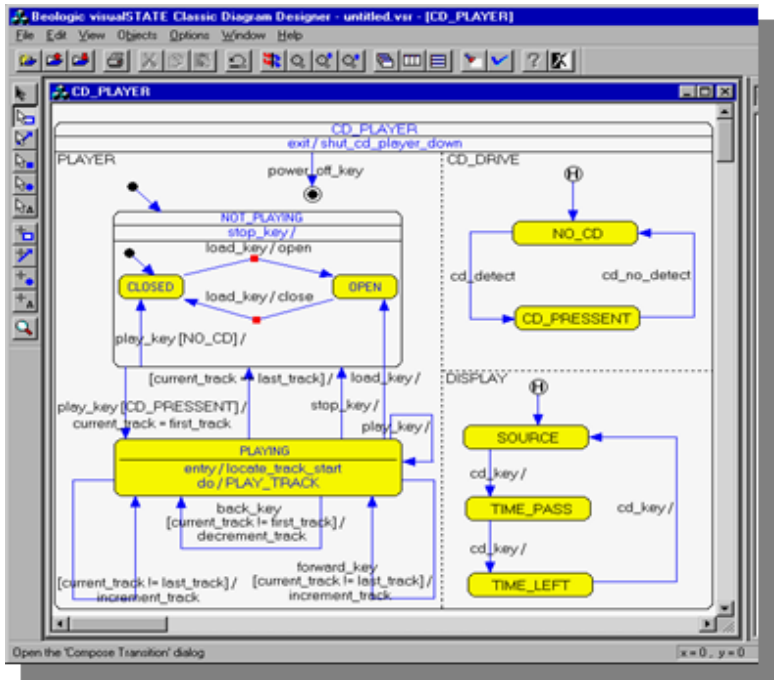
ABB
B&O
Daimler-Benz
Ericson DIAX
ESA/ESTEC
FORD
Grundfos
LEGO
PBS
Siemens ……. (approx. 200)

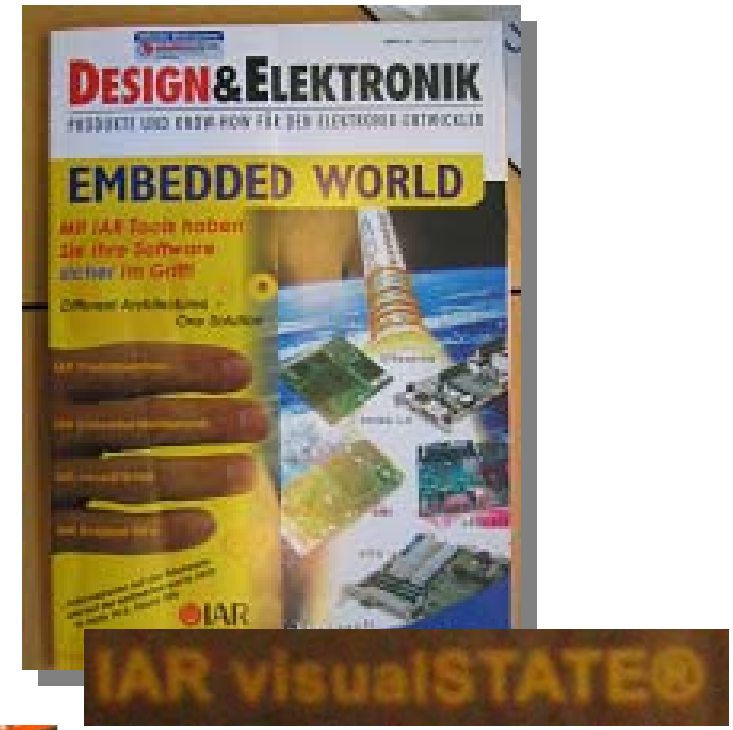
Verification Problems:

- 1.400 components
- 10^{400} states

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

visualSTATE



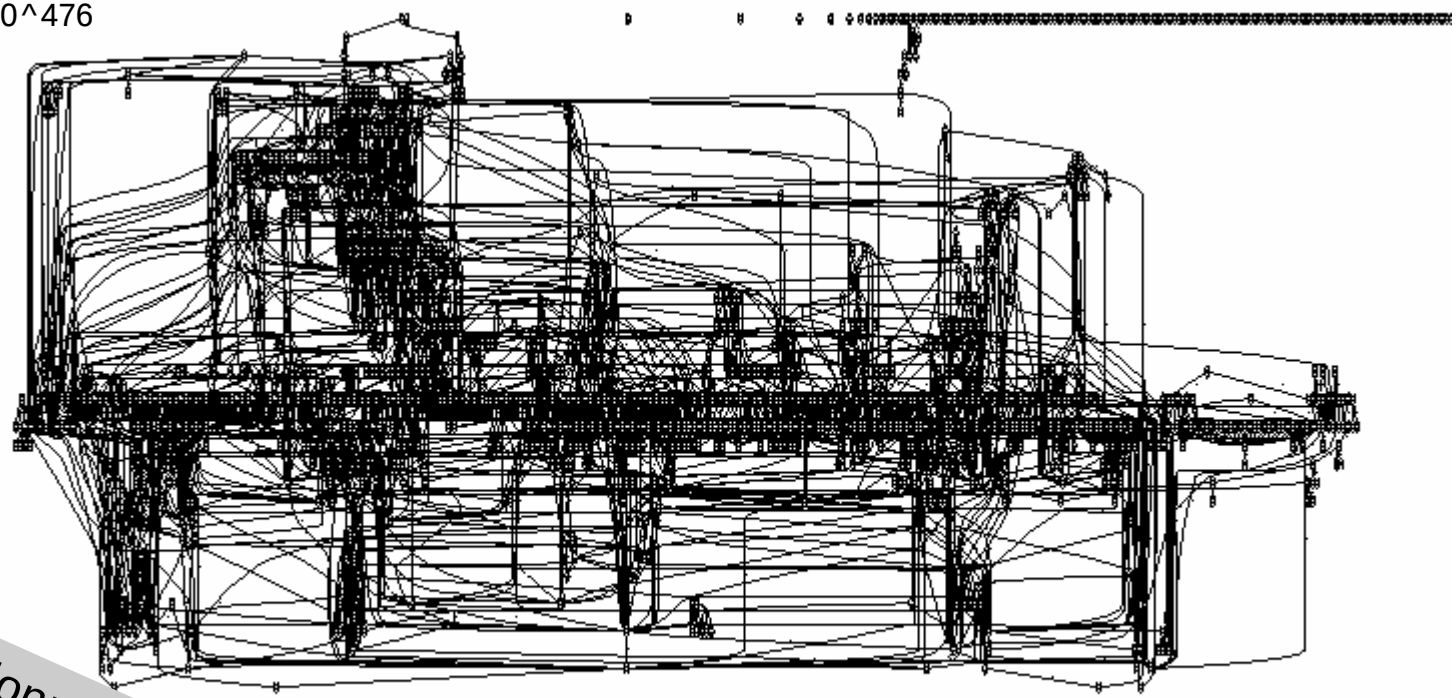


Embedded World
Nürnberg, 2005

Control Programs
A Train Simulator, visualSTATE (VVS)

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

BUGS ?



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

Experimental Breakthroughs

Patented

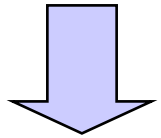
System	Mach.	State Space		Checks	Visual ST	St-of-Art		ComBack	
		Declared	Reach			Sec	MB	Sec	MB
VCR	7	10^5	1279	50	<1	<1	6	<1	7
JVC	8	10^4	352	22	<1	<1	6	<1	6
HI-FI	9	10^7	1416384	120	1200	1.0	6	3.9	6
Motor	12	10^7	34560	123	32	<1	6	2,0	
AVS	12	10^7	1438416	173	3780	6.7	6	5.7	6
Video	13	10^8	1219440	122	---	1.1	6	1.5	6
Car	20	10^11	9.2 10^9	83	---	3.8	9	1.8	6
N6	14	10^10	6399552	443	---	32.3	7	218	6
N5	25	10^12	5.0 10^10	269	---	56.2	7	9.1	6
N4	23	10^13	3.7 10^8	132	---	622	7	6.3	6
Train1	**373**	**10^136**	**---**	**1335**	**---**	**---**	**---**	**25.9**	**6**
Train2	**1421**	**10^476**	**---**	**4708**	**---**	**---**	**---**	**739**	**11**

Machine: 166 MHz Pentium PC with 32 MB RAM

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

Experimental Breakthroughs

Patented



System	Mach.	State Space		Checks	Visual ST	St		ComBack	
		Declared	Reach					ec	MB
VCR	7	10^5	1279	50				1	7
JVC	8	10^4	352						6
HI-FI	9	10^7	1416384					3.9	6
Motor	12	10^7					6	2,0	
AVS	12	10^7					6	5.7	6
Video	13	10^				1.1	6	1.5	6
Car	20				---	3.8	9	1.8	6
N6	1		143		---	32.3	7	218	6
N5			269		---	56.2	7	9.1	6
N4			10^8	132	---	622	7	6.3	6
Train1			---	**1335**	---	---	---	**25.9**	**6**
Train2			---	**4708**	---	---	---	**739**	**11**

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

Machine: 166 MHz P PC with 32 MB RAM

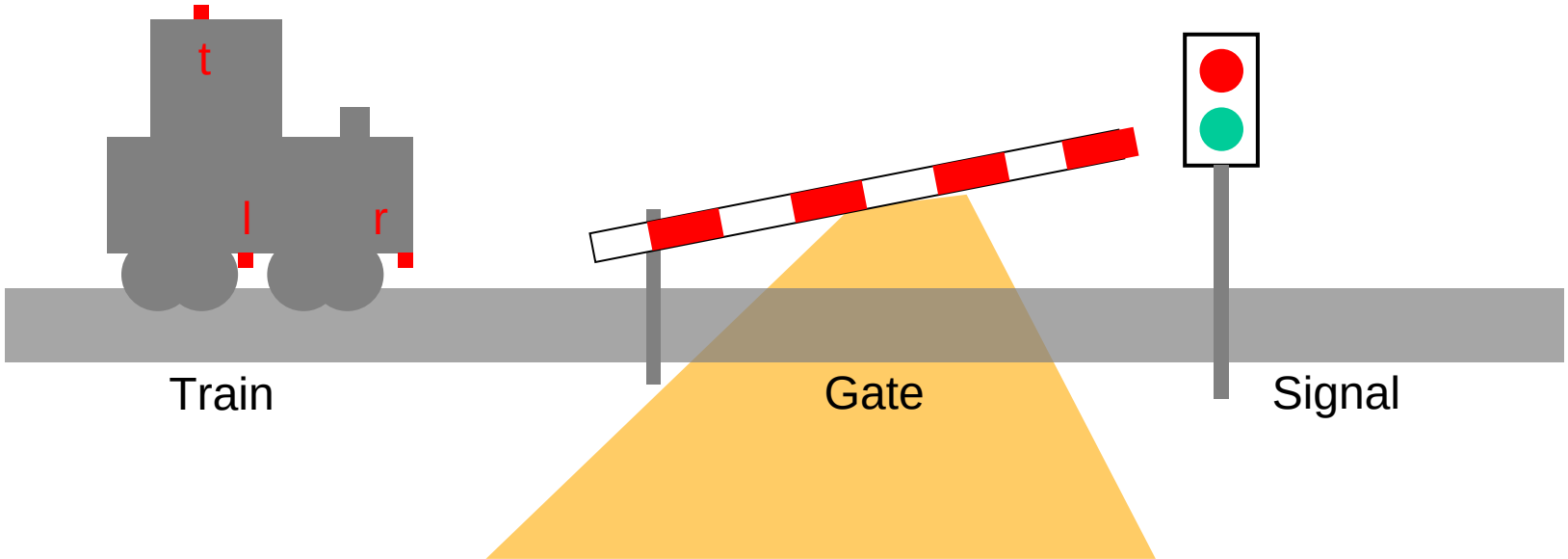
---: Out of memory, or did not terminate after 3 hours.

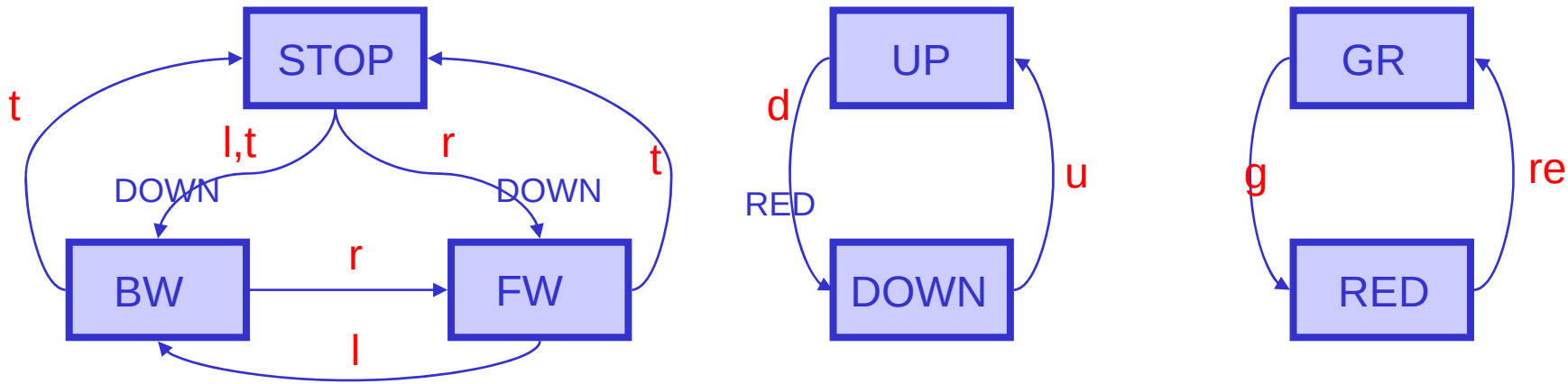
Compositional Backwards Reachability

[TACAS'98]

Example

The Small Train





State-Event Model

visualSTATE

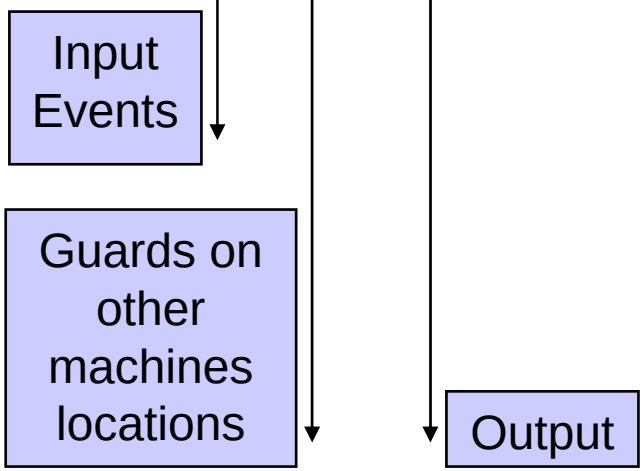
Syntax

n synchronously combined machines

$$M_i = (S_i, s_i^0, T_i)$$

where

$$T_i \subseteq S_i \times E \times G_i \times M(O) \times S_i$$



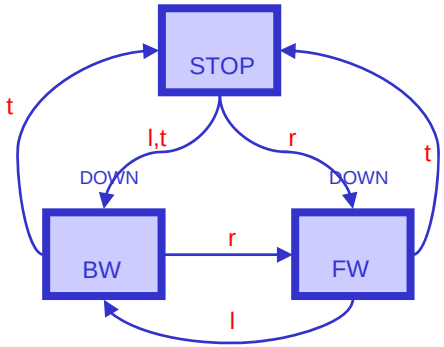
Semantics

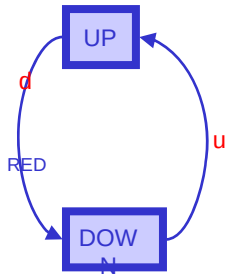
$$(s_1,\ldots,s_n) \xrightarrow{-e,\cup o_i} (s_1',\ldots,s_n')$$

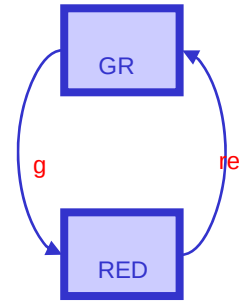
iff

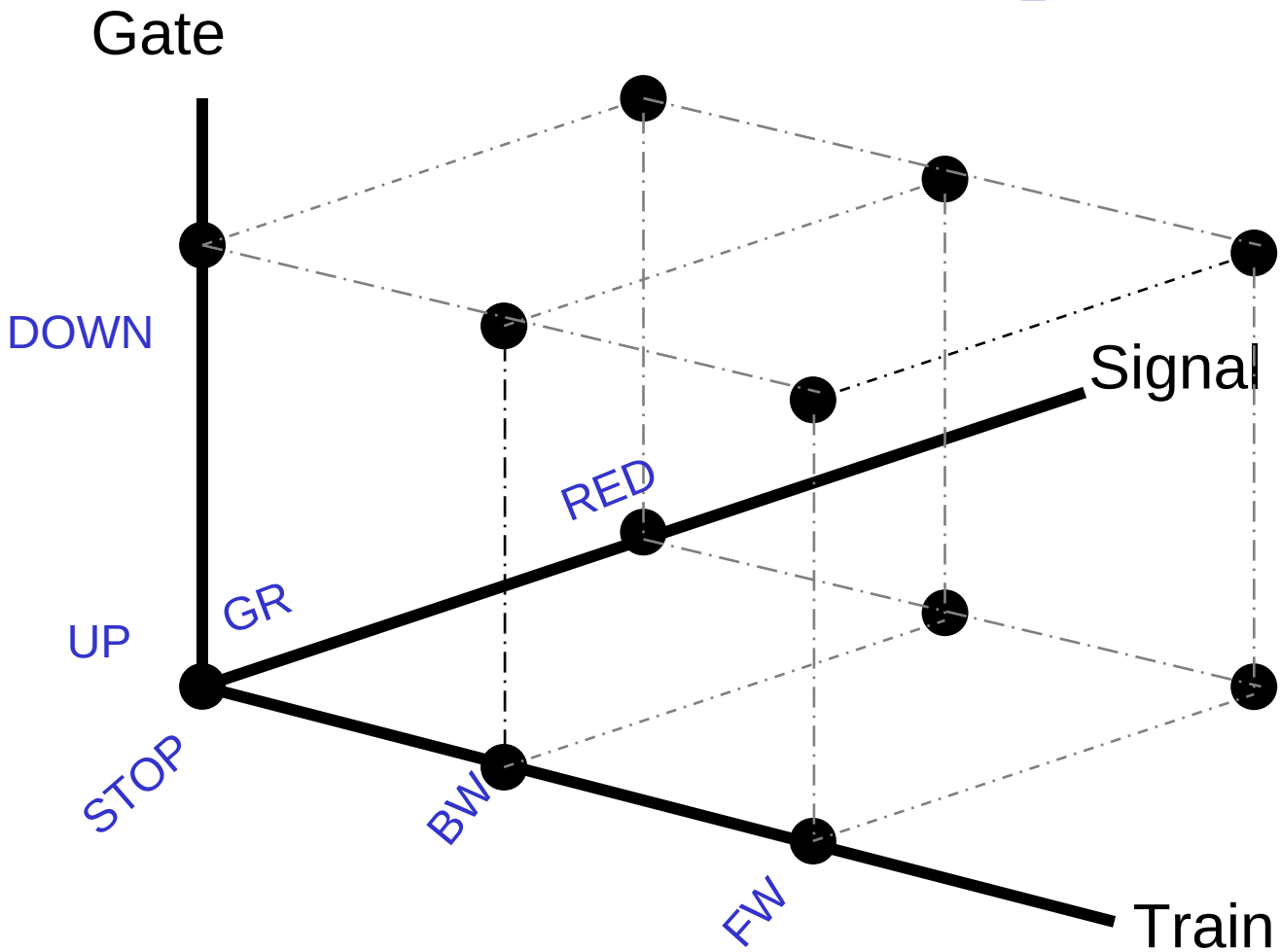
$$\left\{ \begin{array}{l} s_i \xrightarrow{-e,g_i,o_i} s_i' \text{ with } g_i(s_1,\ldots,s_n)=\text{true} \\ \quad \text{or} \\ s_i = s_i' \text{ and } o_i=\emptyset \text{ and whenever } s_i \xrightarrow{-e,\,g_i} \text{ then } g_i(s_1,\ldots,s_n)=\text{false} \end{array} \right\}_{i=1..n}$$

Small Train (cont)

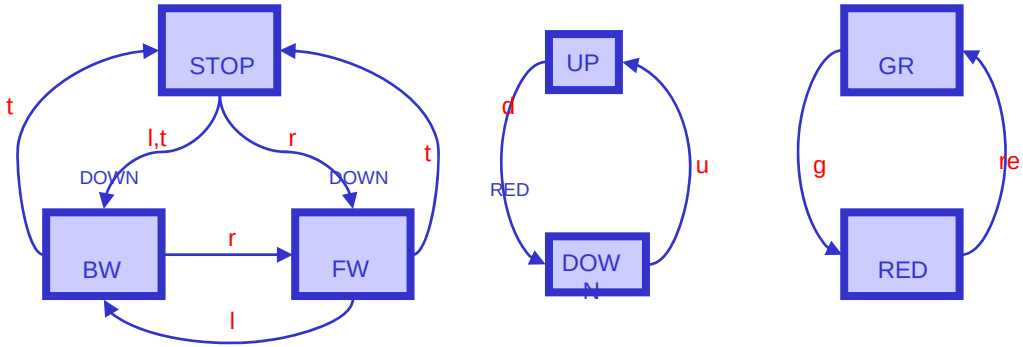


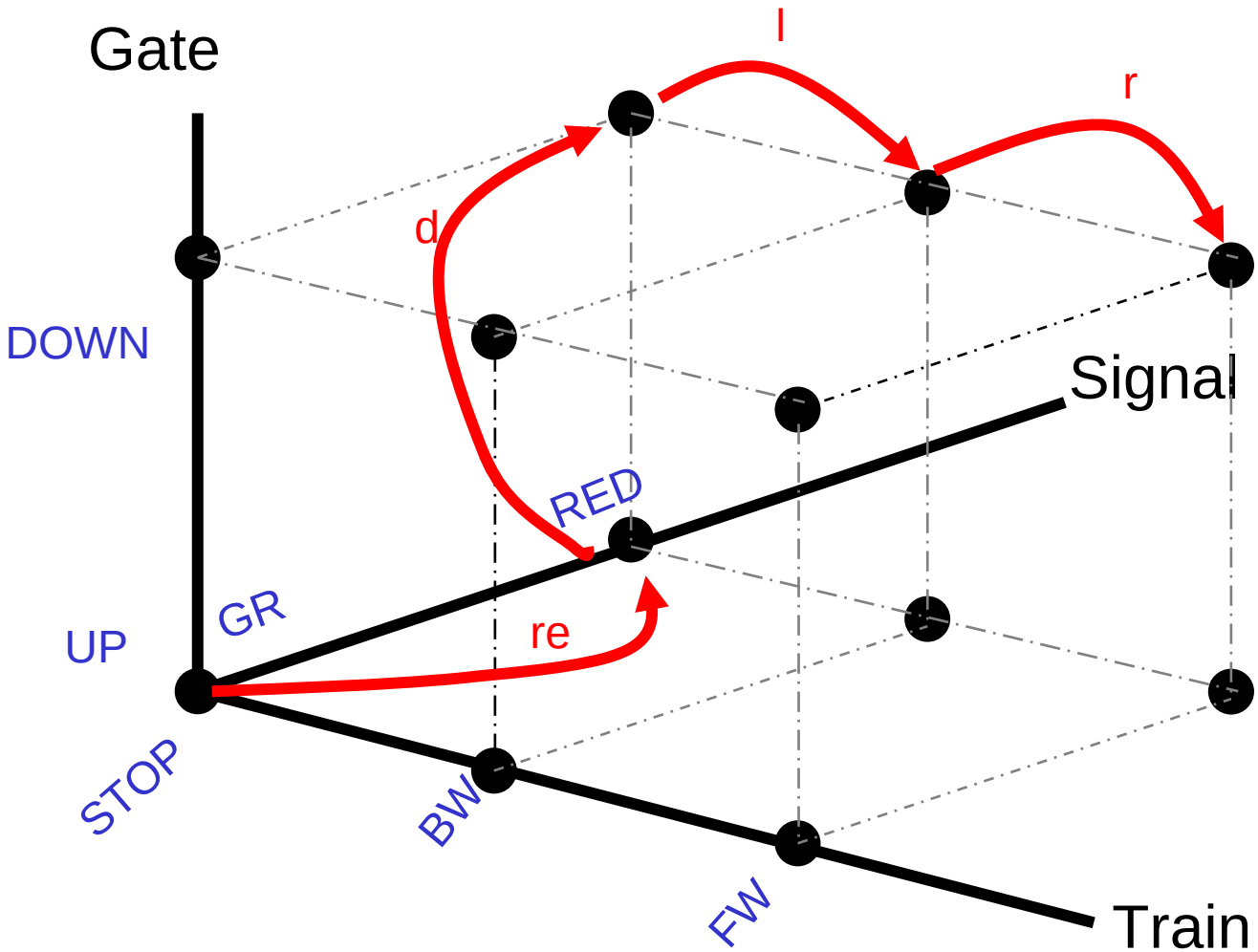






Small Train (cont)

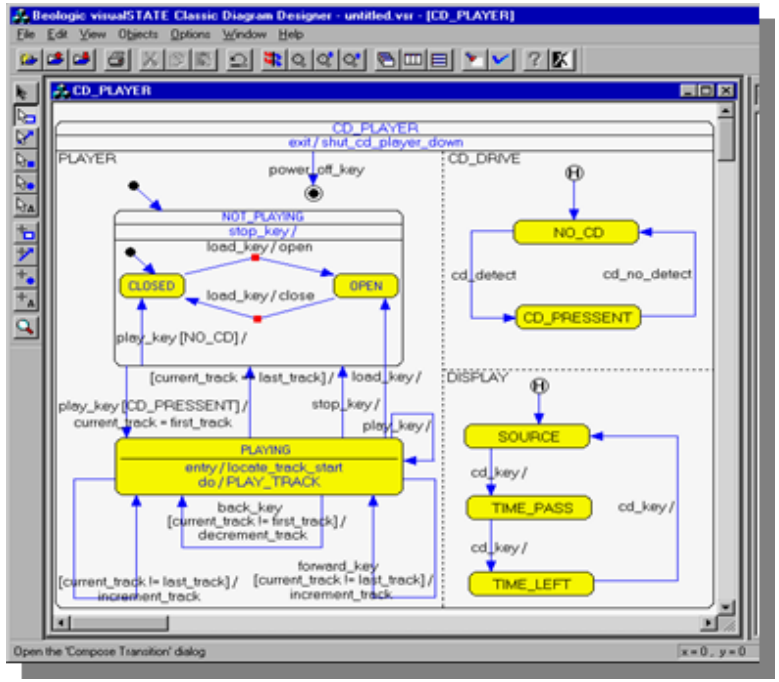




Generic Checks

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!

Reachability

Guard dependencies

Let

$g_1, g_2, g_3, \ldots, g_N$ (N big)

be the guards we want to show reachable

If

$g_i \Rightarrow g_j$ (e.g. g_i=DOWN $\wedge$ UP, g_j=DOWN)

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

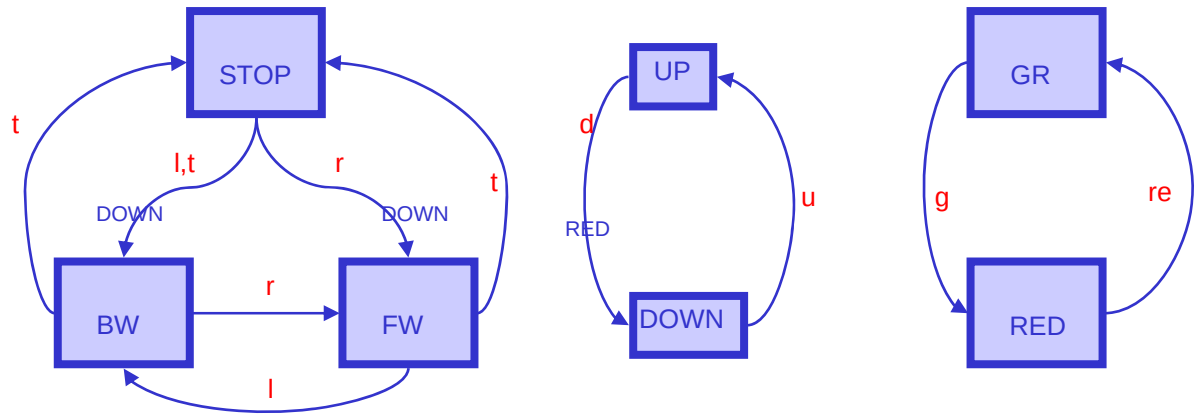
Sort $g_1, g_2, g_3, \ldots, g_N$ according to size and check only if NOT implied by a previously shown reachable guard

➔

40-70-90 % reduction.

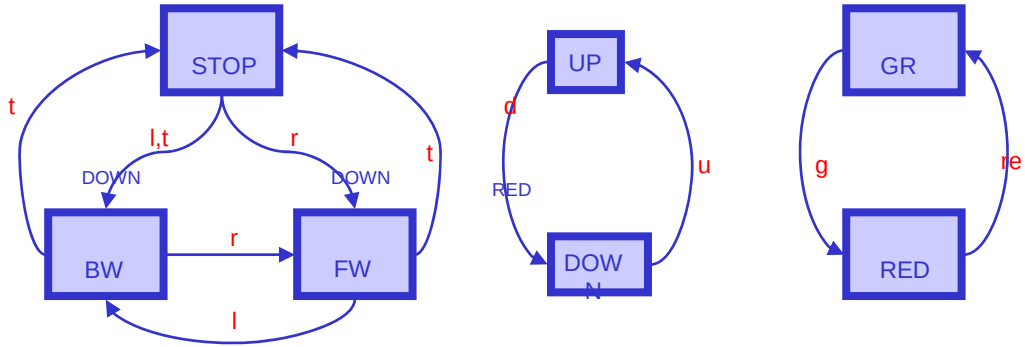
Machine Dependencies

A guard g in machie M_i that depends on/refers 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. DC(GATE) = {GATE,SIGNAL}

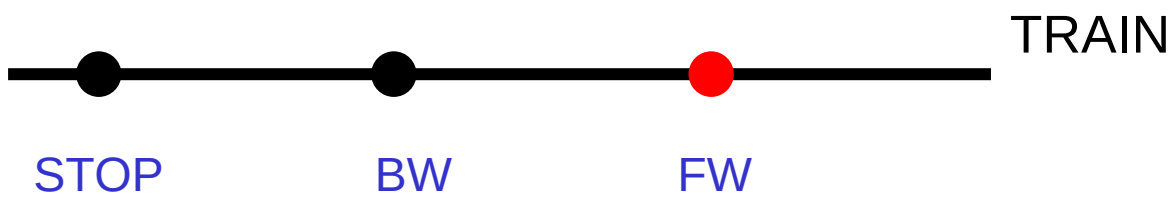
Compositional Backwards



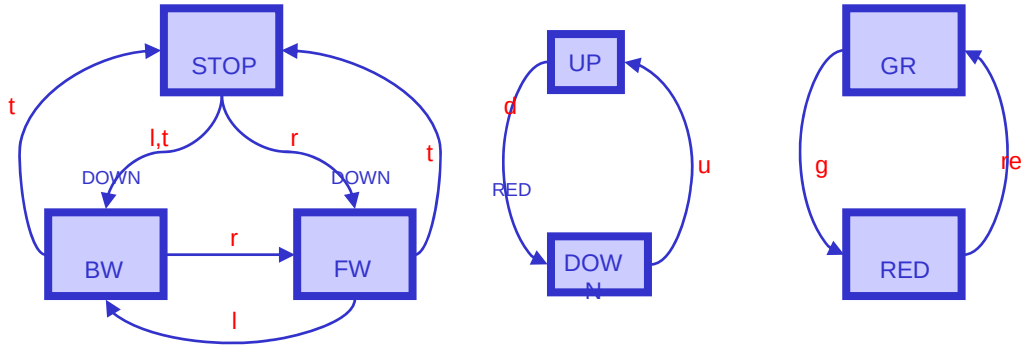
Is FW reachable ?

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

Consider TRAIN



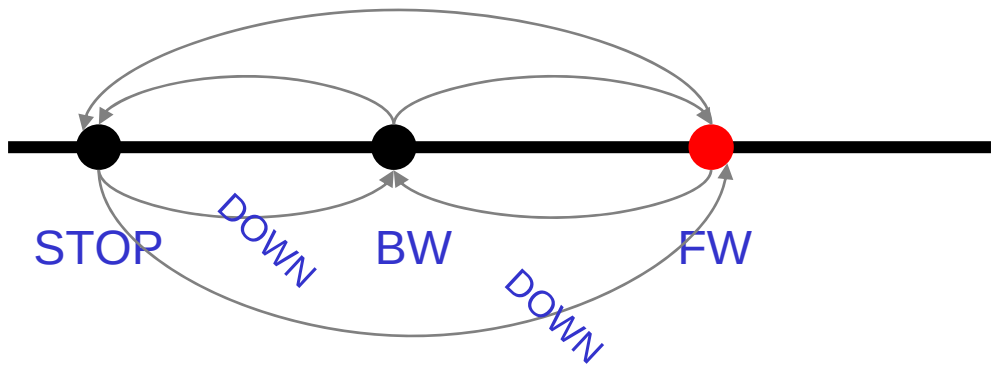
Compositional Backwards



Is FW reachable ?

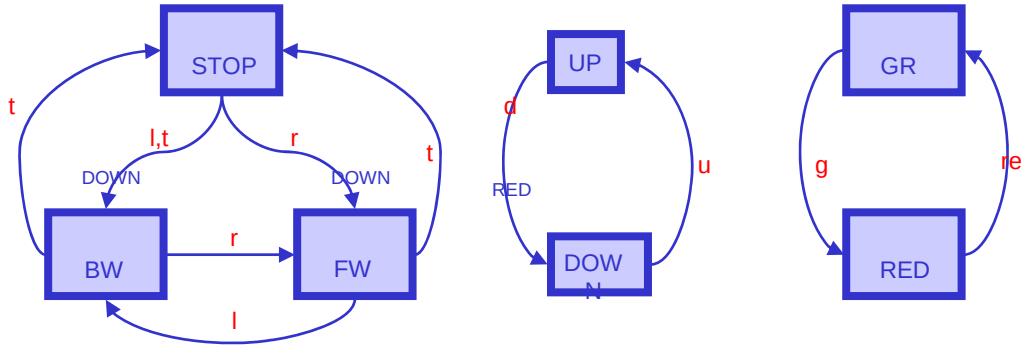
IDEA:
Compute backward reachable states as much as possible with minimal set of machines.
Increase set of considered machines when necessary!

Consider TRAIN



Ignoring INPUT event

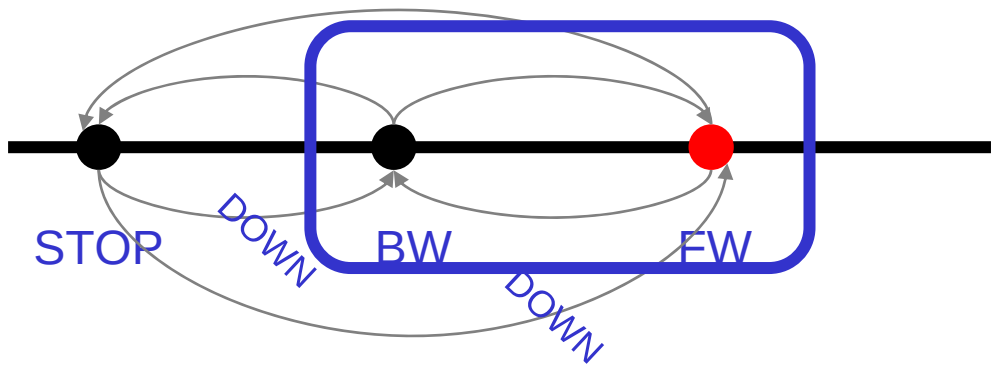
Compositional Backwards



Is FW reachable ?

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

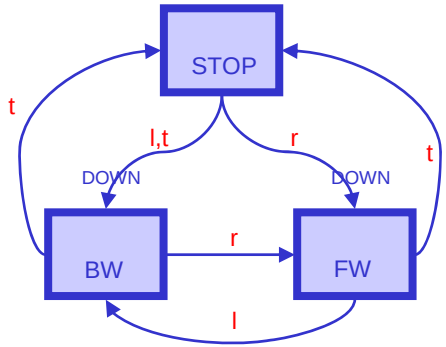
Consider TRAIN

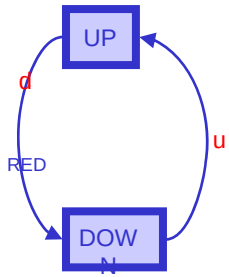


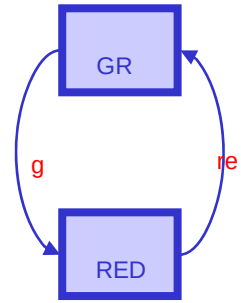
TRAIN

Ignoring INPUT event

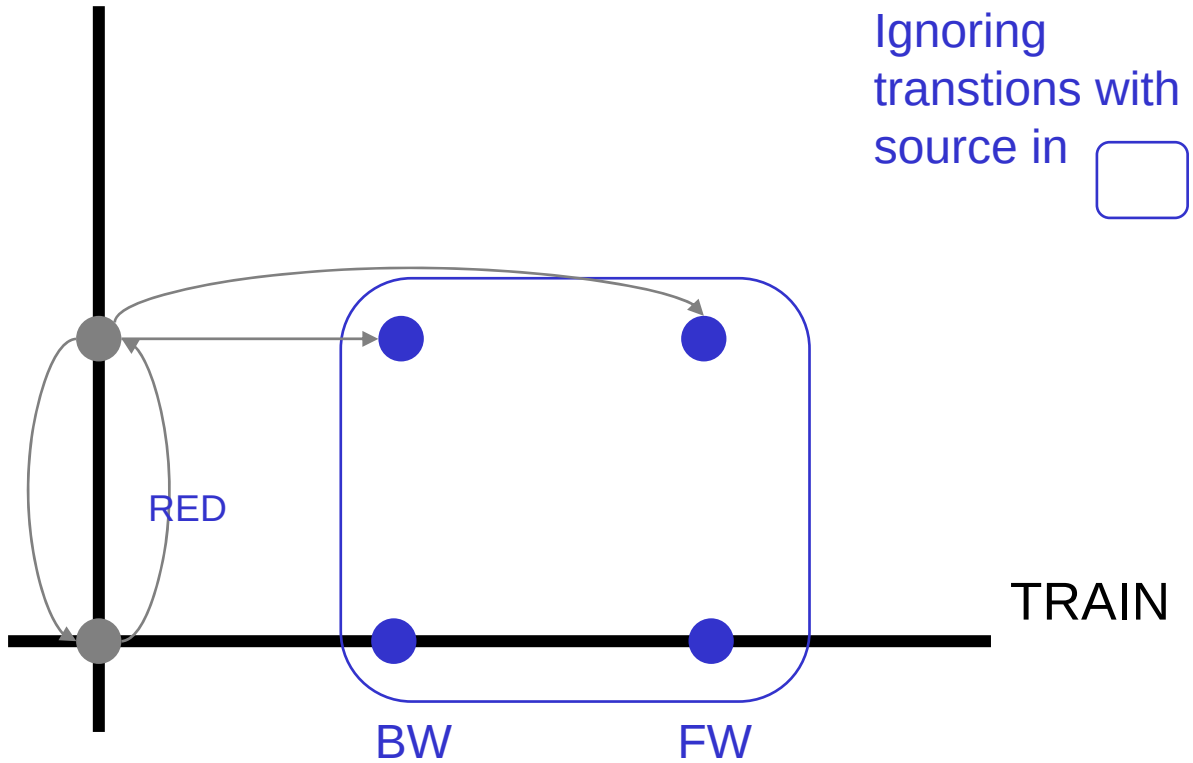
Compositional Backwards



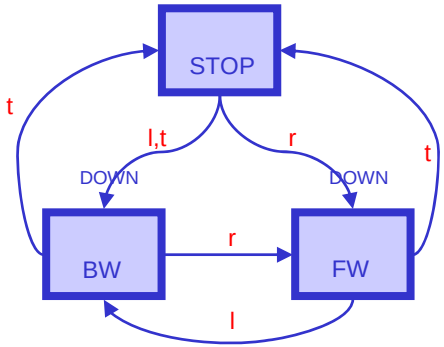


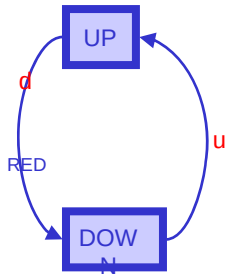


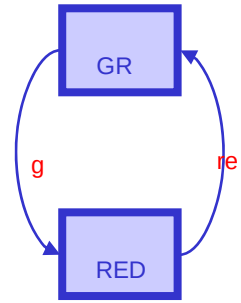
Include GATE !



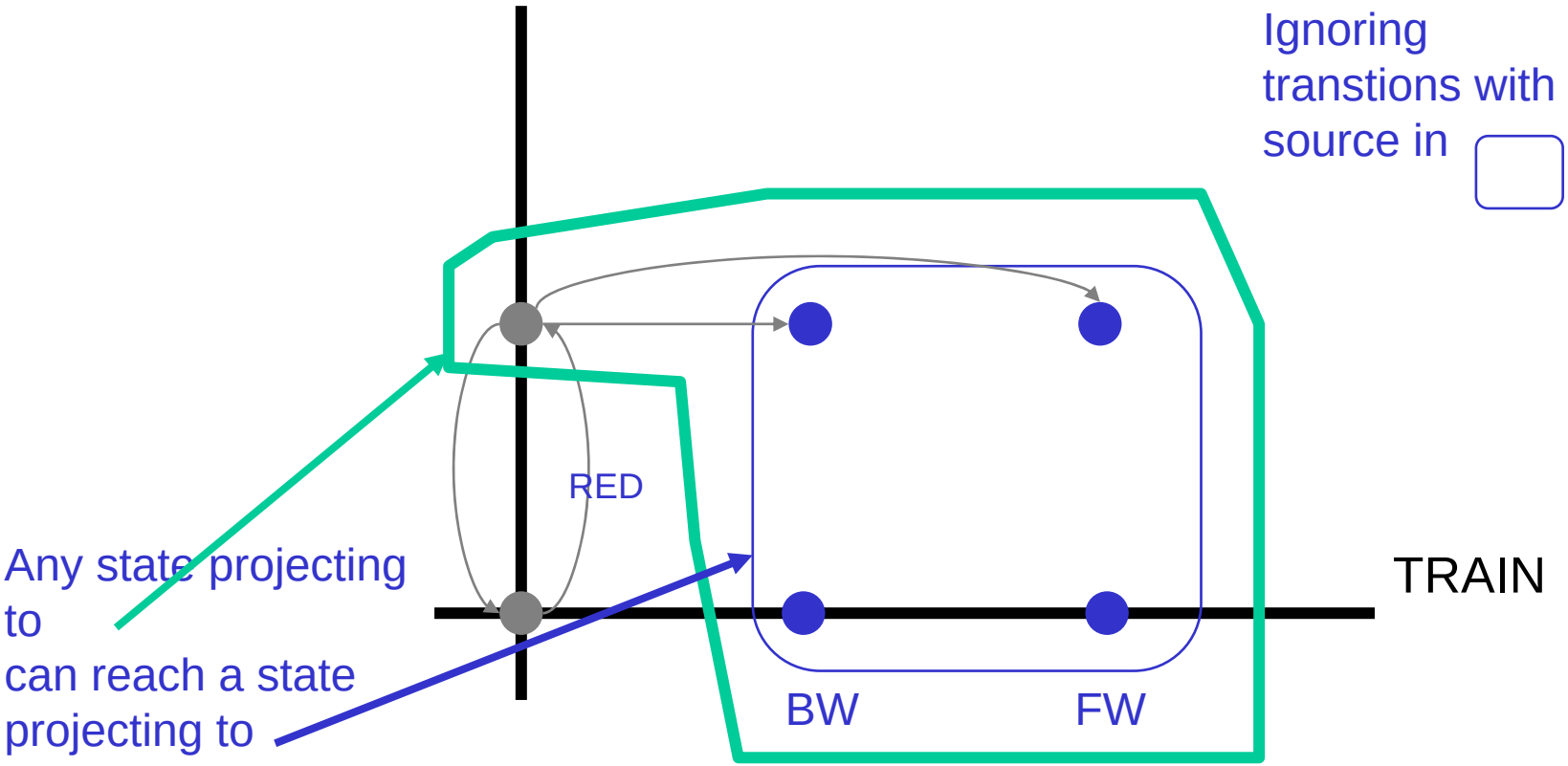
Compositional Backwards



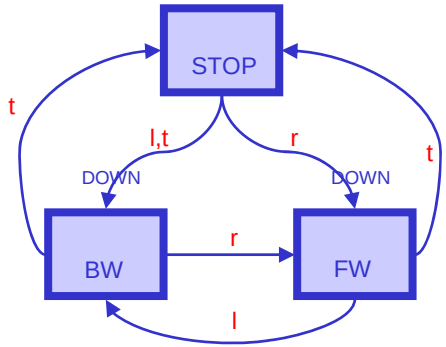


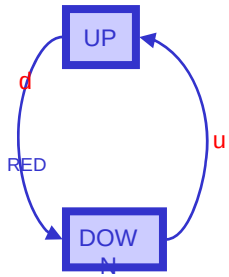


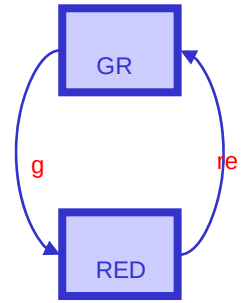
Include GATE !



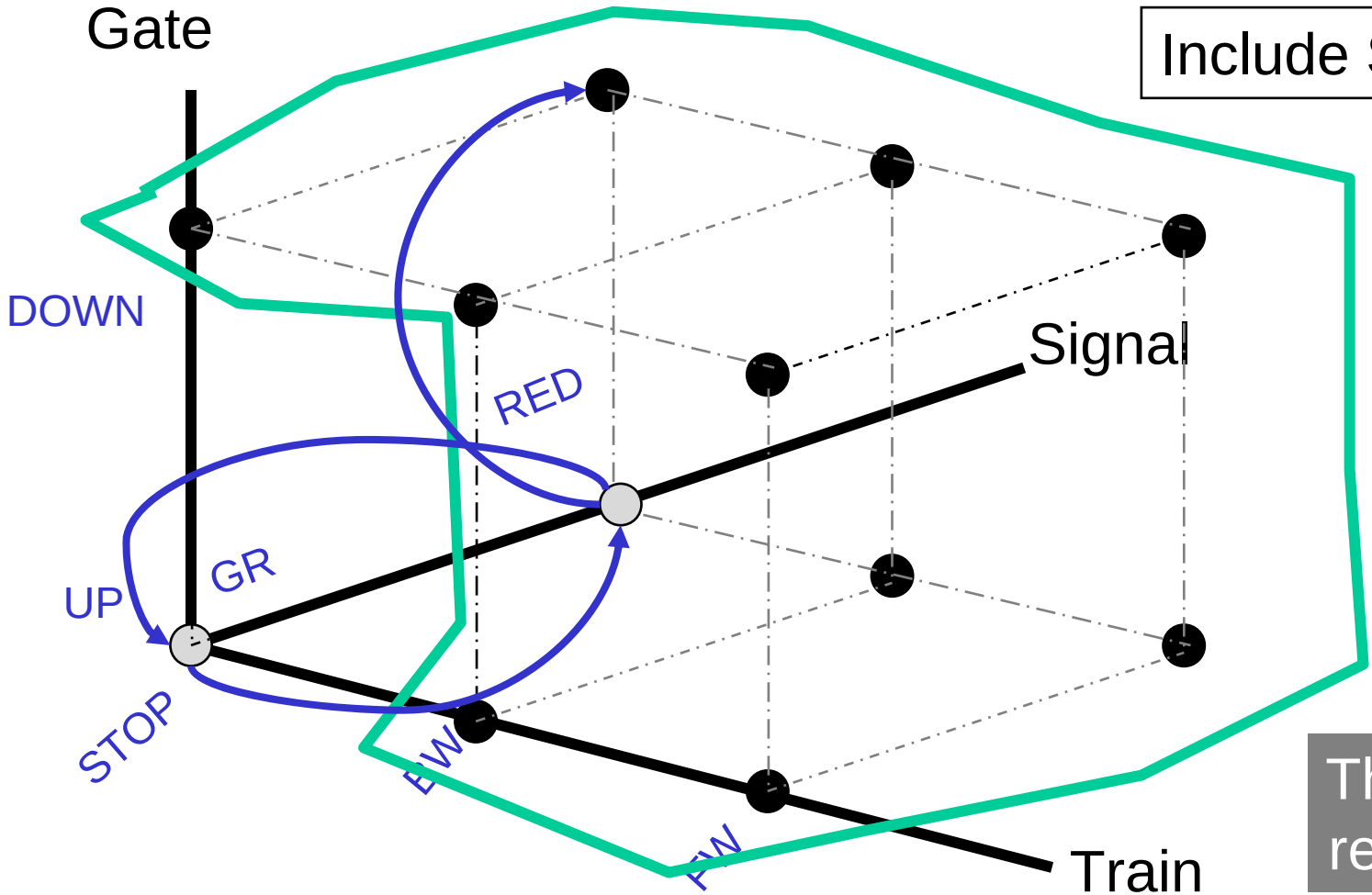
Compositional Backwards







Include SIGNAL !



Thus FW is reachable !

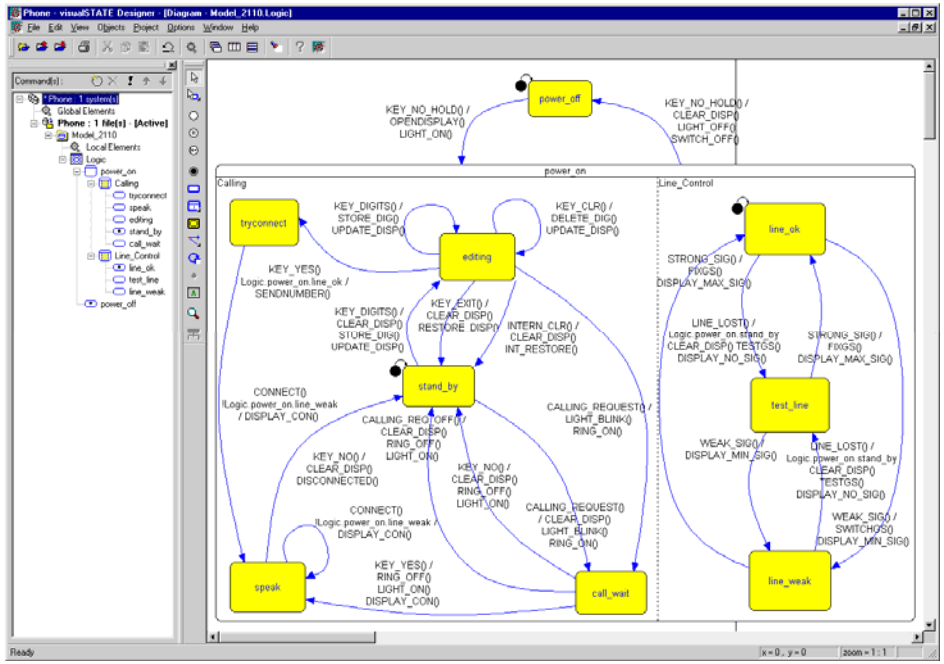
Hierarchical Systems

[TACAS'99]

IDEA

Reuse already known

- reachability properties of superstates to conclude reachability of substates !



Experimental results