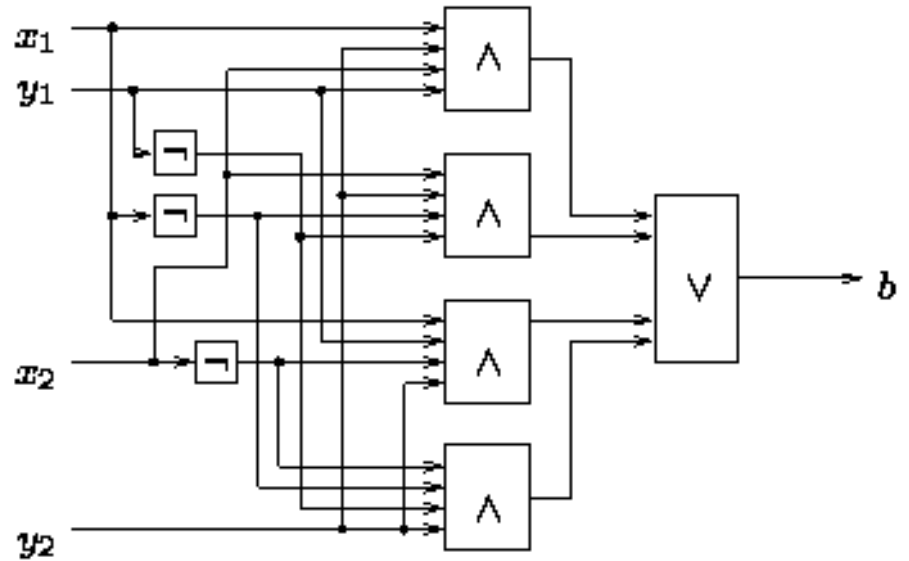
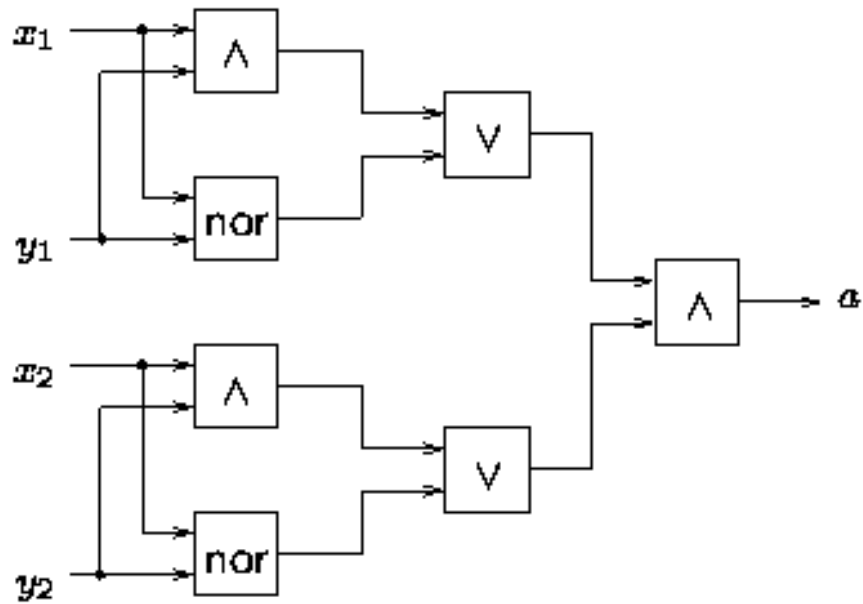


Applications of BDDs

Kim Guldstrand Larsen

- ❖ *BDDs – short review*
- ❖ *Constraint Solving*
- ❖ *Verification*
- ❖ *..*
- ❖ ***SAT-solving** versus **BDDs***

Combinatorial Circuits



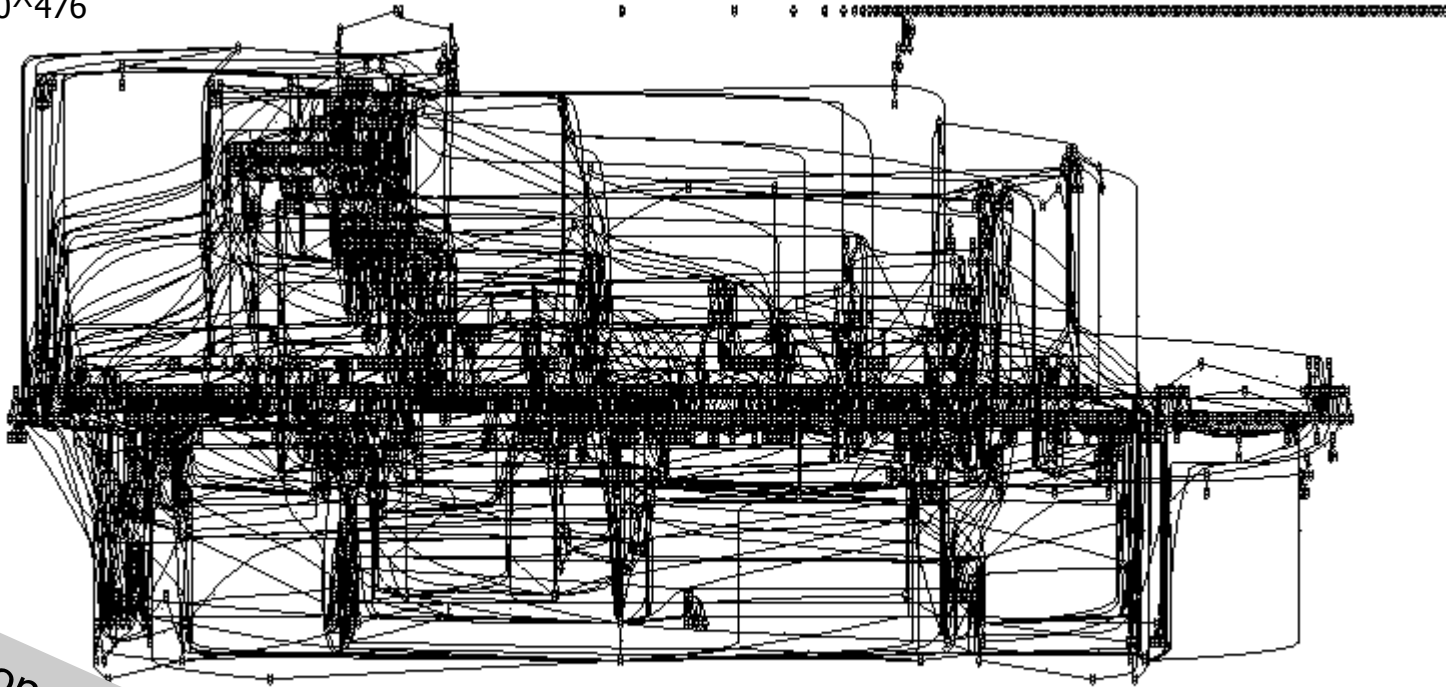
Are they two circuits equivalent?

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!

Binary Decision Diagrams

[Randal Bryant'86]

A short review

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 $var(v)$, and $low(v), high(v) \in V$.

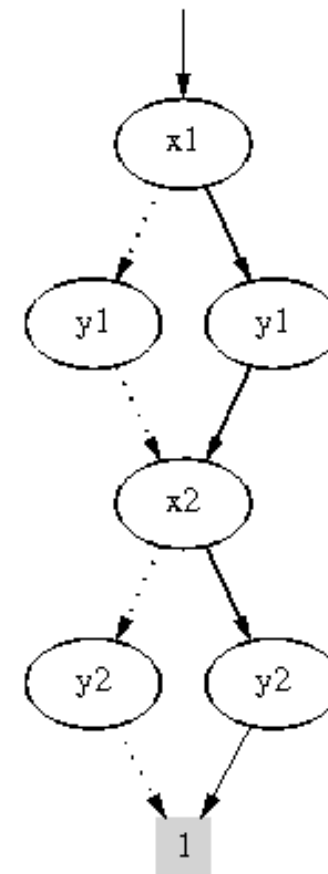
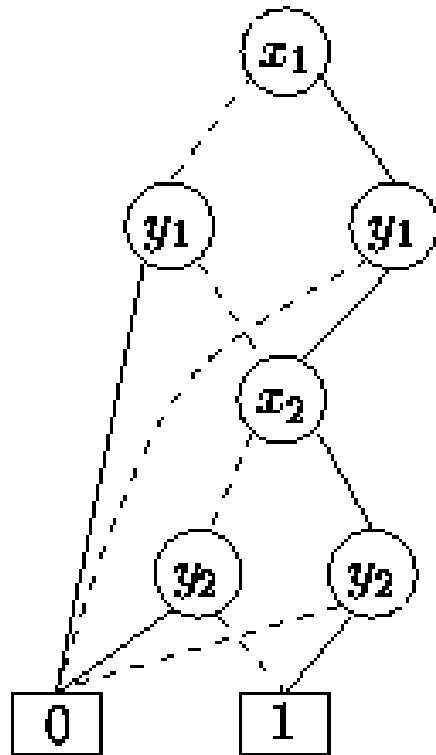
A BDD is *ordered* if on all paths from the root the variables respect a given total order.

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

- 1) $low(u) \neq high(u)$
- 2) $low(u) = low(v), high(u) = high(v), var(u) = var(v)$
implies $u = v$

Reduced Ordered Binary Decision Diagrams

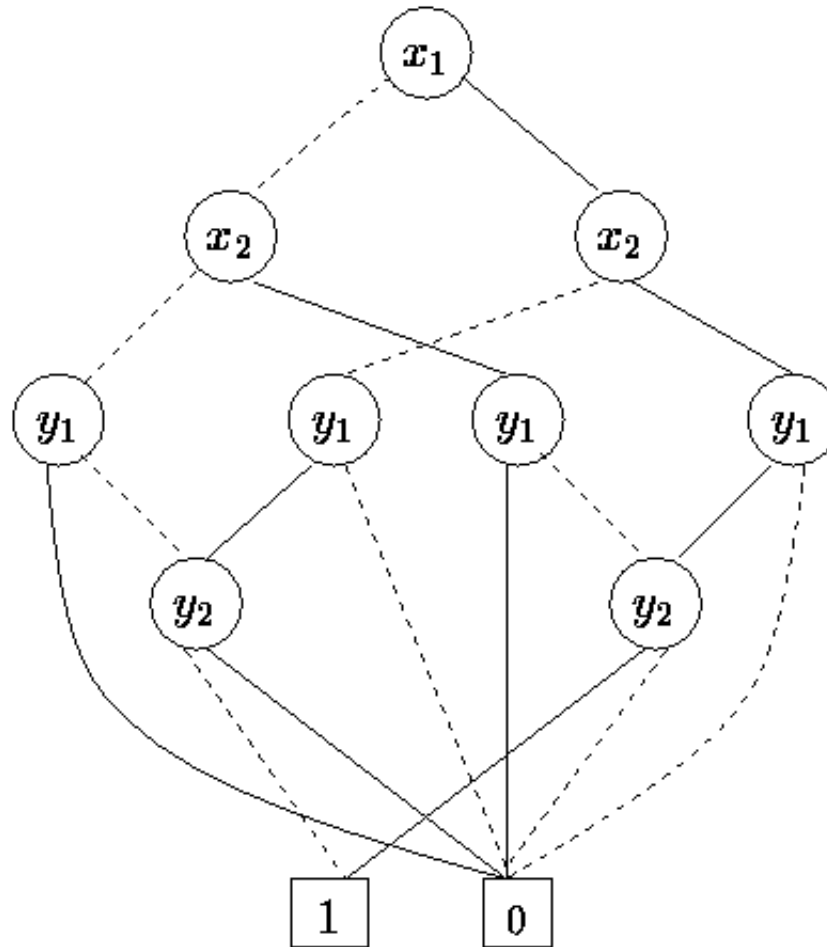
$$(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$$



lben
Edges to 0
implicit

Ordering *does* matter!

Variable ordering
 $x_1 < x_2 < y_1 < y_2$



Canonicity of ROBDDs

$$t_0 = 0$$

$$t_1 = 1$$

$$t_u = x \rightarrow t_h, t_l, \text{ if } u \text{ is a node } (x, l, h)$$

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

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

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

Consequences: b is a tautology, if and only if, $b = \boxed{1}$
 b is satisfiable, if and only if, $b \neq \boxed{0}$

BUILD

Build(t)

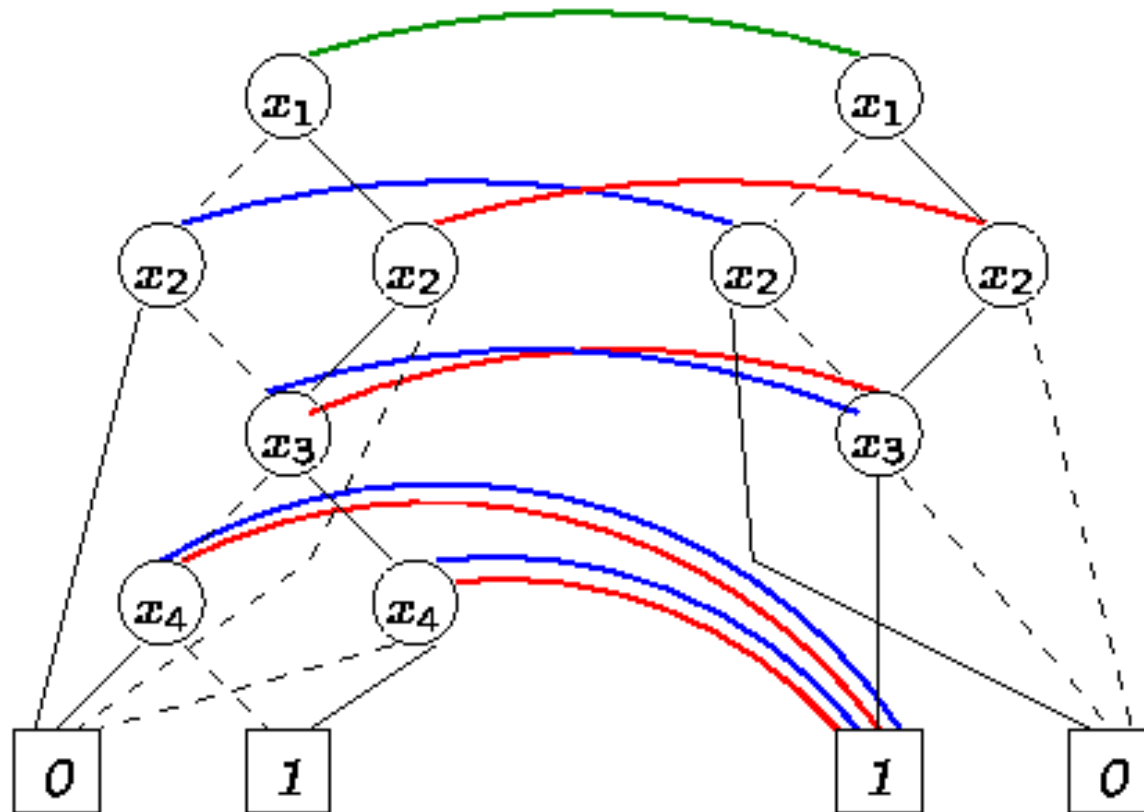
```
3:  function build'( $t, i$ ) =  
4:      if  $i > n$  then if  $t$  is false then return 0 else return 1  
5:      else  $l \leftarrow \text{build}'(t[0/x_i], i + 1)$   
6:            $h \leftarrow \text{build}'(t[1/x_i], i + 1)$   
7:           return makenode( $H, \text{max}, b, i, l, h$ )  
8:  end build'  
9:  
1:   $H \leftarrow \text{emptytable}$    $\text{max} \leftarrow 1$   
10:  $b.\text{root} \leftarrow \text{build}'(t, 1)$   
11: return  $b$ 
```

Run time?

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)),
13:                               app(high(u1), high(u2)))
14:      else if var(u1) < var(u2) then
15:         res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
16:      else (* var(u1) > var(u2) *)
17:         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)),
13:                             app(high(u1), high(u2)))
14:          else if var(u1) < var(u2) then
15:             res ← makenode(var(u1), app(low(u1), u2), app(high(u1), u2))
16:          else (* var(u1) > var(u2) *)
17:             res ← makenode(var(u2), app(u1, low(u2)), app(u1, high(u2)))
18:             G(u1, u2) ← res
19:             return res
20:
21: forall i ≤ max(b1), j ≤ max(b2) : G(i, j) ← empty
22: b.root ← app(b1.root, b2.root)
23: return b
```

Other operations

Restrict	$b[v/x]$
Size (satcount)	$size(b) = \{\rho \mid b[\rho] = 1\} $
Anysat	$anysat(b) = \rho$, for some ρ with $b[\rho] = 1$
Allsat	$allsat(b) = \{\rho \mid b[\rho] = 1\}$
Compose	$compose(b, x, b') = b[x/b']$
Existential quantification	$\exists x.b = b[x/0] \vee b[x/1]$

Constraint Solving & Analysis & IBEN

Mia's skema

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
8-9	mat	eng	dan	tys	eng		
9-10	mat	tys	dan	geo	tys		
10-11	eng	dan	tys	dan	tys		
11-12	dan	dan	bio	mat	gym		
12-13	gym	fys	fys	fys	gym	gym	
13-14			bio	geo			
14-15			bio				

Boolean variables

=====

```
vars d1 d2 d3;
```

```
vars t1 t2 t3;
```

```
vars f1 f2 f3;
```

```
vars e1 e2 e3;
```


--Encodning of days --

=====

man := d1 & d2 & d3;

tir := d1 & d2 & !d3;

ons := d1 & !d2 & d3;

tor := d1 & !d2 & !d3;

fre := !d1 & d2 & d3;

lor := !d1 & d2 & !d3;

xxx := !d1 & !d2 & d3;

son := !d1 & !d2 & !d3;

uge := man + tir + ons + tor + fre;

weekend := lor + xxx + son;

--Encodning of hours--

=====

h1 := t1 & t2 & t3;

h2 := t1 & t2 & !t3;

h3 := t1 & !t2 & t3;

h4 := t1 & !t2 & !t3;

h5 := !t1 & t2 & t3;

h6 := !t1 & t2 & !t3;

h7 := !t1 & !t2 & t3;

h8 := !t1 & !t2 & !t3;

formiddag := h1 + h2 + h3 + h4;

aftermiddag := ! formiddag;

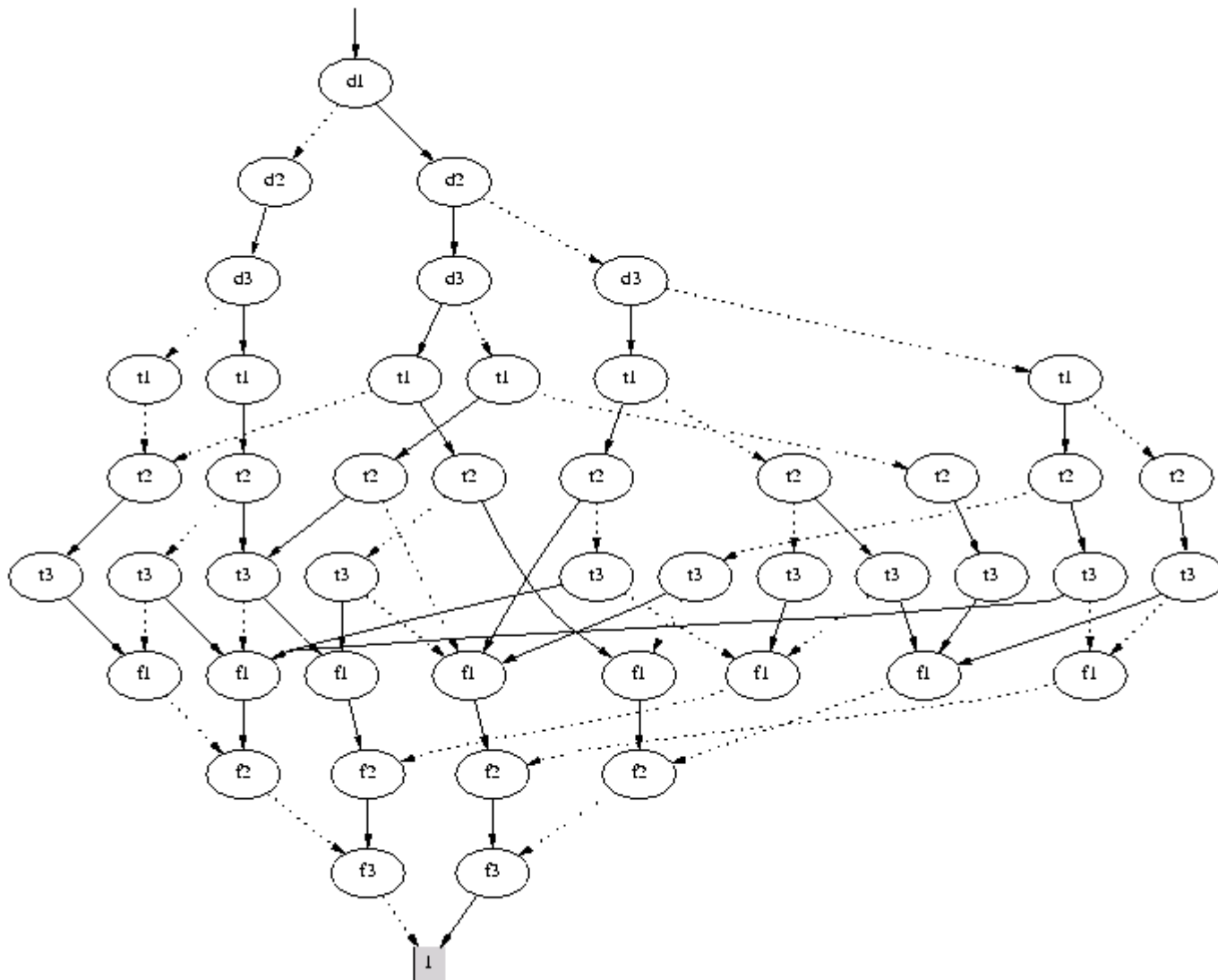
--Encodning of topic

```
dan := f1 & f2 & f3;  
eng := f1 & f2 & !f3;  
mat := f1 & !f2 & f3;  
tys := f1 & !f2 & !f3;  
geo := !f1 & f2 & f3;  
bio := !f1 & f2 & !f3;  
fys := !f1 & !f2 & f3;  
gym := !f1 & !f2 & !f3;
```

--Mia's skema--

=====

```
skema := man & h1 & mat +
        man & h2 & mat +
        man & h3 & eng +
        man & h4 & dan +
        man & h5 & gym +
        tir & h1 & eng +
        tir & h2 & tys +
        tir & h3 & dan +
        tir & h4 & dan +
        tir & h5 & fys +
        ons & h1 & dan +
        ons & h2 & dan +
        ons & h3 & tys +
        ons & h4 & bio +
        ons & h5 & fys +
        ons & h6 & bio +
        ons & h7 & bio +
        tor & h1 & tys +
        tor & h2 & geo +
        tor & h3 & dan +
        tor & h4 & mat +
        tor & h5 & fys +
        tor & h6 & geo +
        fre & h1 & eng +
        fre & h2 & tys +
        fre & h3 & tys +
        fre & h4 & gym +
        lor & h5 & gym;
```



--Various questions --

=====

q1 := (skema & mat) => formiddag;

q2 := (skema & fys) => eftermiddag;

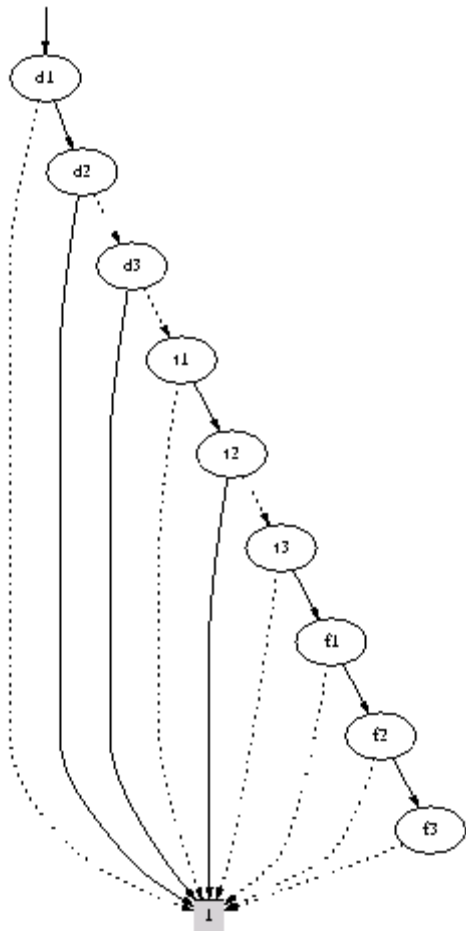
q3 := (skema & dan) => (man + tir + ons);

q4 := (skema & gym) => uge;

konfliktfri :=

((skema & (subst [e1/f1 e2/f2 e3/f3] (skema))) =>
((e1=f1) & (e2=f2) & (e3=f3)));





An important puzzle

Per, Kristian, Ole and Jens are to hold an Xmas-party.

Unfortunately, they are almost out of money which severely limits the amount of beer at the party.

In fact, they have to make do with 1 Tuborg, 1 Carlsberg, 1 Xmas (a Danish Xmas beer), and one Carls Special.

However, the four guys have individual requirements which must be fulfilled at all costs. In particular,

Per only drinks Tuborg and Carlsberg;

Kristian only drinks Carlsberg and Xmas;

Ole essentially drinks everything except Xmas,

and Jens can only drink Carlsberg

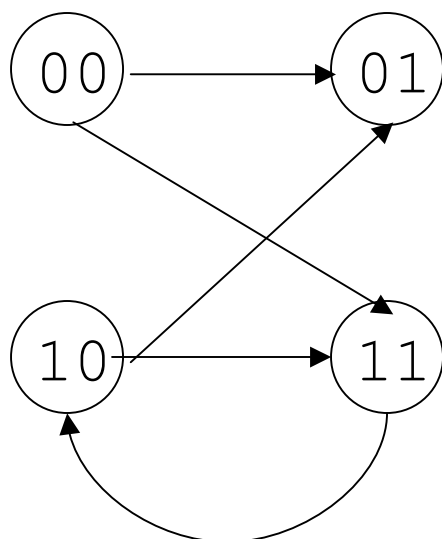
and Carls Special.

Is it possible to plan the party drinking so that they all get something to drink?

ROBDDs and Verification

[...,McMillan'90,.....,VVS'97]

ROBDD encoding of transition system

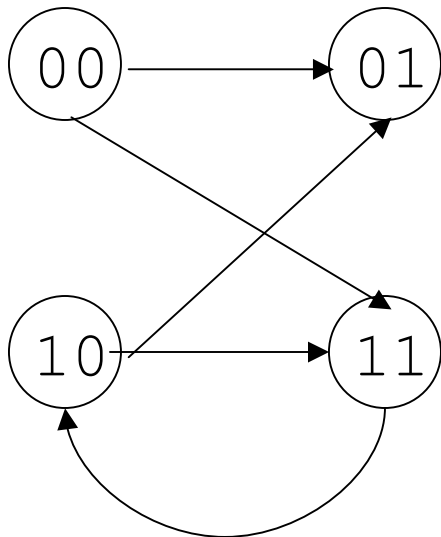


Encoding of states using binary variables (here x_1 and x_2).

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

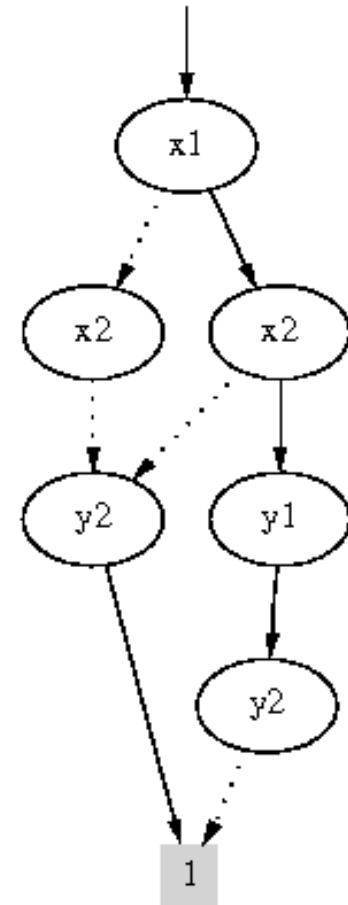
```
Trans ( $x_1, x_2, y_1, y_2$ ) :=  
    ! $x_1$  & ! $x_2$  & ! $y_1$  &  $y_2$   
  + ! $x_1$  & ! $x_2$  &  $y_1$  &  $y_2$   
  +  $x_1$  & ! $x_2$  & ! $y_1$  &  $y_2$   
  +  $x_1$  & ! $x_2$  &  $y_1$  &  $y_2$   
  +  $x_1$  &  $x_2$  &  $y_1$  & ! $y_2$ ;
```

ROBDD representation (cont.)



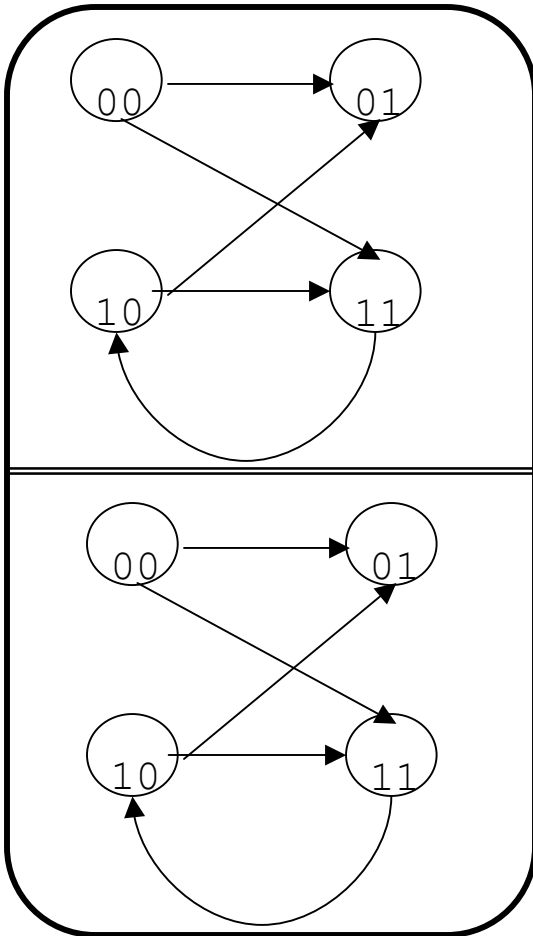
```

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

ATrans (\mathbf{x}, \mathbf{y})



BTrans (\mathbf{u}, \mathbf{v})

Asynchronous composition

$$\begin{aligned} \text{Trans}(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) = & \\ & (\text{ATrans}(\mathbf{x}, \mathbf{y}) \ \& \ \mathbf{v}=\mathbf{u}) \\ & + (\text{BTrans}(\mathbf{u}, \mathbf{v}) \ \& \ \mathbf{y}=\mathbf{x}) \end{aligned}$$

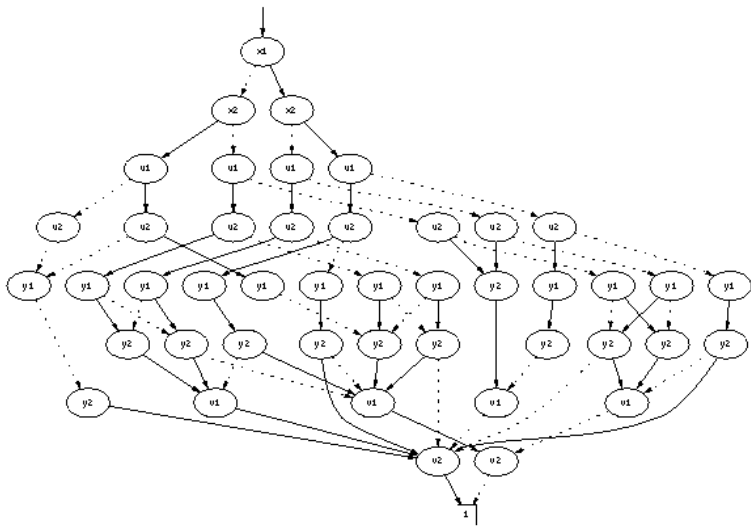
Synchronous composition

$$\begin{aligned} \text{Trans}(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}) = & \\ & (\text{ATrans}(\mathbf{x}, \mathbf{y}) \ \& \ \text{BTrans}(\mathbf{u}, \mathbf{v})) \end{aligned}$$

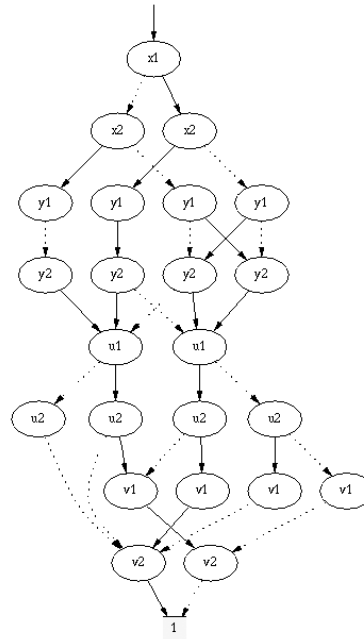
Which ordering to choose?

Ordering?

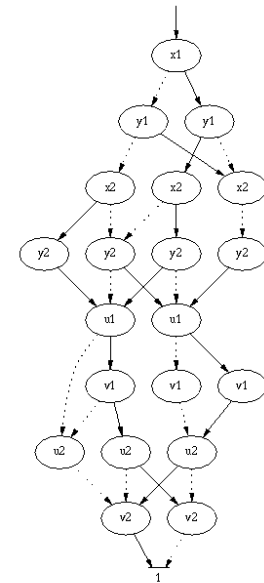
45 nodes
 $x_1, x_2, u_1, u_2, y_1, y_2, v_1, v_2$



23 nodes
 $x_1, x_2, y_1, y_2, u_1, u_2, v_1, v_2$



20 nodes
 $x_1, y_1, x_2, y_2, u_1, v_1, u_2, v_2$



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

Reachable States

$\text{Reach}(\mathbf{x}) := \text{Init}(\mathbf{x});$

REPEAT

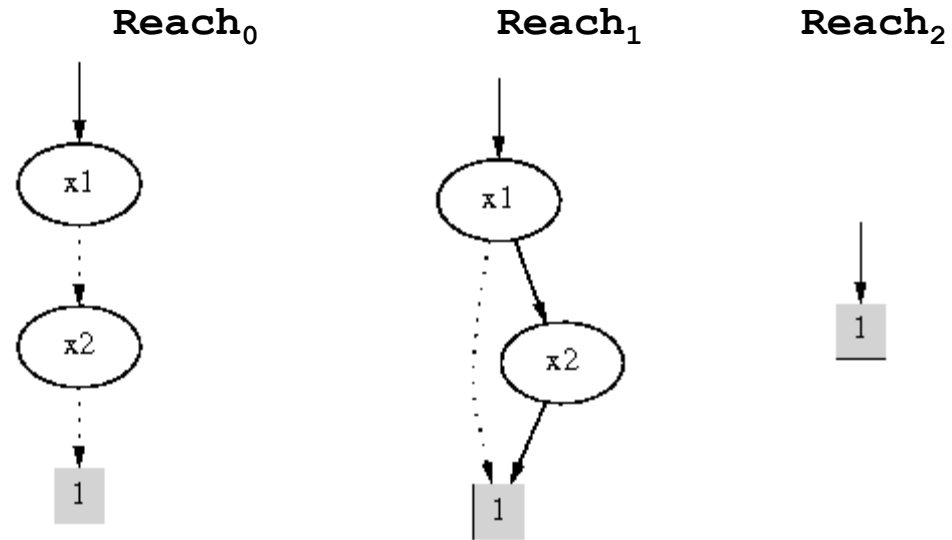
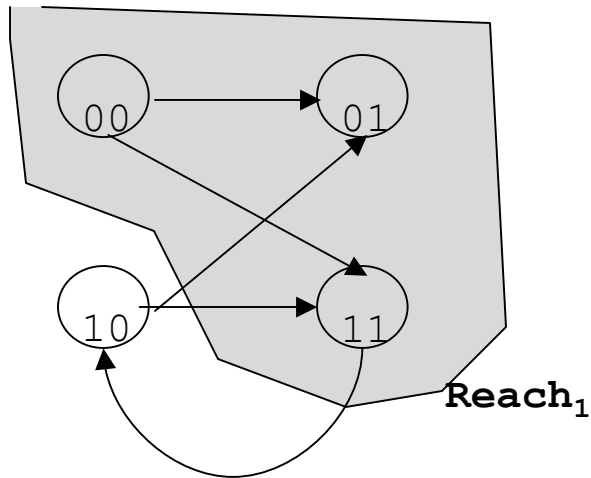
$\text{Old}(\mathbf{x}) := \text{Reach}(\mathbf{x});$

$\text{New}(\mathbf{y}) := \text{Exists } \mathbf{x}. (\text{Reach}(\mathbf{x}) \ \& \ \text{Trans}(\mathbf{x}, \mathbf{y}));$

$\text{Reach}(\mathbf{x}) := \text{Old}(\mathbf{x}) + \text{New}(\mathbf{x})$

UNTIL $\text{Old}(\mathbf{x}) = \text{Reach}(\mathbf{x})$

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



A MUTEX Algorithm

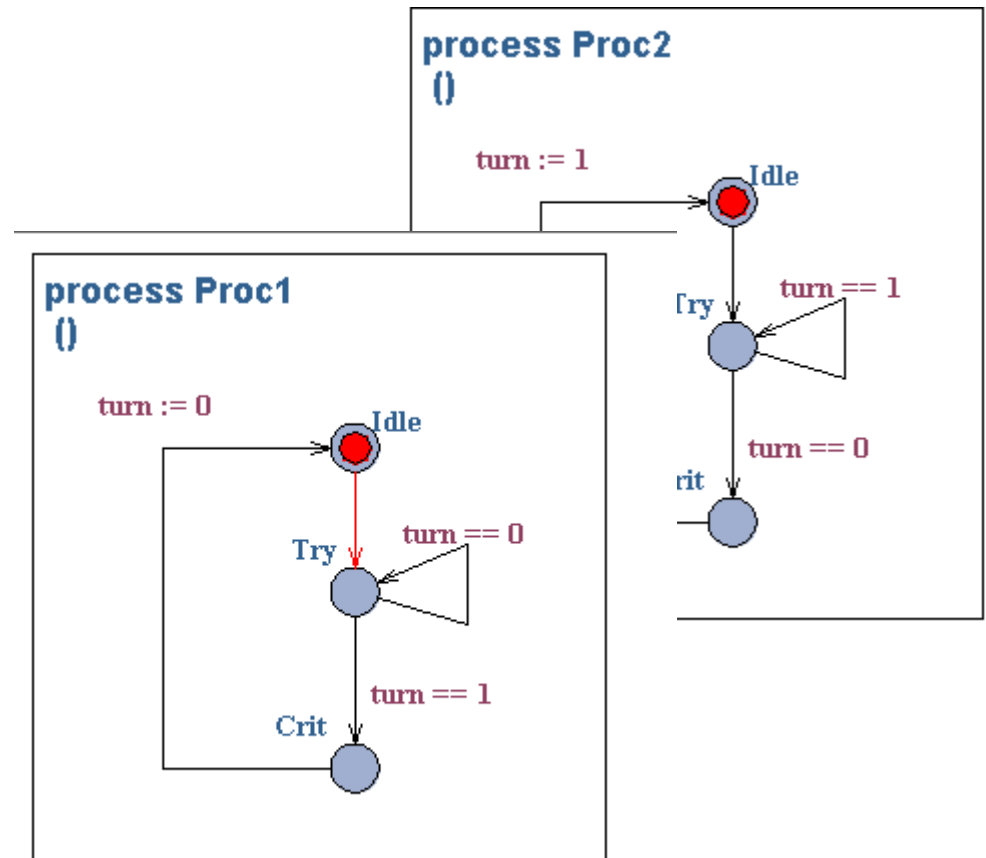
Clarke & Emerson

```
P1 :: while True do
  T1 : wait(turn=1)
  C1 : turn:=0
endwhile

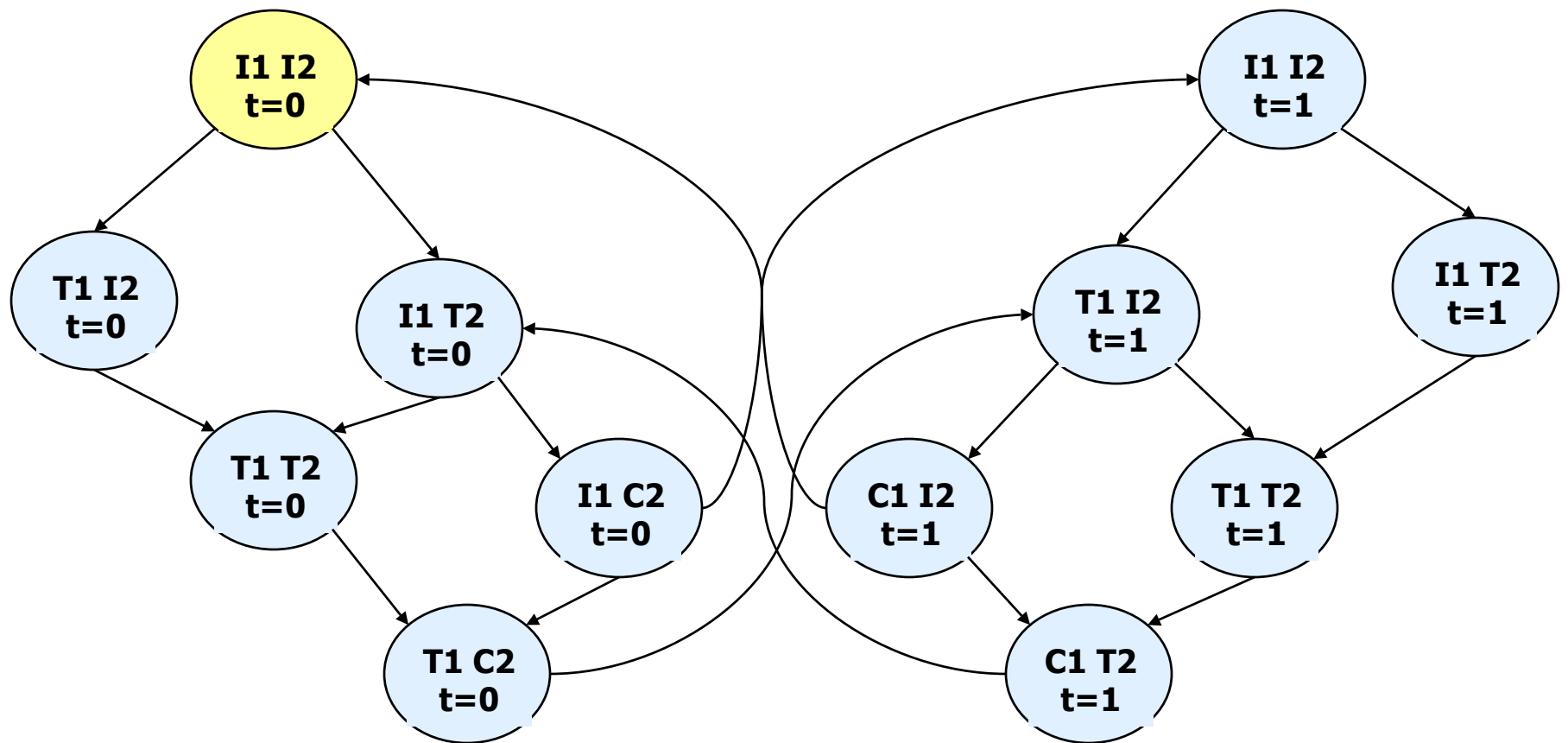
||

P2 :: while True do
  T2 : wait(turn=0)
  C2 : turn:=1
endwhile
```

Mutual Exclusion Program



Global Transition System



A MUTEX Algorithm

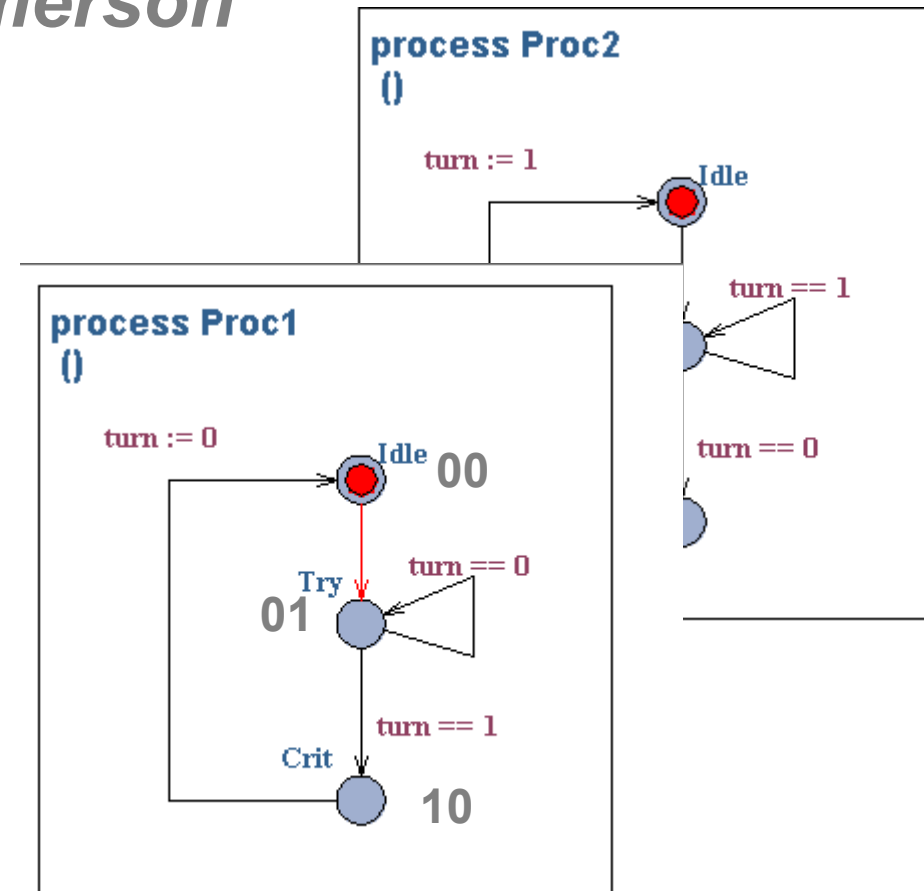
Clarke & Emerson

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

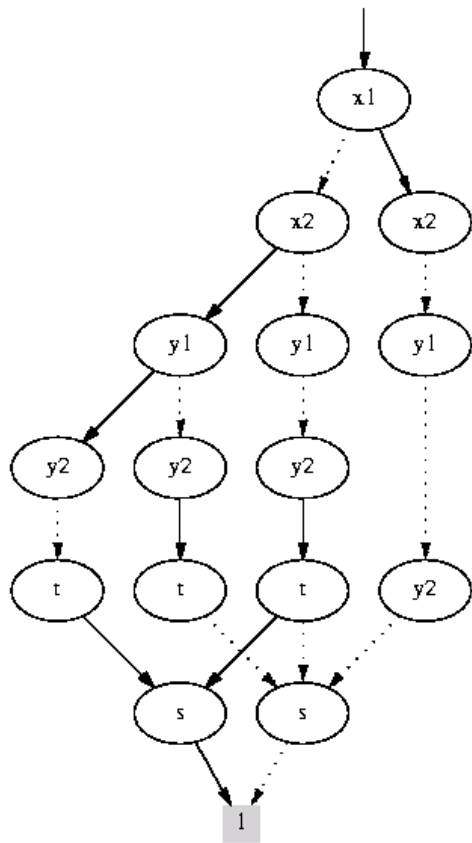
```
ATrans := (!x1 & !x2 & !y1 & y2 & (s=t))  
        + (!x1 & x2 & !y1 & y2 & !t & !s)  
        + (!x1 & x2 & y1 & !y2 & t & s)  
        + (x1 & !x2 & !y1 & !y2 & !s);
```

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

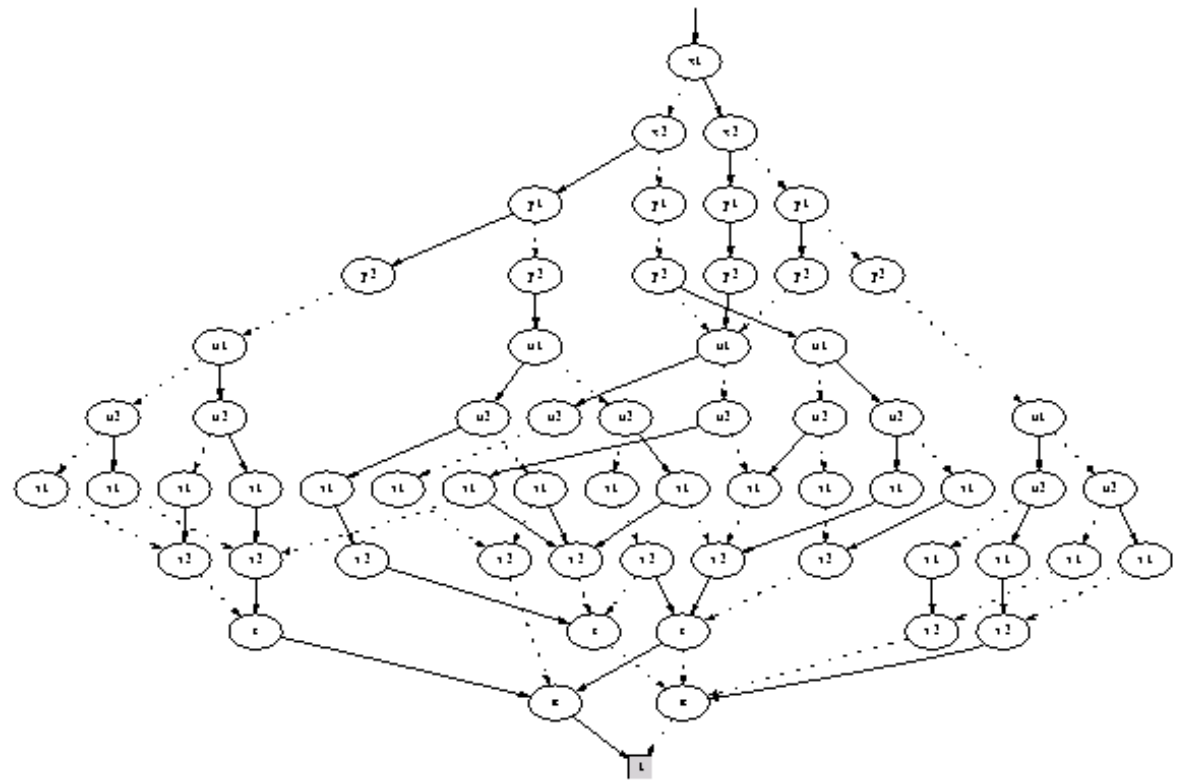
```
TT := (ATrans & (u1=v1) & (u2=v2))  
      + (BTrans & (x1=y1) & (x2=y2));
```



BDDs for Transition Relations



ATrans



TT

Reachable States

$\text{Reach}(\mathbf{x}) := \text{Init}(\mathbf{x});$

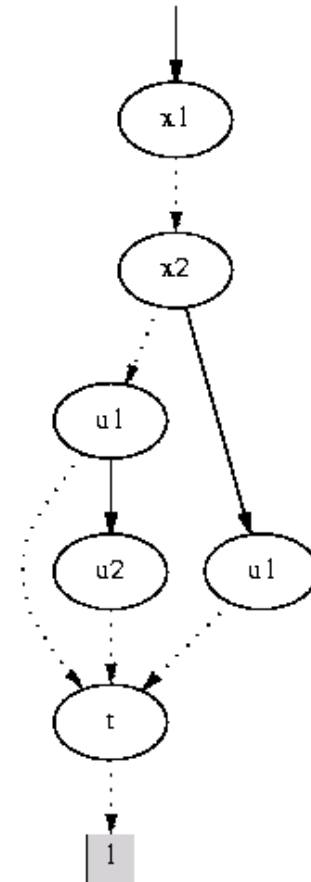
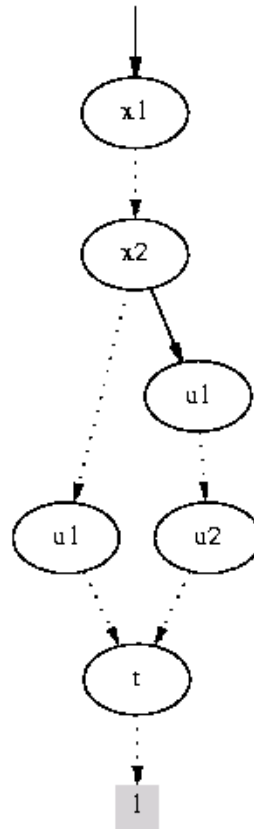
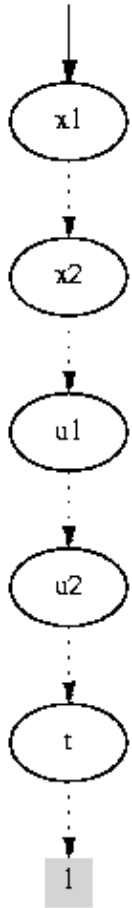
REPEAT

$\text{Old}(\mathbf{x}) := \text{Reach}(\mathbf{x});$

$\text{New}(\mathbf{y}) := \text{Exists } \mathbf{x}. (\text{Reach}(\mathbf{x}) \ \& \ \text{Trans}(\mathbf{x}, \mathbf{y}));$

$\text{Reach}(\mathbf{x}) := \text{Old}(\mathbf{x}) + \text{New}(\mathbf{x})$

UNTIL $\text{Old}(\mathbf{x}) = \text{Reach}(\mathbf{x})$



Reachable States

$\text{Reach}(\mathbf{x}) := \text{Init}(\mathbf{x});$

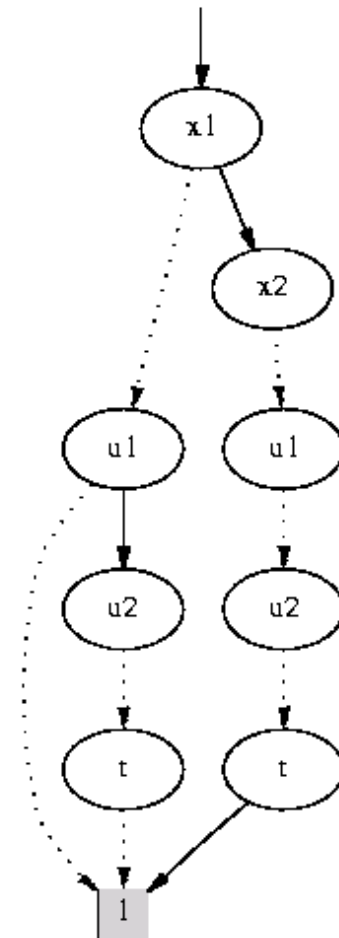
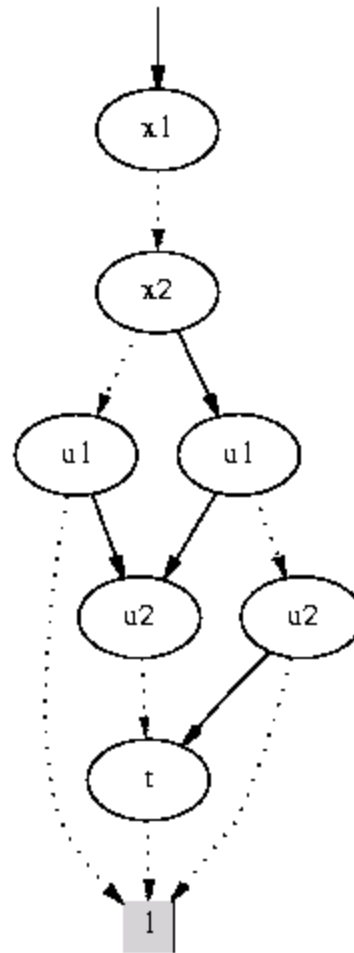
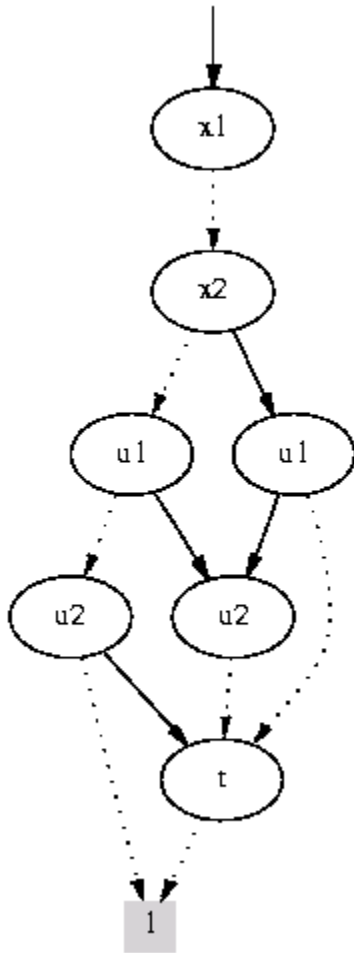
REPEAT

$\text{Old}(\mathbf{x}) := \text{Reach}(\mathbf{x});$

$\text{New}(\mathbf{y}) := \text{Exists } \mathbf{x}. (\text{Reach}(\mathbf{x}) \ \& \ \text{Trans}(\mathbf{x}, \mathbf{y}));$

$\text{Reach}(\mathbf{x}) := \text{Old}(\mathbf{x}) + \text{New}(\mathbf{x})$

UNTIL $\text{Old}(\mathbf{x}) = \text{Reach}(\mathbf{x})$

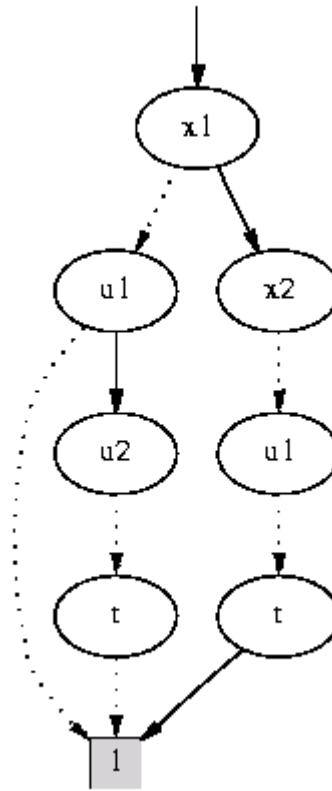
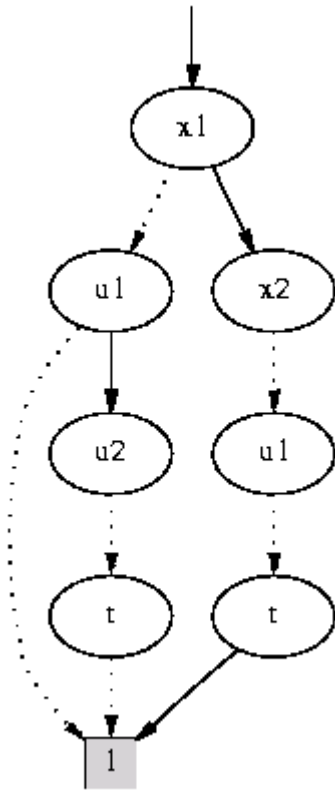


Reachable States

```

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

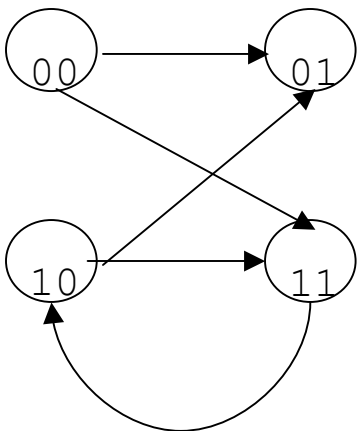
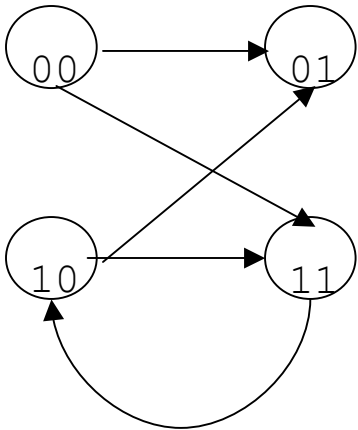
MUTEX ?

Reach &
 x_1 & $!x_2$ &
 u_1 & $!u_2$

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**))

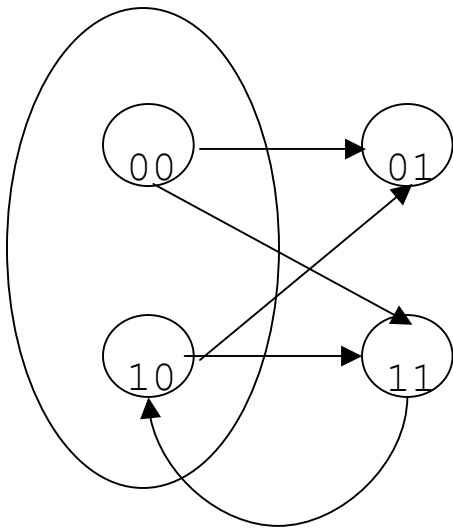
 &

 Forall **v**. Trans(**u**,**v**) =>

 (Exists **y**. Trans(**x**,**y**) & Bis(**y**,**v**));

UNTIL Bis(**x**,**u**)=Old(**x**,**u**)

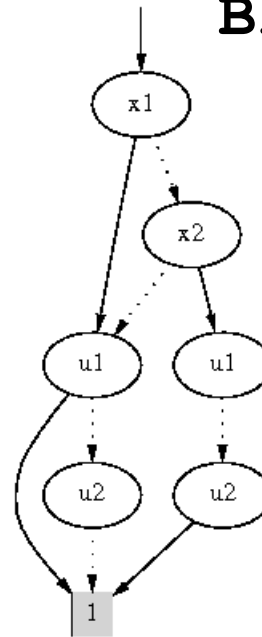
Bisimulation (cont.)



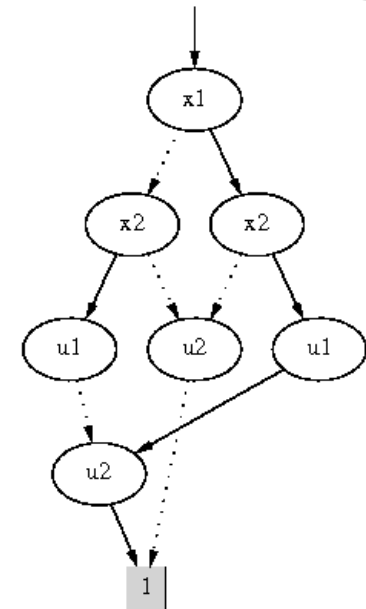
Bis₀



Bis₁

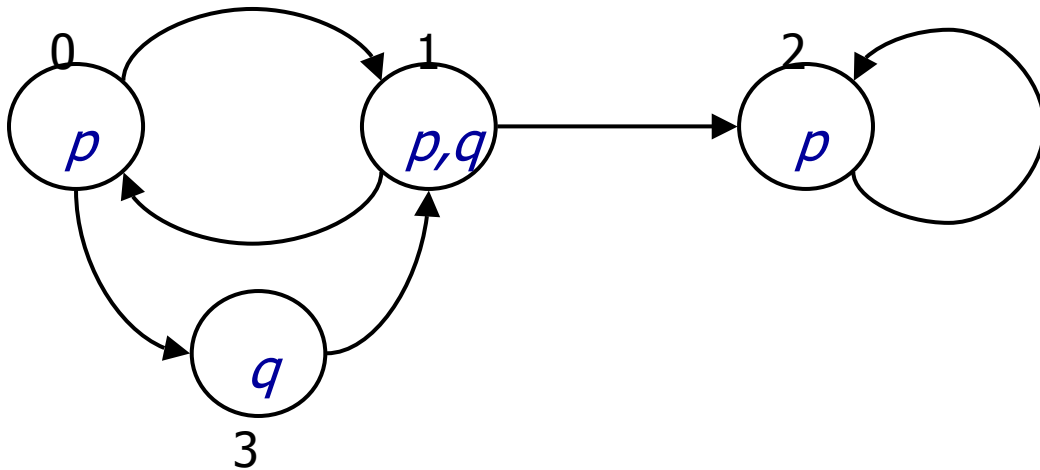


Bis₂



3 equivalence classes
= 6 pairs in final bisimulation

Model Checking



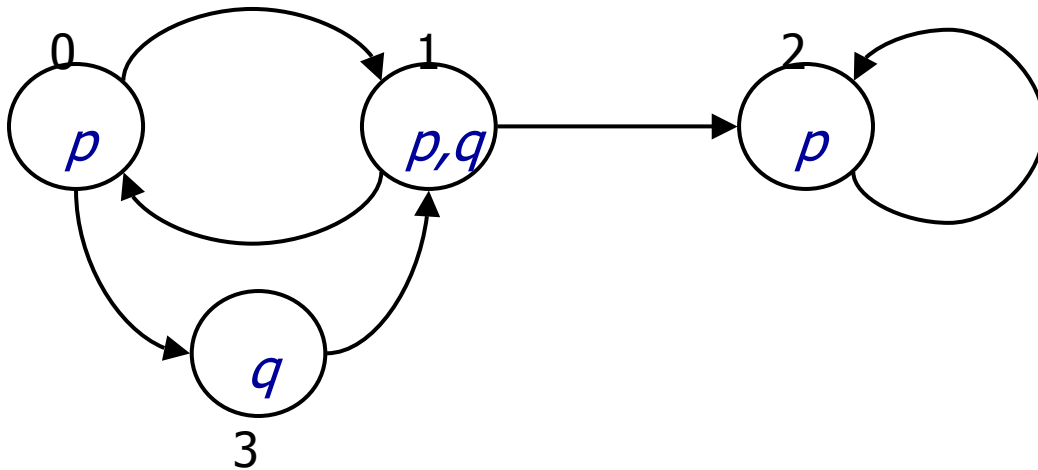
```
vars x1 x2;
vars y1 y2;
```

```
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) := ..... ;
```

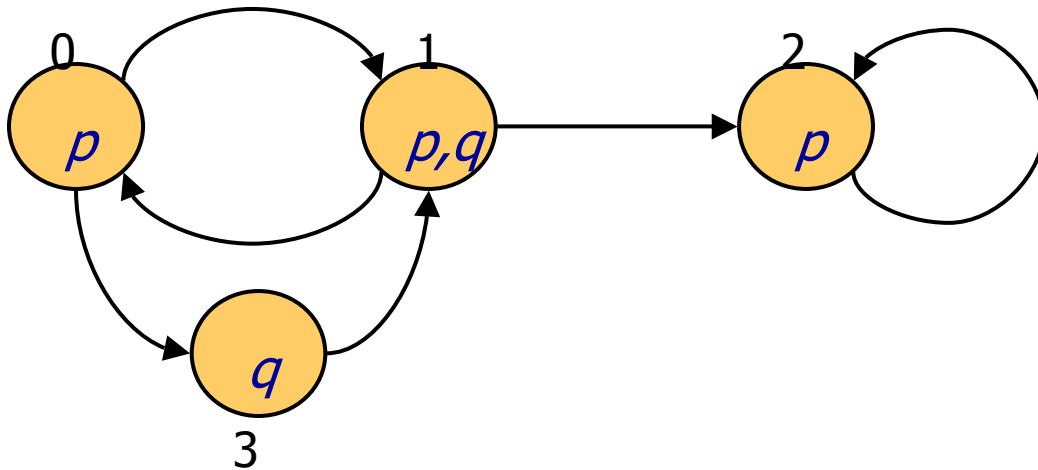
Model Checking



$\langle \rangle P$

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

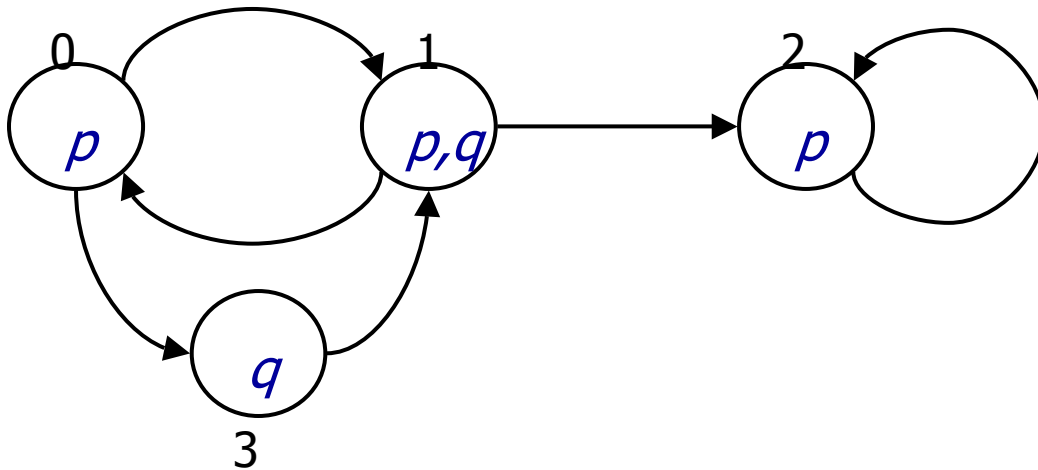
Model Checking



$\langle \rangle P$

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

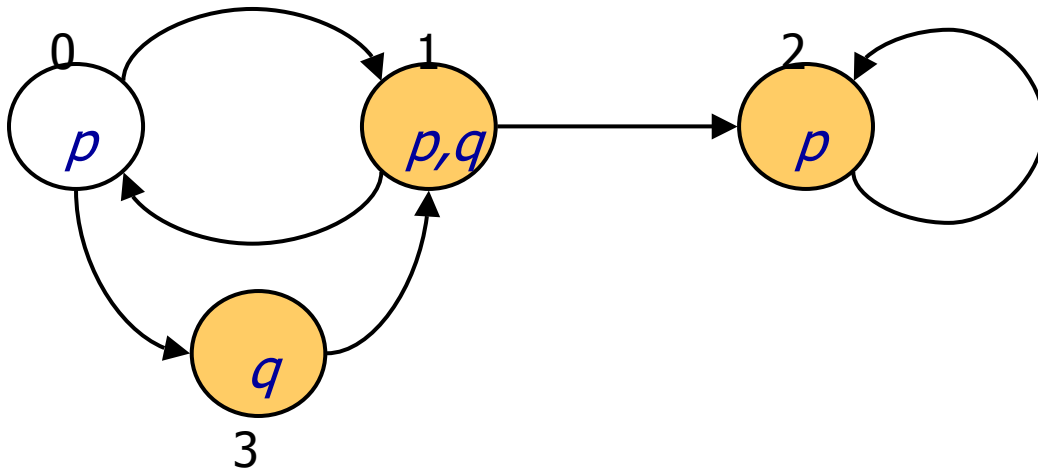
Model Checking



[] P

Forall $y1, y2$.
Trans ($x1, x2, y1, y2$) \Rightarrow
P ($y1, y2$);

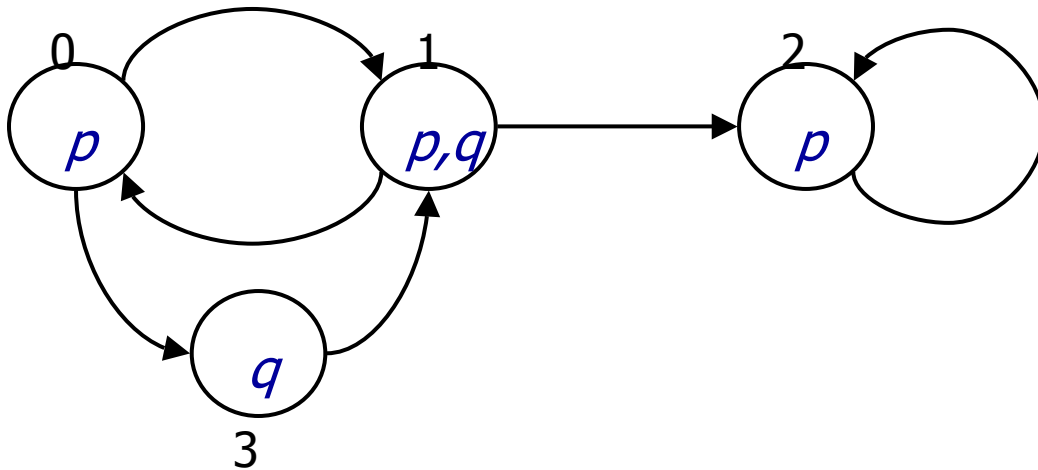
Model Checking



[] P

Forall $y1, y2$.
Trans ($x1, x2, y1, y2$) \Rightarrow
P ($y1, y2$);

Model Checking

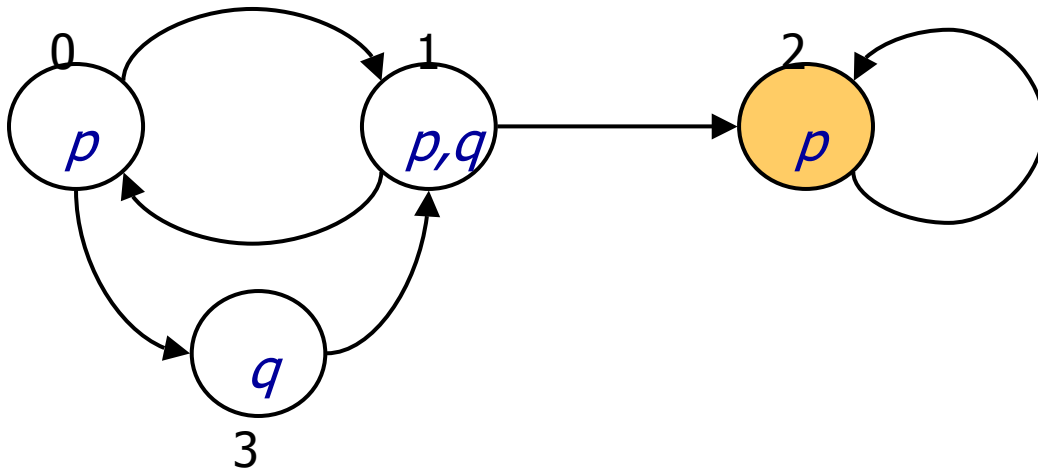


ALWAYS P

max fixpoint

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

Model Checking

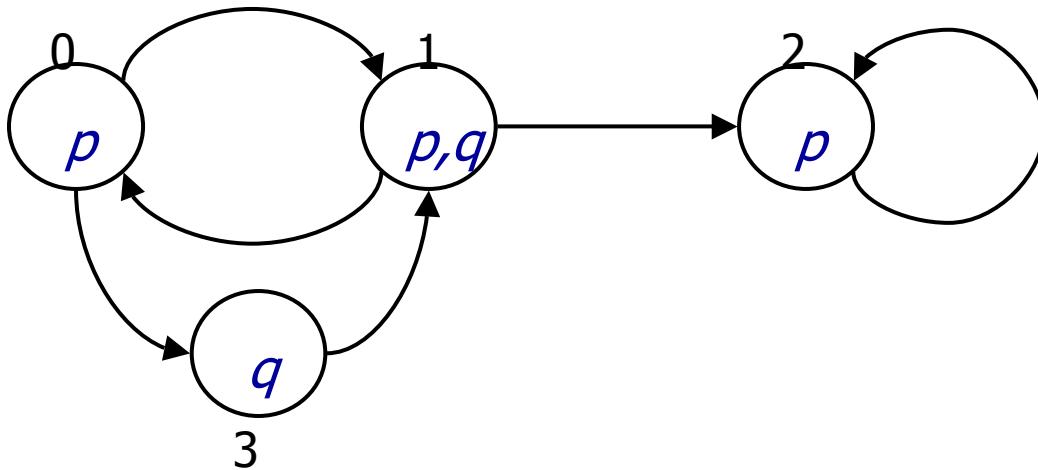


ALWAYS P

max fixpoint

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

Model Checking

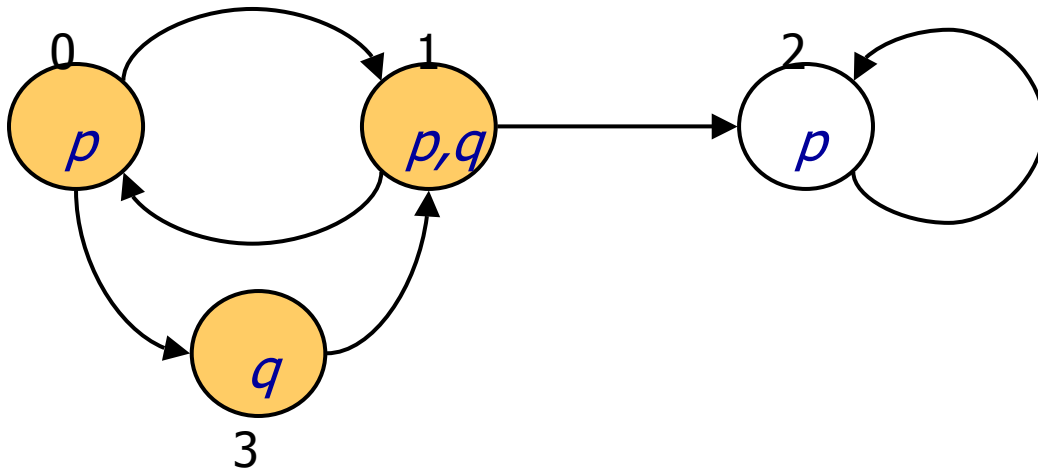


P UNTIL Q

min fixpoint

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


Model Checking



P UNTIL Q

min fixpoint

$$\begin{aligned}
 \mathbf{U}(x1, x2) = & \\
 & Q(x1, x2) + \\
 & \{ P(x1, x2) \ \& \\
 & \quad \text{Forall } y1, y2. \\
 & \quad \text{Trans}(x1, x2, y1, y2) \Rightarrow \\
 & \quad \mathbf{U}(y1, y2) \\
 & \};
 \end{aligned}$$

Partitioned Transition Relation

Relational Product

LARGE

$$\text{Exists } \mathbf{yv}. (\text{T}(\mathbf{xu}, \mathbf{yv}) \ \& \ S(\mathbf{yv}))$$

Asynchronous

Synchronous

$$\text{T}(\mathbf{xu}, \mathbf{yv}) =$$

$$\begin{aligned} & (\text{ATrans}(\mathbf{x}, \mathbf{y}) \ \& \ \mathbf{v}=\mathbf{u}) \\ + & (\text{BTrans}(\mathbf{u}, \mathbf{v}) \ \& \ \mathbf{y}=\mathbf{x}) \end{aligned}$$

$$\text{T}(\mathbf{xu}, \mathbf{yv}) =$$

$$\begin{aligned} & \text{ATrans}(\mathbf{x}, \mathbf{y}) \\ & \& \ \text{BTrans}(\mathbf{u}, \mathbf{v}) \end{aligned}$$

$$\begin{aligned} & \text{Exists } \mathbf{y}. \text{ATrans}(\mathbf{x}, \mathbf{y}) \ \& \ S(\mathbf{yu}) \\ + & \\ & \text{Exists } \mathbf{v}. \text{BTrans}(\mathbf{u}, \mathbf{v}) \ \& \ S(\mathbf{xv}) \end{aligned}$$

$$\begin{aligned} & \text{Exists } \mathbf{yv}. \\ & \text{Atrans}(\mathbf{x}, \mathbf{y}) \\ & \& \ \text{Btrans}(\mathbf{u}, \mathbf{v}) \ \& \ S(\mathbf{yv}) \end{aligned}$$

$$\begin{aligned} & \text{Exists } \mathbf{y}. \\ & \text{Atrans}(\mathbf{x}, \mathbf{y}) \\ & \& \ (\text{Exists } \mathbf{v}. \text{Btrans}(\mathbf{u}, \mathbf{v}) \ \& \ S(\mathbf{yv})) \end{aligned}$$

BDD

visualSTATE

CIT project VVS (w DTU)



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*

- *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

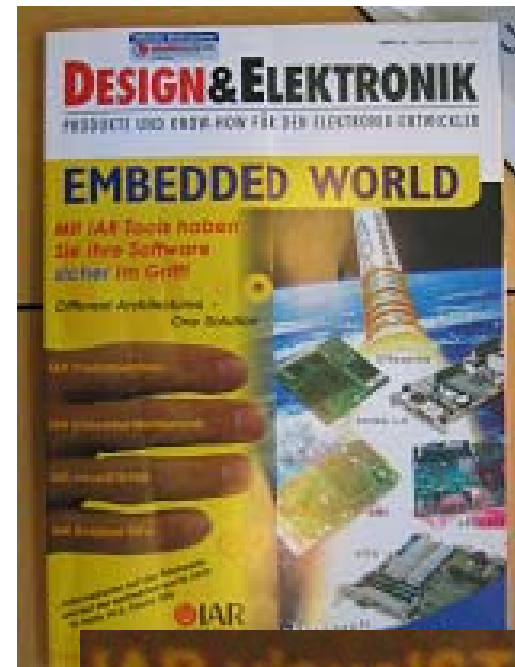
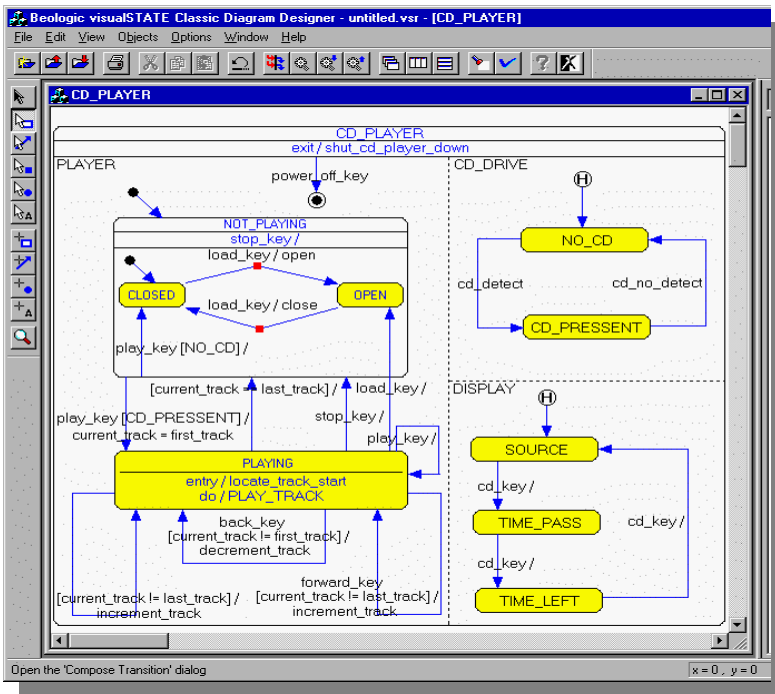
Siemens (approx. 200) BDDs & Verification

Verification Problems:

- *1.400 components*
- *10⁴⁰⁰ states*

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

visualSTATE



IAR visualSTATE®

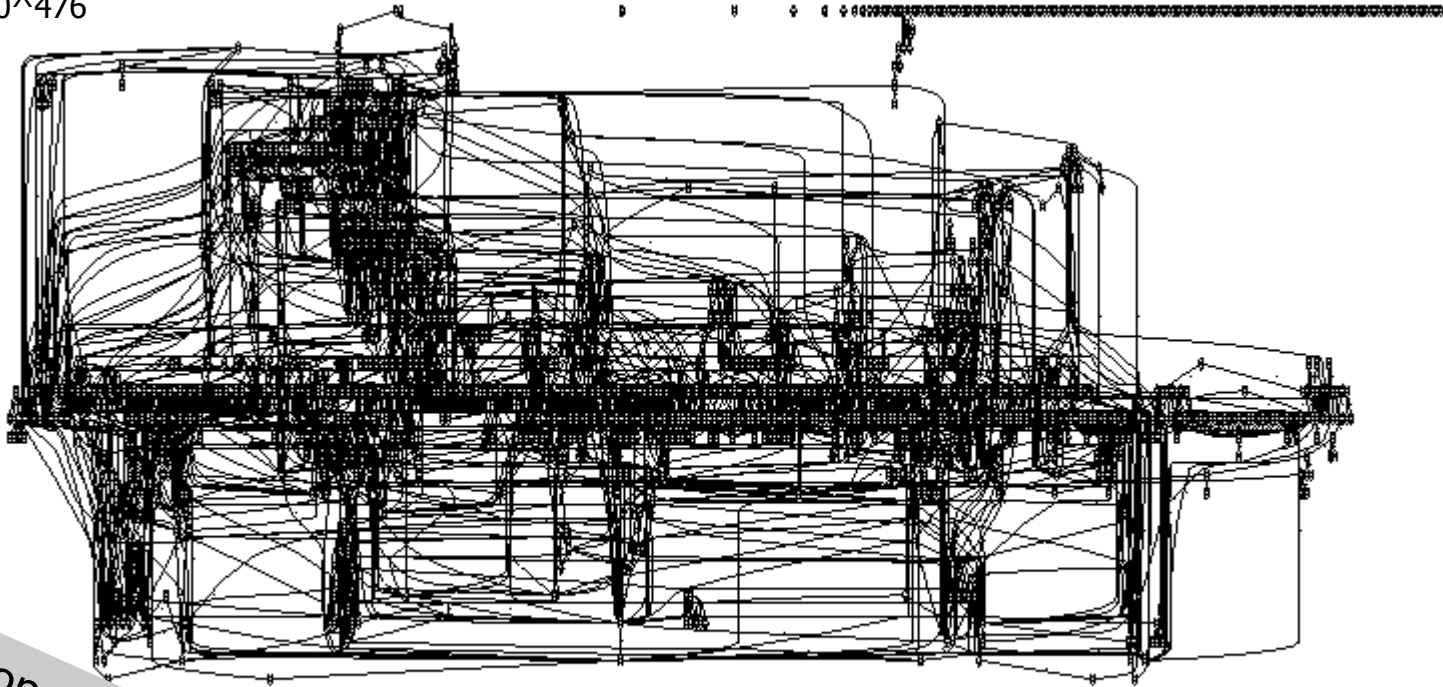


Control Programs

A Train Simulator, visualSTATE (VVS)

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

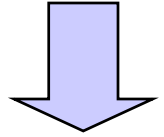
BUGS ?



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

Experimental Breakthroughs

Patented



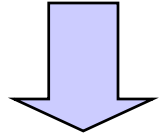
System	Mach.	State Space		Checks	Visual ST	St-of-Art		ComBack	
		Declared	Reach			Sec	MB	Sec	MB
VCR	7	10 ⁵	1279	50	<1	<1	6	<1	7
JVC	8	10 ⁴	352	22	<1	<1	6	<1	6
HI-FI	9	10 ⁷	1416384	120	1200	1.0	6	3.9	6
Motor	12	10 ⁷	34560	123	32	<1	6	2,0	
AVS	12	10 ⁷	1438416	173	3780	6.7	6	5.7	6
Video	13	10 ⁸	1219440	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	10¹³⁶	---	1335	---	---	---	25.9	6
Train2	1421	10⁴⁷⁶	---	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	S+	t	ComBack	
		Declared	Reach					Sec	MB
VCR	7	10 ⁵	1279	50				<1	7
JVC	8	10 ⁴	352					<1	6
HI-FI	9	10 ⁷	1416384					3.9	6
Motor	12	10 ⁷	-				0	2,0	
AVS	12	10 ⁷				6	6	5.7	6
Video	13	10 ⁷				1.1	6	1.5	6
Car	20				---	3.8	9	1.8	6
N6	11			143	---	32.3	7	218	6
N5				269	---	56.2	7	9.1	6
N4				132	---	622	7	6.3	6
Train1				---	---	---	---	25.9	6
Train2	14			---	---	---	---	739	11

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

Machine: 166 MHz Pentium PC with 32 MB RAM

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

SAT-solving

Davis-Putnam

ϕ in CNF, i.e. $\phi = (l_{1,1} \vee \dots \vee l_{1,n_1}) \wedge \dots \wedge (l_{k,1} \vee \dots \vee l_{k,n_k})$
 where $l_{i,j}$ is literal.



```

Function SAT( $\phi$ )
/unit propagation/
Repeat
  For each unit clause ( $l$ ) in  $\phi$ 
  do delete from  $\phi$  any clause containing  $l$ 
      delete  $!l$  from any clause in  $\phi$ 
  if  $\phi$  is empty return TRUE
  if  $\phi$  contains the empty clause return FALSE
Until no further change
/splitting/
Choose literal  $l$  occurring in  $\phi$  (from smallest clause)
if SAT( $\phi \wedge (l)$ ) then return TRUE
else if SAT( $\phi \wedge (!l)$ ) then return TRUE
else return FALSE
    
```

THEOREM

SAT(ϕ)=TRUE
 iff
 ϕ is satisfiable
 -
 SAT(ϕ)=FALSE
 iff
 ϕ is unsatisfiable
 -
 SAT always terminates

SAT-Solving

- A very active research area with phenomenal performance improvements, e.g.:
 - Analysis of Vital Processor Interlockings (The Netherlands): formulas of size 120K
 - Stålmarks method (PROVER \rightarrow Trusted Logic) – based on DP plus learning
 - HeerHugo
 - CHAFF – utilizing cash
- Work on SAT-solving for Timed Propositional logic:
$$\phi ::= a \mid x-y \leq m \mid \neg \phi \mid \phi \wedge \phi$$
- See Bulletin of EATCS, February 2005.