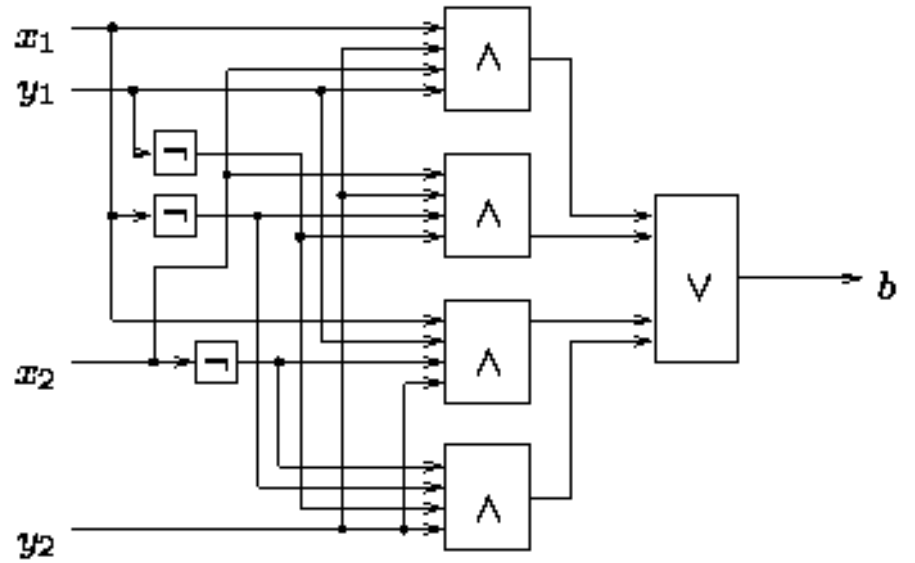
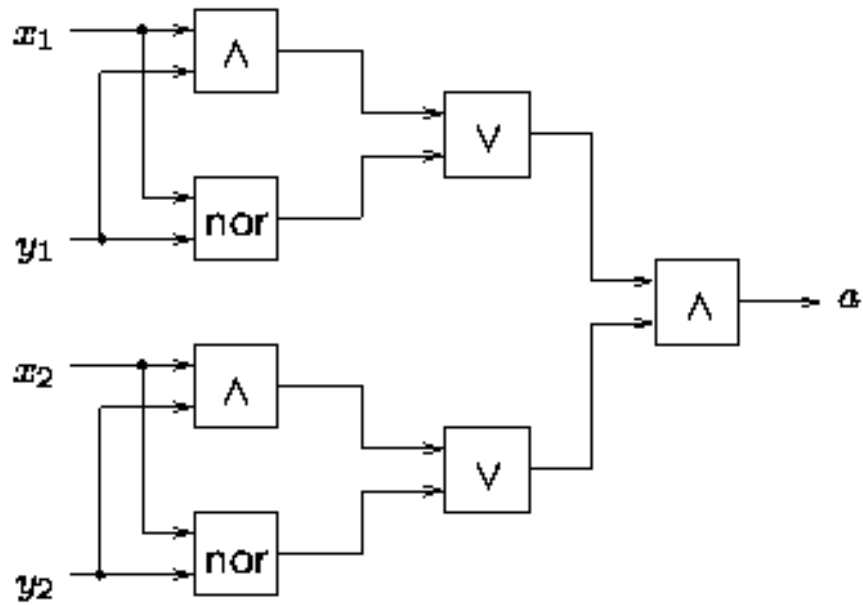


# Binary Decision Diagrams

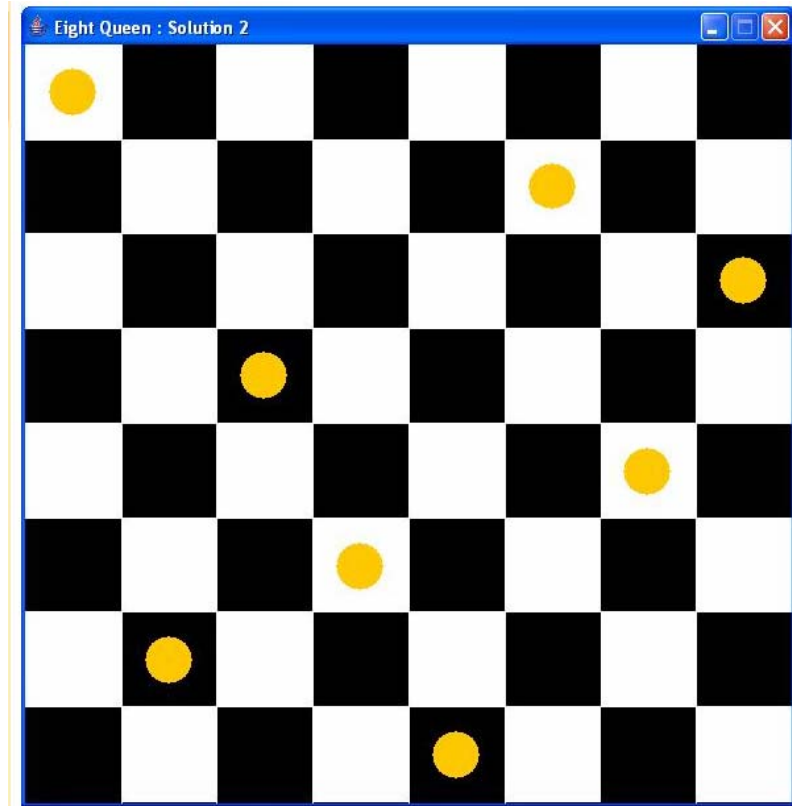
# Combinatorial Circuits



# 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**



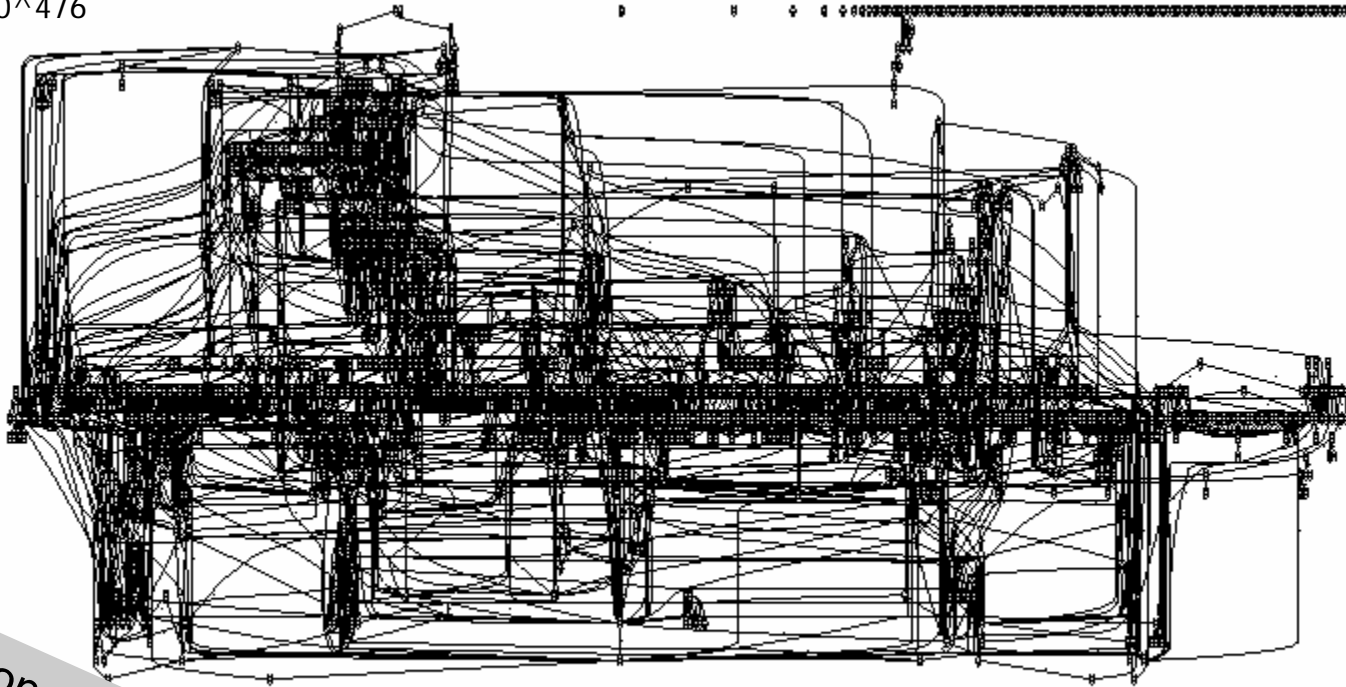
**Eight Queen**

# 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 representation of *boolean functions* allowing effective manipulation (satisfiability, validity,....)

**or**

- Compact representation of *sets* over finite universe allowing effective manipulations.

# Boolean Logic

# Boolean Functions

Boolean functions:  $\mathbb{B} = \{0, 1\}$ ,

$$f : \mathbb{B} \times \cdots \times \mathbb{B} \rightarrow \mathbb{B}$$

Boolean expressions:

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \Rightarrow t \mid t \Leftrightarrow t$$

Truth assignments:  $\rho$ ,

$$[v_1/x_1, v_2/x_2, \dots, v_n/x_n]$$

Satisfiable: Exists  $\rho$  such that  $t[\rho] = 1$

Tautology: For all  $\rho$ ,  $t[\rho] = 1$

# Truth Tables

	$\neg$
0	1
1	0

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\Rightarrow$	0	1
0	1	1
1	0	1

$\Leftrightarrow$	0	1
0	1	0
1	0	1

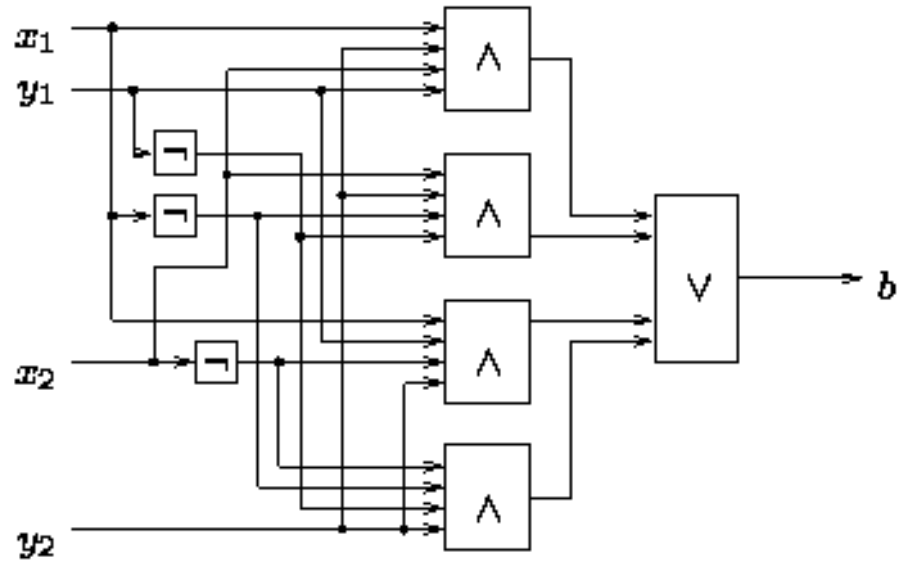
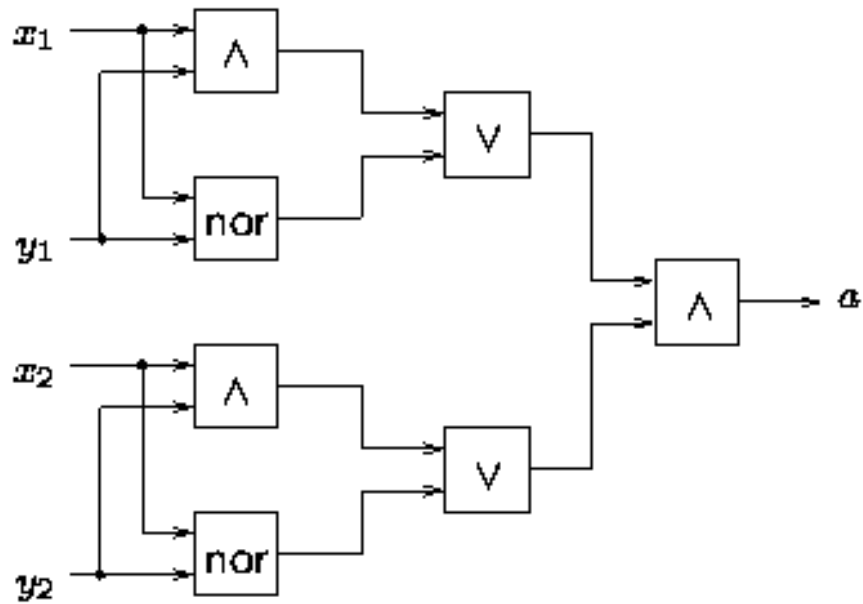
$xyz$	$x \rightarrow y, z$
000	0
001	1
010	0
011	1
100	0
101	0
110	1
111	1

$x_1 \cdots x_n$	$f(x_1, \dots, x_n)$
0...0	1
0...1	0
$\vdots$	$\vdots$
1...1	0

$2^n$  entries



# Combinatorial Circuits



Are they two circuits equivalent?

# “Good” Representations of Boolean Functions

Always perfect representations are hopeless

## Normalforms

- Disjunctive NF
- Conjunctive NF
- If-then-else NF
- .....

### THEOREM (Cook’s theorem)

*Satisfiability of Boolean expressions is NP-complete*

Compact representations are

- **compact** and
- **efficient**

on **real-life** examples

# If-Then-Else Operator

Let  $t$ ,  $t_1$  and  $t_2$  be boolean expressions.

## Syntax

$$t \rightarrow t_1, t_2$$

## Semantics

If-Then-Else operator  $t \rightarrow t_1, t_2$  is equivalent to  $(t \wedge t_1) \vee (\neg t \wedge t_2)$ .

$t$	$t_1$	$t_2$	$t \rightarrow t_1, t_2$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

# 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**.

# Binary Decision Structures

# 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

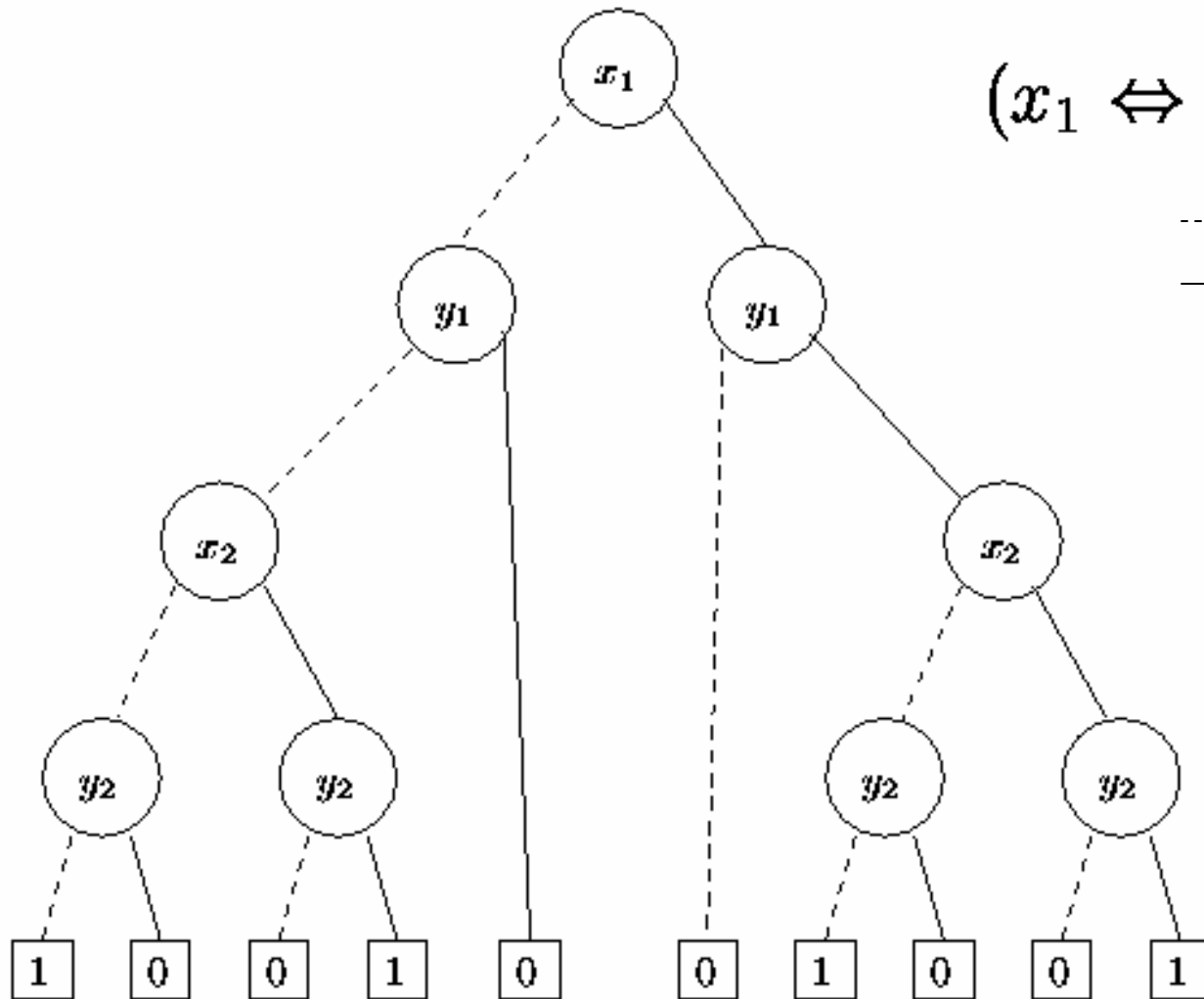
Let  $x$  be an arbitrary boolean variable. Any boolean expressions  $t$  is equivalent to

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

## Corollary

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

# Binary Decision *Trees*



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

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

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

# Binary Decision *Diagrams* allows NODES to be shared

**Equivalence**  $\sim$  on nodes:

$n \sim m$  **iff**

**either** both  $n$  and  $m$  are terminals  
and have the same value

**or** both are non-terminals with  $\text{var}(n) = \text{var}(m)$  **and**

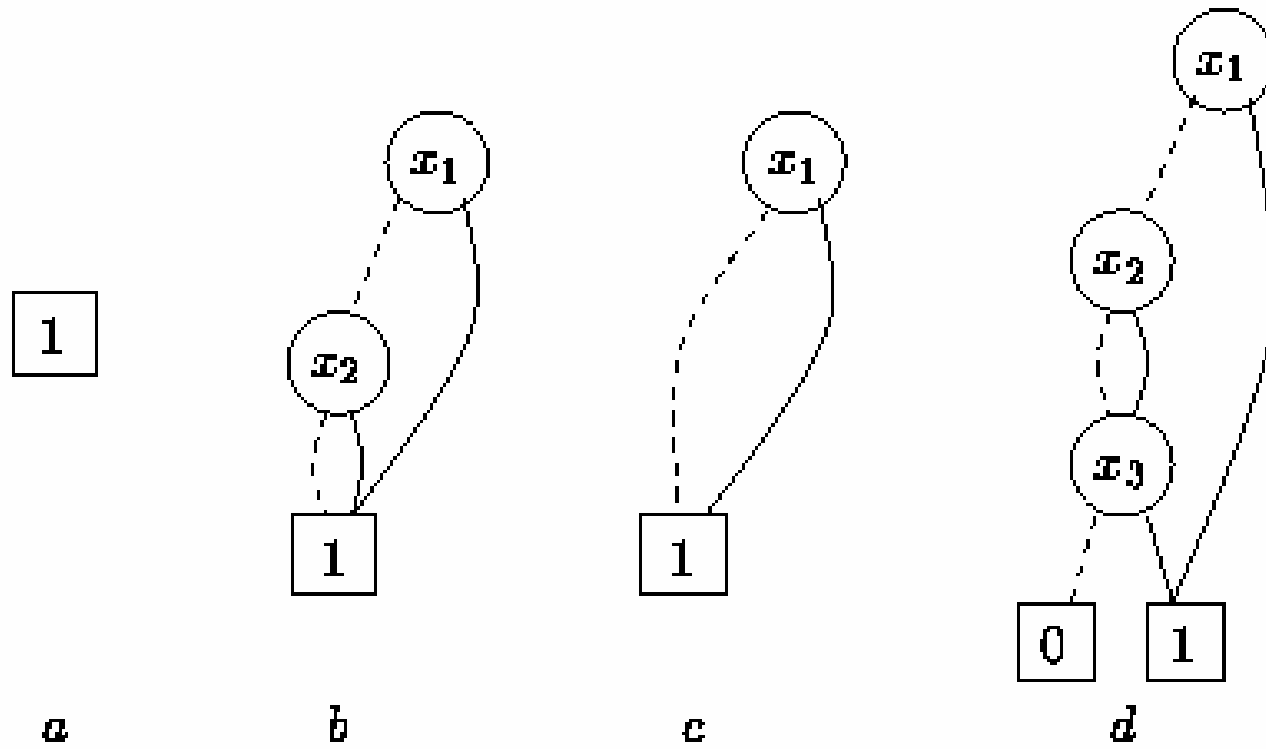
**1.**  $n' \sim m'$  **when**  $n \text{-}0\text{-}\rightarrow n'$ ,  $m \text{-}0\text{-}\rightarrow m'$ , **and**

**2.**  $n' \sim m'$  **when**  $n \text{-}1\text{-}\rightarrow n'$ ,  $m \text{-}1\text{-}\rightarrow m'$

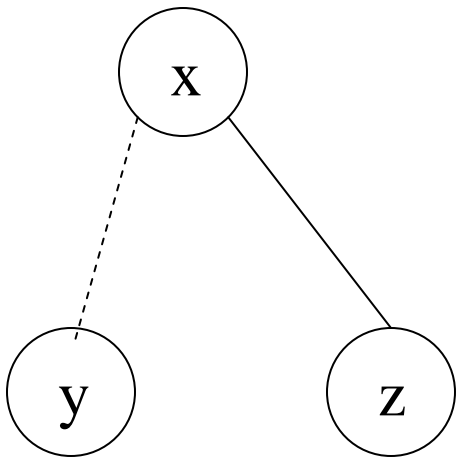
Have you seen this somewhere  
before ?



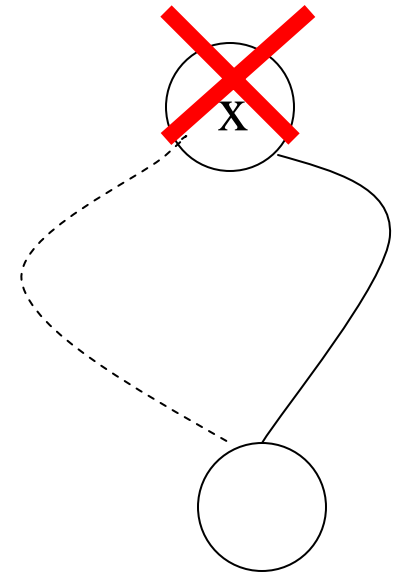
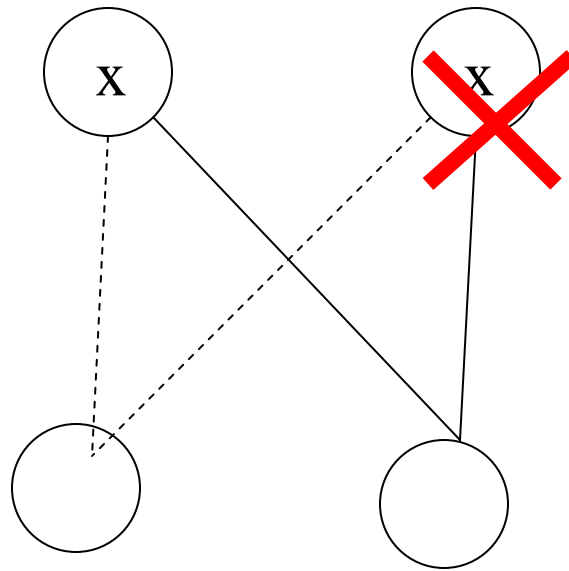
# Orderedness & Redundant *TESTS*



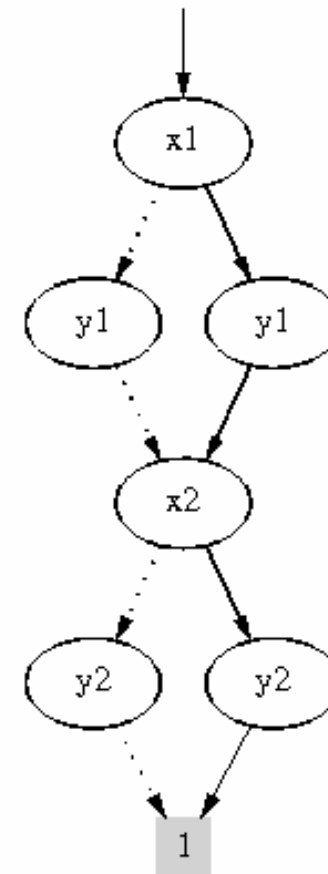
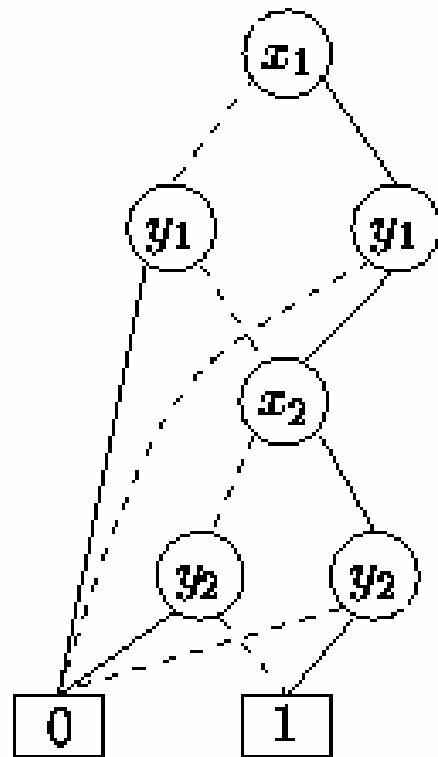
# Orderedness & Reducedness



$x < y$   $x < z$



# Reduced Ordered Binary Decision Diagrams



*Iben*  
*Edges to 0*  
*implicit*

# 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  $low(v), high(v) \in V$ .

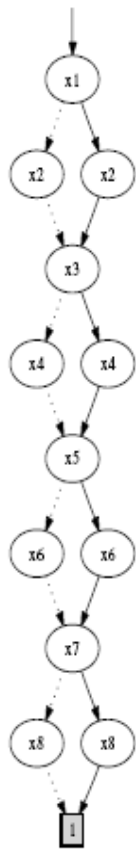
A BDD is *ordered* if on all paths from the root the variables respect a given total order.

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

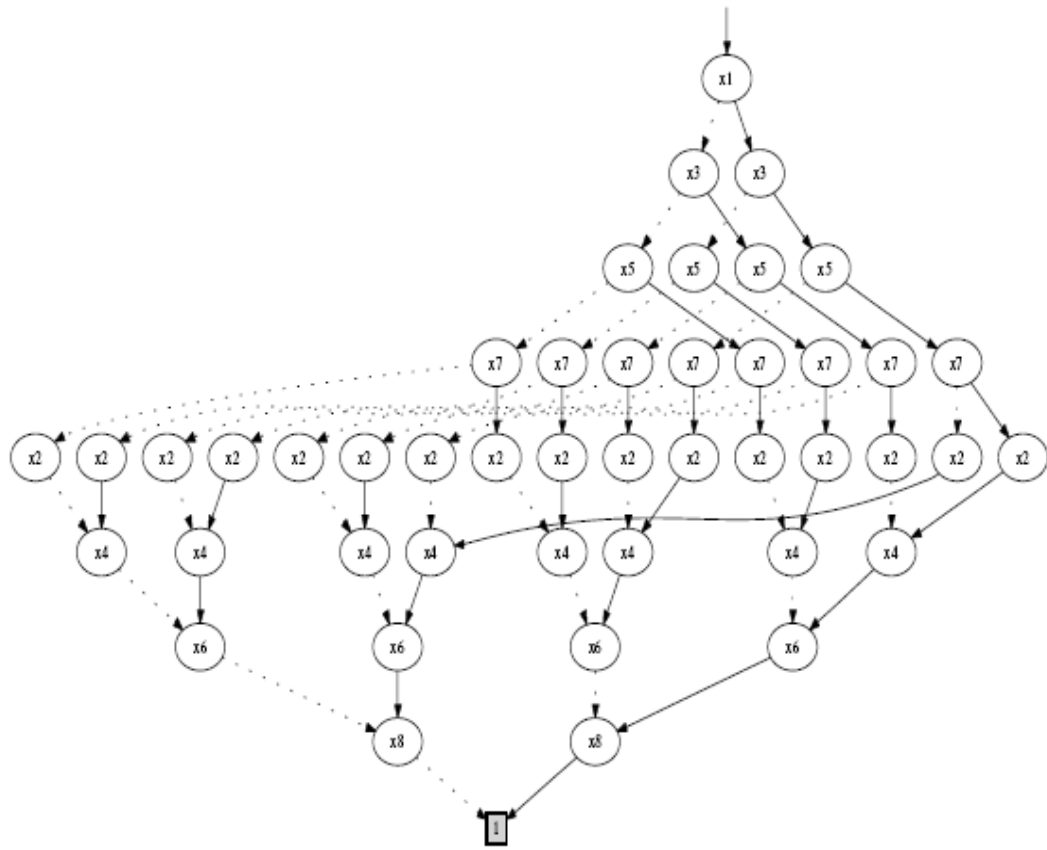
- 1)  $low(u) \neq high(u)$
- 2)  $low(u) = low(v), high(u) = high(v), var(u) = var(v)$   
implies  $u = v$

# Ordering **DOES** matter

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



$x_1 < x_2 < \dots < x_8$



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

# Canonicity of ROBDDs

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

**Lemma 1 (Canonicity lemma)** For any function  $f: \mathbb{B}^n \rightarrow \mathbb{B}$  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,  $b \neq \boxed{0}$

# Algorithms on ROBDDs

# Array Implementation

Assume  $x_1 < x_2 < x_3$ .

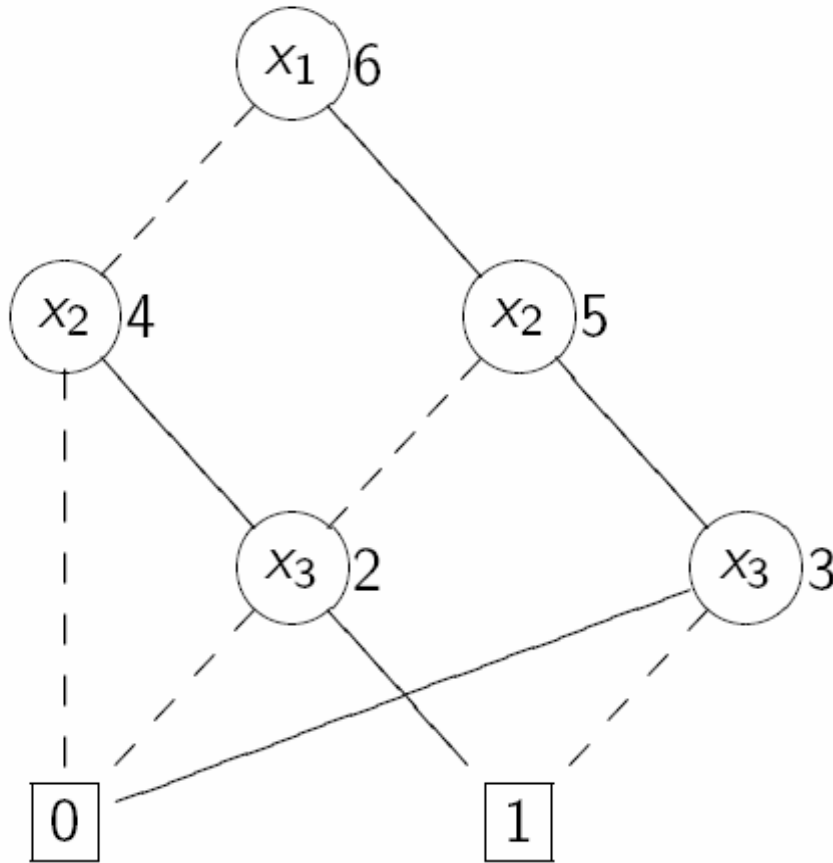


Table  $T$  :

$u \mapsto (var(u), low(u), high(u))$

$u$	$var$	$low$	$high$
0	4	-	-
1	4	-	-
2	3	0	1
3	3	1	0
4	2	0	2
5	2	2	3
6	1	4	5

Inverse table  $H$  :

$(var, low, high) \mapsto u$ .

Example:  $T(4) = (2, 0, 2)$ ,  $H(1, 4, 5) = 6$ , and  $H(3, 0, 2) = \text{undef}$ .



# MakeNode

$T : u \mapsto (var(u), low(u), high(u))$        $H : (var, low, high) \mapsto u$

**Makenode** ( $var, low, high$ ): Node =

**if**  $low = high$  **then**

**return**  $low$

**else**

$u := H(var, low, high)$

**if**  $u \neq undef$  **then**

**return**  $u$

**else**

        add a new node (row) to  $T$  with attributes ( $var, low, high$ )

**return**  $H(var, low, high)$

**end if**

**end if**

# Build

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

**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**

$low := \text{Build}(t[0/x_i], i + 1)$

$high := \text{Build}(t[1/x_i], i + 1)$

$var := i$

**return**  $\text{Makenode}(var, low, high)$

**end if**

**Complexity ??**

# Boolean Operations on BDDs

Let us assume that ROBDDs for boolean expressions  $t_1$  and  $t_2$  are already constructed.

How to construct ROBDD for

- $\neg t_1$
- $t_1 \wedge t_2$
- $t_1 \vee t_2$
- $t_1 \Rightarrow t_2$
- $t_1 \Leftrightarrow t_2$

with an emphasis on **efficiency**?

## Idea ( $x_1 < x_2 < x_3 < \dots < x_n$ )

- $x_i = x_j$

$$(x_i \rightarrow t_1, t_2) \wedge (x_i \rightarrow t'_1, t'_2)$$

$\equiv$

$$x_i \rightarrow (t_1 \wedge t'_1), (t_2 \wedge t'_2)$$

- $x_i < x_j$

$$(x_i \rightarrow t_1, t_2) \wedge (x_j \rightarrow t'_1, t'_2)$$

$\equiv$

$$x_i \rightarrow (t_1 \wedge (x_j \rightarrow t'_1, t'_2)), (t_2 \wedge (x_j \rightarrow t'_1, t'_2))$$

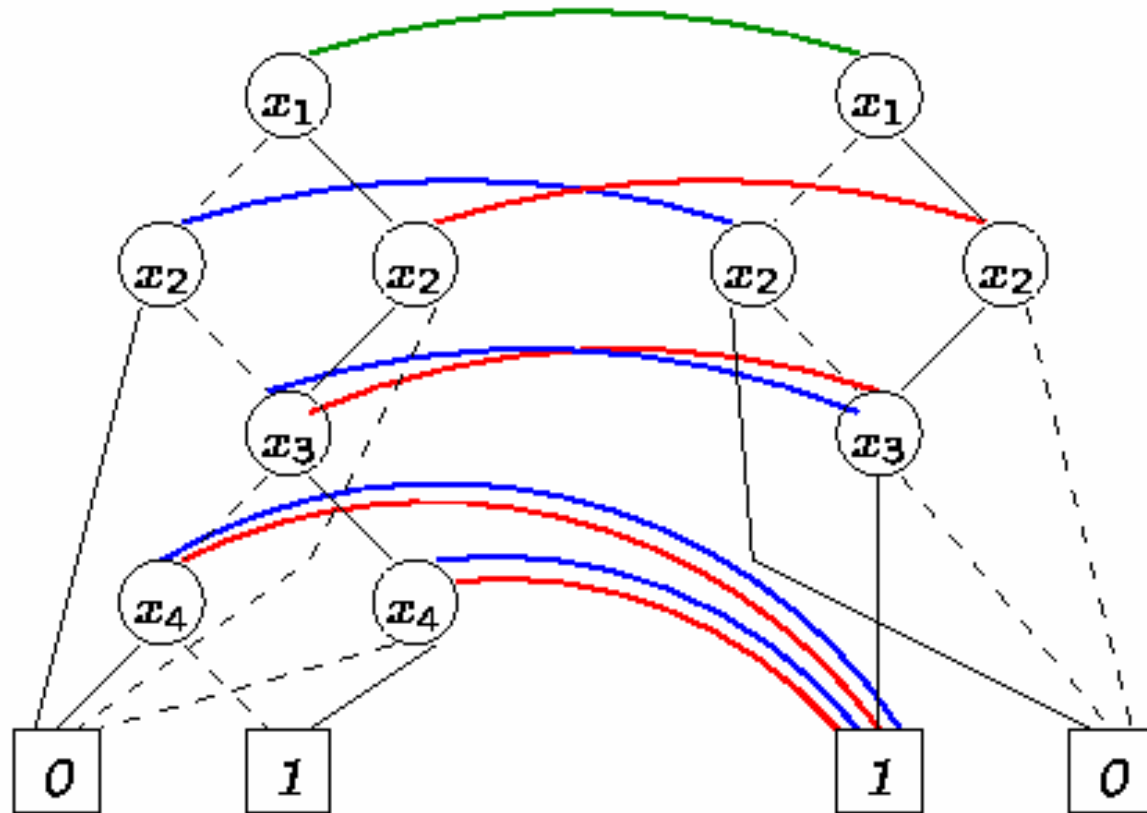
The same equivalences hold also for  $\vee$ ,  $\Rightarrow$  and  $\Leftrightarrow$ .

# 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
19:
20:
21: b.root ← app(b1.root, b2.root)
22: return b
```

**Complexity ??**

# 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
```

Complexity  $O(|b_1||b_2|)$

# 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  $t' \rightarrow t[1/x_i], t[0/x_i]$



# Application of ROBDDs

Constraint Solving & Analysis  
& **IBEN**

# Mia's Schedule 4th Grade

	<b>Mon</b>	<b>Tue</b>	<b>Wed</b>	<b>Thu</b>	<b>Fri</b>	<b>Sat</b>	<b>Sun</b>
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;

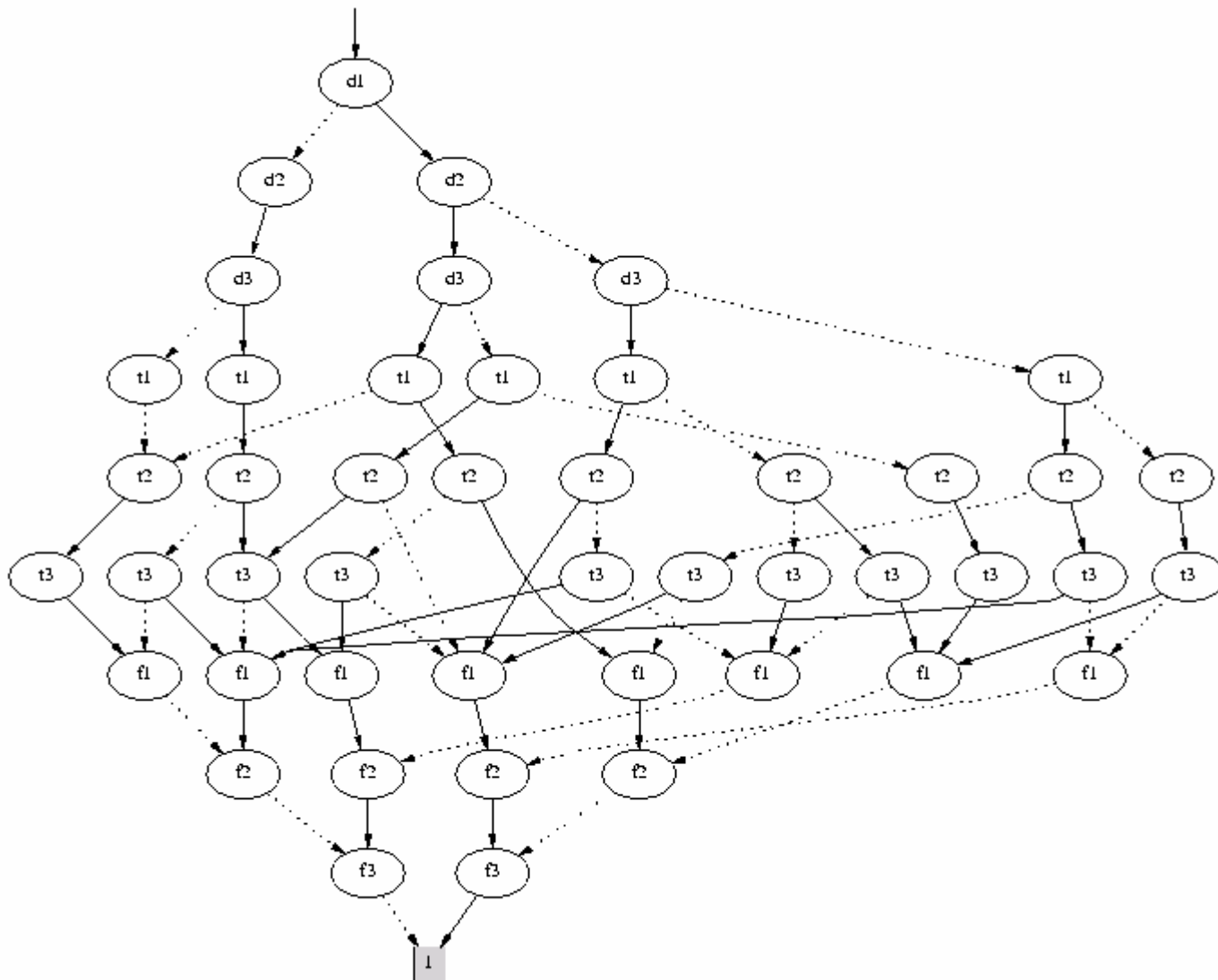
aftermiddag := ! formiddag;

```
--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 Schedule --

=====

```
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)));

