Tecniche di Specifica e di Verifica

Boolean Decision Diagrams I
(BDDs)

Outline

- **NuSMV**
- The state explosion problem.
- Techniques for overcoming this problem:
  - Compact representation of the state space.
    - **BDDs**.
  - **Abstractions (bisimulations)**
  - **Symmetries**.
  - **Partial Order Reductions**.

NuSMV

- **N**ew **S**ymbolic **M**odel **V**erifier.
- Developed at **CMU-IRST (Ed Clarke, Ken McMillan, Cimatti et al.)** as extension/reimplementation of **SMV**.
- **NuSMV** has its own input language (also called **SMV**!).

NuSMV

- You must prepare your verification problem in this language.
- An **NuSMV** program is a convenient way to describe a **Kripke structure**.
- You can insert the properties you want to verify in the program.
- Read the tutorial and on a need-to-know basis, the manual.

Parallel Composition

- $\mathbf{TS_1 = (S_1, S_1^0, \Sigma_1, R_1)}$ $\mathbf{R_1 \subseteq S_1 \times \Sigma_1 \times S_1}$
- $\mathbf{TS_2 = (S_2, S_2^0, \Sigma_2, R_2)}$ $\mathbf{R_2 \subseteq S_2 \times \Sigma_2 \times S_2}$

- $\mathbf{a} \in \Sigma_1$ and $\mathbf{a} \notin \Sigma_2$
  – An "***internal***" action of $\mathbf{TS_1}$.
- $\mathbf{a} \in \Sigma_1 \cap \Sigma_2$
  – A ***common*** (***synchronizing***) action of $\mathbf{TS_1}$ and $\mathbf{TS_2}$.

Parallel Composition

- $TS_1 = (S_1, S_1^0, \Sigma_1, R_1)$ $R_1 \subseteq S_1 \times \Sigma_1 \times S_1$
- $TS_2 = (S_2, S_2^0, \Sigma_2, R_2)$ $R_2 \subseteq S_2 \times \Sigma_2 \times S_2$
- $TS = (TS_1 \parallel TS_2) = (S, S^0, \Sigma, R)$.
  - $S = S_1 \times S_2$
  - $S^0 = S_1^0 \times S_2^0$
  - $\Sigma = \Sigma_1 \cup \Sigma_2$

Parallel Composition

- $\mathbf{TS_1 = (S_1, S_1^0, \Sigma_1, R_1)}$ $\quad \mathbf{R_1 \subseteq S_1 \times \Sigma_1 \times S_1}$
- $\mathbf{TS_2 = (S_2, S_2^0, \Sigma_2, R_2)}$ $\quad \mathbf{R_2 \subseteq S_2 \times \Sigma_2 \times S_2}$
- $\mathbf{TS = (TS_1 \parallel TS_2) = (S, S^0, \Sigma, R)}$.

  - $\mathbf{R \subseteq S \times \Sigma \times S}$
    - $\mathbf{S = S_1 \times S_2}$.
  - $\mathbf{R((s1, s2), a, (t1, t2))}$?
  - if $\mathbf{a} \in \Sigma_1$ and $\mathbf{a} \notin \Sigma_2$
  - then $\mathbf{R_1(s1, a, t1)}$ and $\mathbf{s2 = t2}$.

Parallel Composition

- $\mathbf{TS_1 = (S_1, S_1^0, \Sigma_1, R_1)}$ $\mathbf{R_1 \subseteq S_1 \times \Sigma_1 \times S_1}$
- $\mathbf{TS_2 = (S_2, S_2^0, \Sigma_2, R_2)}$ $\mathbf{R_2 \subseteq S_2 \times \Sigma_2 \times S_2}$
- $\mathbf{TS = (TS_1 \parallel TS_2) = (S, S^0, \Sigma, R)}$.

– $\mathbf{R \subseteq S \times \Sigma \times S}$
  - $\mathbf{S = S_1 \times S_2}$.

– $\mathbf{R((s1, s2), a, (t1, t2))}$?

– if $\mathbf{a} \in \Sigma_2$ and $\mathbf{a} \notin \Sigma_1$

– then $\mathbf{R_2(s2, a, t2)}$ and $\mathbf{s1 = t1}$.

Parallel Composition

- $\mathbf{TS_1 = (S_1, S_1^0, \Sigma_1, R_1)}$ $\mathbf{R_1 \subseteq S_1 \times \Sigma_1 \times S_1}$
- $\mathbf{TS_2 = (S_2, S_2^0, \Sigma_2, R_2)}$ $\mathbf{R_2 \subseteq S_2 \times \Sigma_2 \times S_2}$
- $\mathbf{TS = (TS_1 \parallel TS_2) = (S, S^0, \Sigma, R)}$.

  – $\mathbf{R \subseteq S \times \Sigma \times S}$
    - $\mathbf{S = S_1 \times S_2}$.

  – **R((s1, s2), a, (t1, t2))** ?

  – if $\mathbf{a \in \Sigma_1}$ **and** $\mathbf{a \in \Sigma_2}$

  – then $\mathbf{R_1(s1, a, t1)}$ **and** $\mathbf{R_2(s2, a, t2)}$

Parallel Composition

- **$TS = (TS_1 \parallel TS_2) \parallel TS_3$**
- **$TS = TS_1 \parallel (TS_2 \parallel TS_3)$**
- **$TS = TS_1 \parallel TS_2 \parallel TS_3$**

Parallel Composition

- $\mathbf{TS = TS_1 \parallel TS_2 \ldots \parallel TS_n}$
- $\mathbf{Size(TS_i) \approx |S_i| = k_i \geqslant 2}$
- **Description of** $\boldsymbol{TS \approx k_1 + k_2 \ldots + k_n}$
- $\mathbf{Size(TS) = k_1 \times k_2 \ldots \times k_n}$

  $\mathbf{\geqslant 2^n\ !}$
- Size of **TS** is ***exponential*** in **n** (the ***number*** of ***components***).
- ***State space explosion problem***.

How to circumvent state space explosion?

- Use succinct representations of the state space.
  - **Boolean Decision Diagrams**.
- Reduce **TS** to **TS'** such than:
  - **TS** has the required property *iff* **TS'** has the required property.
    - Symmetries
    - **Abstractions (bisimulations)**
    - Partial order reductions.

Symbolic Model checking

- **$K = (S, S_0, R, AP, V)$**
- ψ a **CTL** formula
- To check whether:
  - **$K, s \models \psi$**
- We need to
  - compute $|[\psi]| = \mathbf{states}(\psi) = \{s \mid K, s \models \psi\}$.
  - then check whether $s \in |[\psi]|$.

Symbolic Model checking

- **K = (S, S_0, R, AP, V)**
- ψ a **CTL** formula

- **S' ⊆ S** can be represented as a ***boolean function***.
- **R** can be represented as a ***boolean function***.
- |[ψ]| can then be represented as a ***boolean function***.

- ***Boolean functions*** represent the ***characteristic functions*** of the given ***sets of states***.

BDDs

- Boolean functions can be (often) ***succinctly represented*** as ***boolean decision diagrams***.
- **BDDs** are easy to manipulate.
- ***Not all boolean functions have a succinct representation***.
- ***Use BDDs to represent and manipulate the boolean functions associated with the model checking process.***

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = $\{0,1\} \times \ldots . \times \{0,1\}$.**
  - **Range = $\{0, 1\}$**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = $\{0,1\} \times \ldots . \times \{0,1\}$.**
  - **Range = $\{0, 1\}$**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?
  - Answer : $\mathbf{2^{2^3} = 2^8}$ **!**

Truth Tables

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$\mathbf{g} : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \to \{0, 1\}$$

Boolean Expressions

- Given a set of ***Boolean variables x,y,…*** and the constants **1** (true) and **0** (false):

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

- The semantics of ***Boolean Expressions*** is defined by means of ***truth tables*** as usual.
- Given an ordering of Boolean variables, ***Boolean expressions*** can be used to express ***Boolean functions***.

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- **$x \wedge y$** represents the function:
  - **$f : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$**
    - **$f(0, 0) =$**
    - **$f(0, 1) =$**
    - **$f(1, 0) =$**
    - **$f(1, 1) =$**

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- **$x \wedge y$** represents the function:
  - **$f : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$**
    - **$f(0, 0) = 0$**
    - **$f(0, 1) = 0$**
    - **$f(1, 0) = 0$**
    - **$f(1, 1) = 1$**

Boolean functions and expressions

x	**y**	**z**	**g**
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

$$\mathbf{g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \to \{0, 1\}}$$

$$\mathbf{g = ((x \Leftrightarrow y) \land z) \lor ((x \Leftrightarrow \neg y) \land \neg z)}$$

Boolean expressions and functions

x	y	z	g
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

$$g = (x \land y \land \neg z) \lor (x \land \neg y \land z) \lor (\neg x \land y)$$

Boolean expressions and functions

x	y	z	g
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$g = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (\neg x \wedge y)$$

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

Three Representations

- ***Boolean functions***
- ***Truth tables***
- ***Propositional formulas***.
- Three ***equivalent*** representations.
- Here is a ***fourth one***!

Boolean Decision Tree

- A ***boolean function*** is represented as a (***binary***) ***tree***.
- Each ***internal node*** is labeled with a (boolean) ***variable***.
- Each ***internal node*** has a ***positive*** (***full line***) and a ***negative*** (***dotted line***) ***successor***.
- The ***terminal nodes*** are labeled with **0** or **1**.

Boolean Decision Diagrams

- A **compact way** of representing boolean functions.
- Can be used in **CTL** model checking.
  - Represent a subset of states as a boolean function.
  - Represent the transition relation as a boolean function.
  - Reduce **EX(ψ)**, **EU(ψ$_1$, ψ$_2$)** and **EG(ψ)** to manipulating boolean functions and checking for **boolean function equality**.
- Go from **NuSMV** (program) representation ***directly*** to its **BDD** representation!

If-Then-Else operator

$(x \rightarrow s_1 , s_0) \equiv (x \wedge s_1) \vee (\neg x \wedge s_0)$

x	y	z	$x \rightarrow y , z$
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

x	y	$x \rightarrow y , 0$	$x \wedge y$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

x	y	$x \rightarrow 1 , y$	$x \vee y$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

If-Then-Else representation

Let $x \in AP$, then

- $x \equiv x \to 1, 0$
- $\neg\varphi \equiv \varphi \to 0, 1$
- $\varphi_1 \wedge \varphi_2 \equiv \varphi_1 \to \varphi_2, 0$
- $\varphi_1 \vee \varphi_2 \equiv \varphi_1 \to 1, \varphi_2$

***Theorem:* Every boolean formula can be written in *If-Then-Else representaton*.**

Assume $\varphi_1 \equiv x \to \psi_1, \psi_2$ then

$$\varphi_1 \to \varphi_2, \varphi_3 \equiv (x \to \psi_1, \psi_2) \to \varphi_2, \varphi_3 \equiv$$

$$\equiv x \to (\psi_1 \to \varphi_2, \varphi_3), (\psi_2 \to \varphi_2, \varphi_3)$$

If-Then-Else normal form

***ITE normal form*: a boolean expression is written in *ITE normal form* if it only contains constants 0 and 1, If-Then-Else is the only operator occurring in the expression and tests are only performed on variables.**

Boolean decision trees.

If-Then-Else normal form

$$\mathbf{x \wedge y = x \rightarrow y , 0}$$

Shannon Expansion:

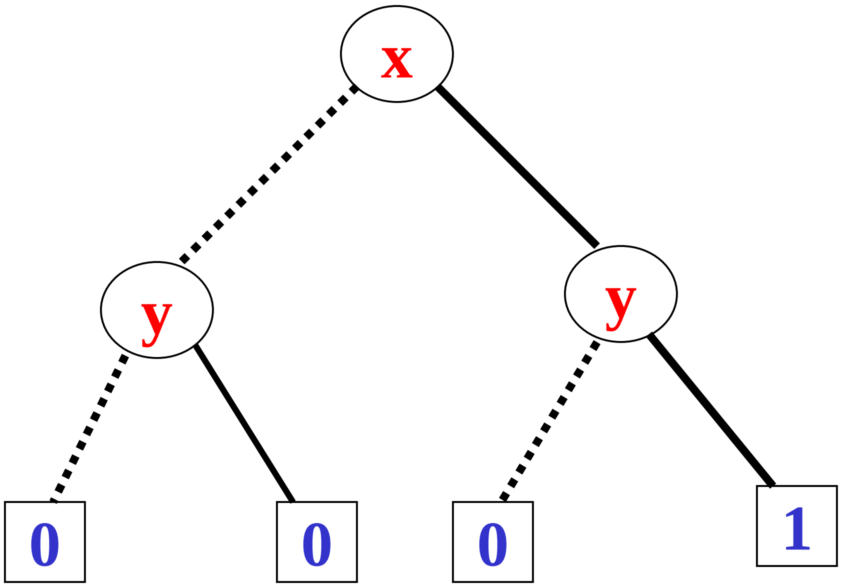
$$\boxed{\mathbf{f = (x \wedge f_{[1/x]}) \vee (\neg x \wedge f_{[0/x]})}}$$

$$\mathbf{f = x \rightarrow f_{[1/x]} , f_{[0/x]}}$$

where

$$\mathbf{f_{[a/x]}(\ldots,x,\ldots) = f(\ldots,a,\ldots)}$$

for $\mathbf{a = 0,1}$.



$\mathbf{x \wedge y}$

If-Then-Else normal form

***ITE normal form*: a boolean expression is written in *ITE normal form* if it only contains constants 0 and 1, If-Then-Else is the only operator occurring in the expression and tests are only performed on variables.**

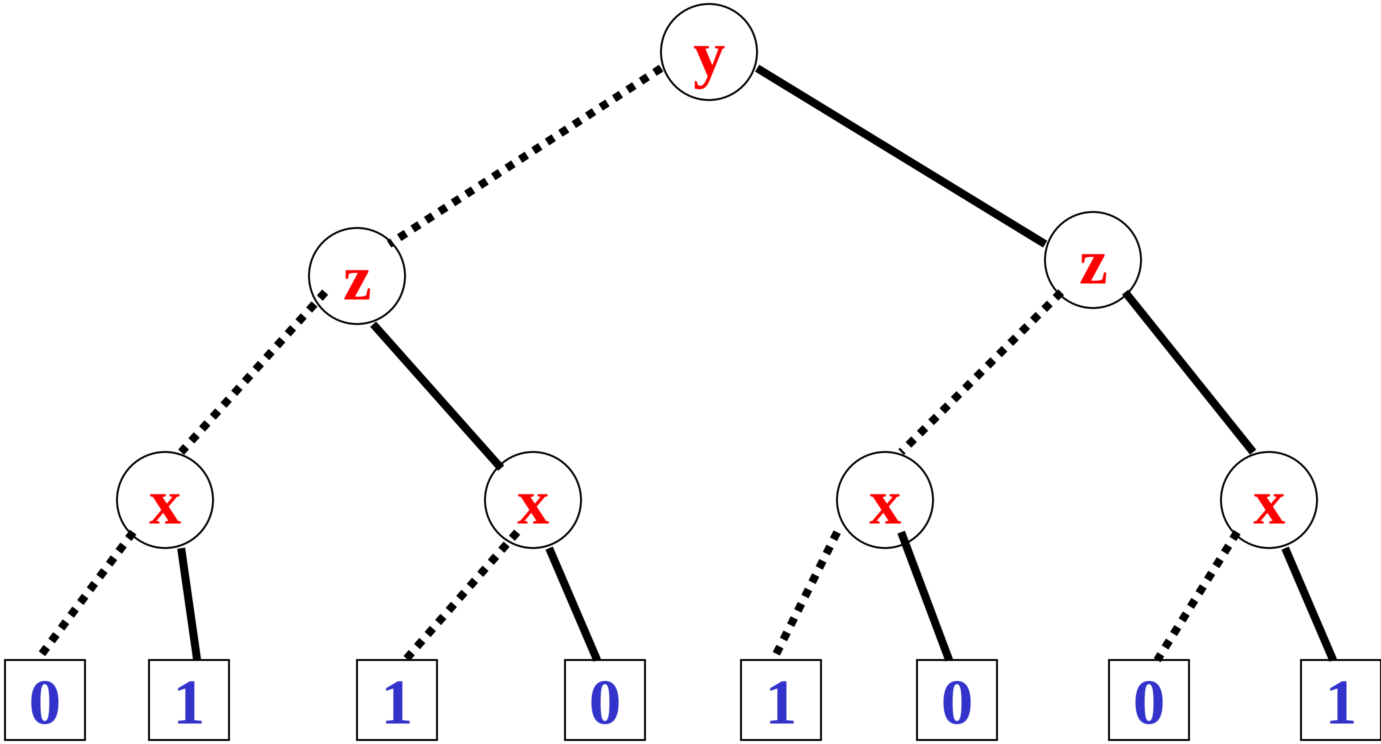
***Theorem:* Every boolean formula can be written in *ITE normal form*.**

Proof: by trivial induction on the structure of boolean formulae.

Boolean Decision Tree

- A ***boolean function*** is represented as a (***binary***) ***tree***.
- Each ***node*** is ***labeled*** with a (boolean) ***variable***.
- Each ***node*** has a ***positive*** (***full line***) and a ***negative*** (***dotted line***) ***successor***.
- The ***terminal nodes*** are labeled with **0** or **1**.

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

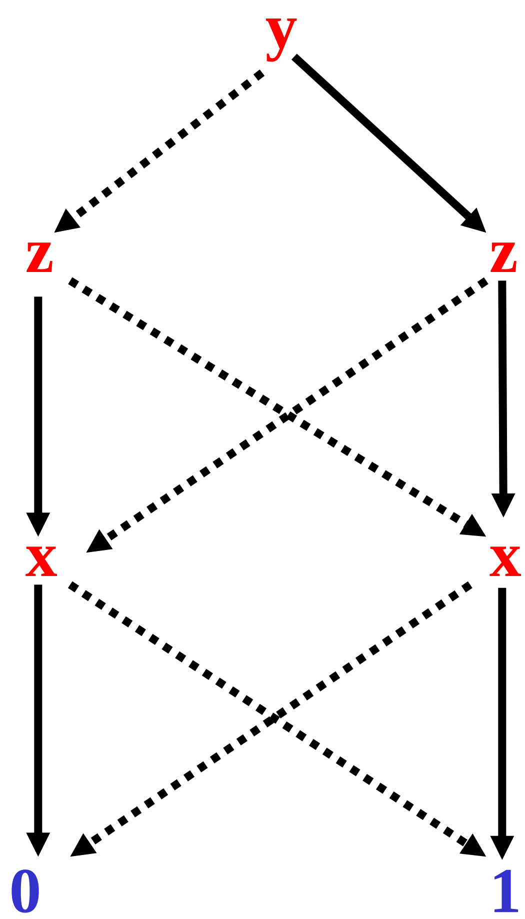


$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

BDDs

A **BDD** is ***finite rooted directed acyclic graph*** in which:

- There is a ***unique initial node*** (the ***root***)
- Each ***terminal node*** is labeled with a **0** or **1**.
- Each ***non-terminal*** (internal) node ***v*** has three attribute:
  - ***var(v)***, and
  - exactly ***two successors low(v)*** and ***high(v)***: one labeled **0** (***dotted edge***, ***low(v)***) and the other labeled **1** (***solid edge***, ***high(v)***).



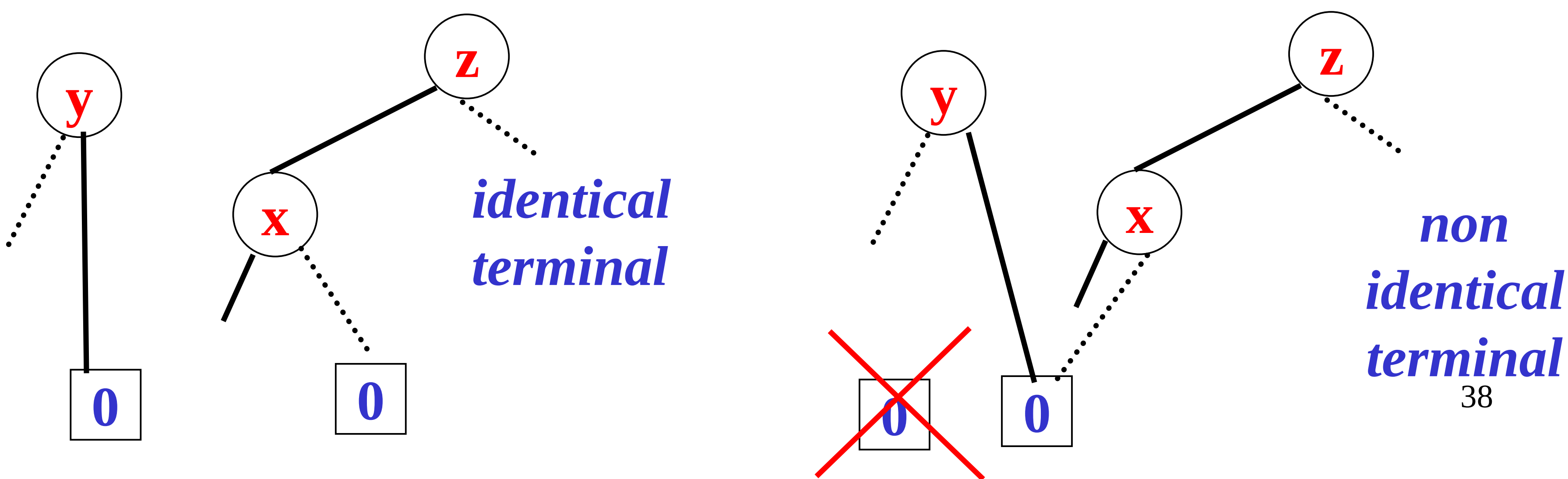
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Reduction Rules

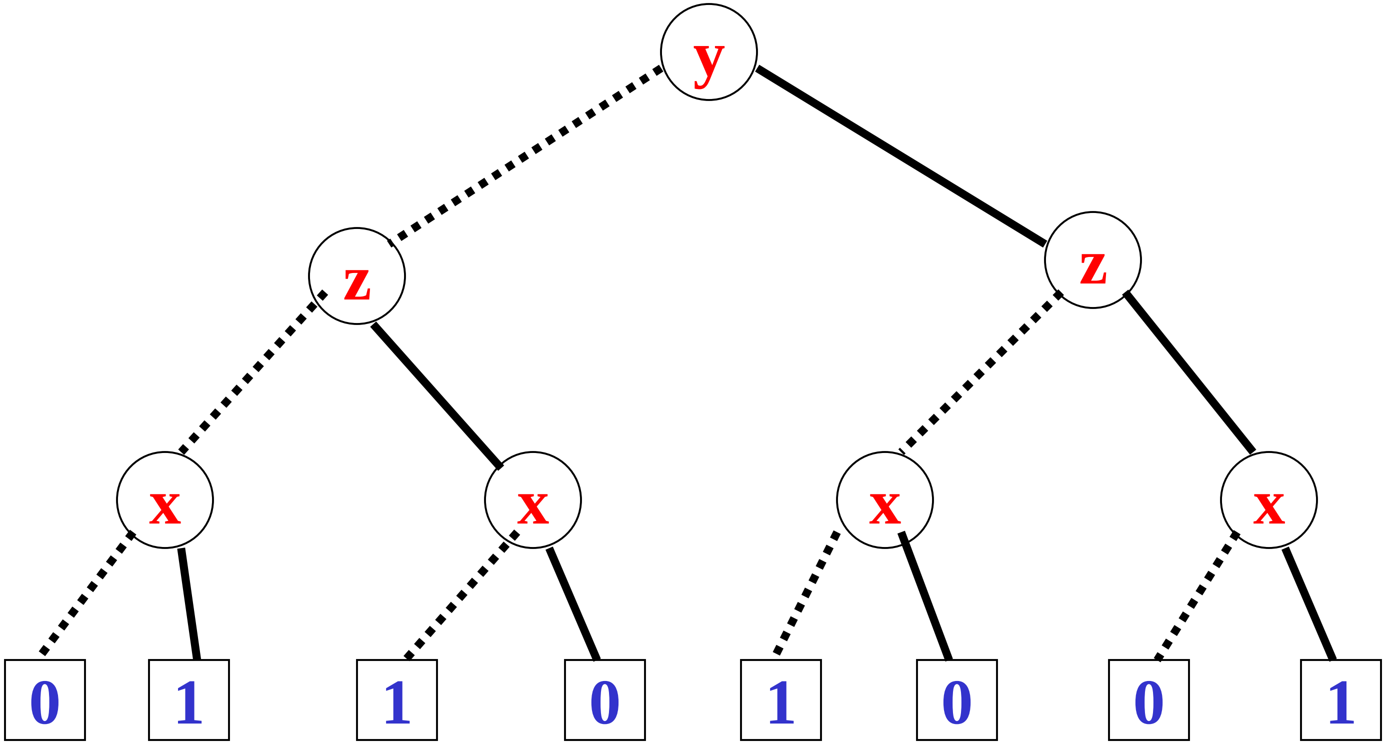
- Three reduction rules:
  - **Share identical terminal nodes. (R1)**
  - **Remove redundant tests (R2)**
  - **Share identical non-terminal nodes. (R3)**

Reduction Rules

- Three reduction rules:
  - **Share identical terminal nodes. (R1)**
- If a **BDD** contains ***two terminal nodes* m** and **n** both ***labeled* 0** then, ***remove* n** and ***direct all incoming edges at* n *to* m.**
- **Similarly for *two terminal nodes labeled* 1.**

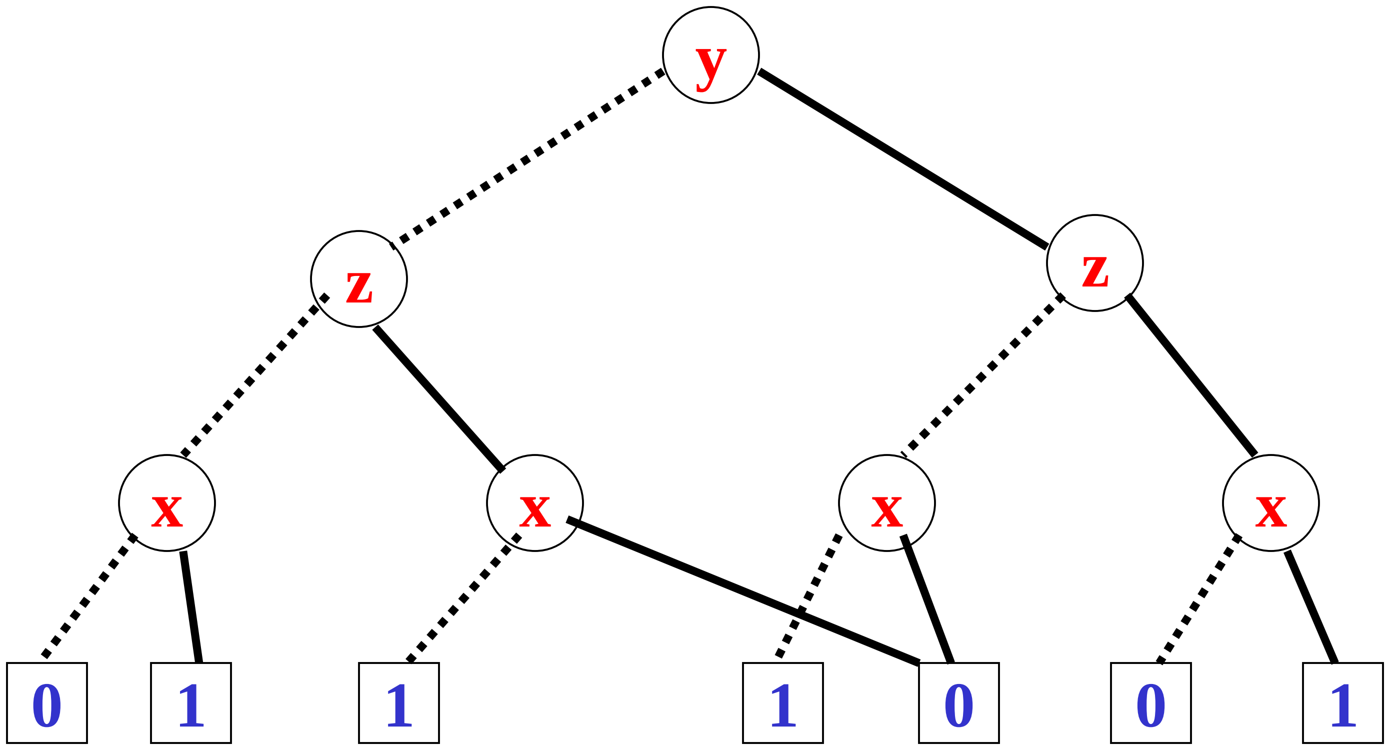


Share identical terminal nodes. (R1)



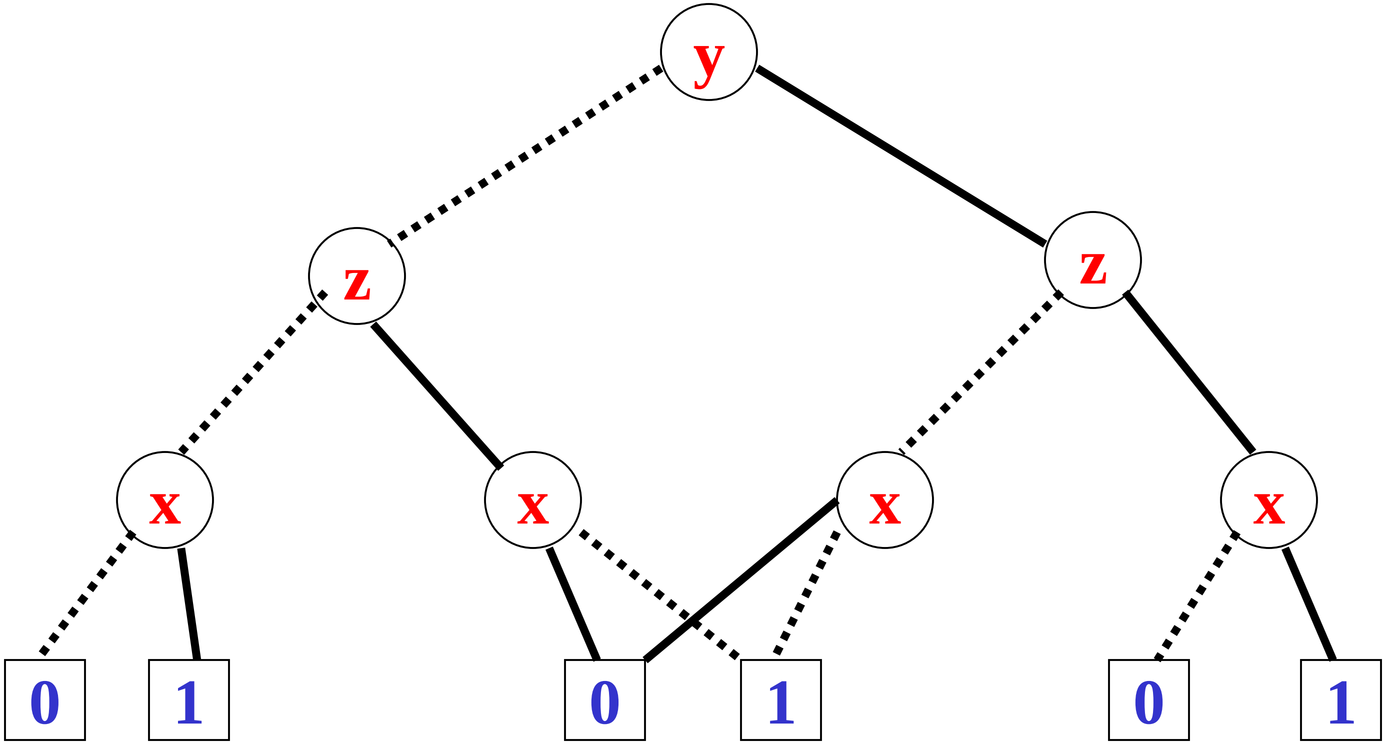
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Share identical terminal nodes. (R1)



$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

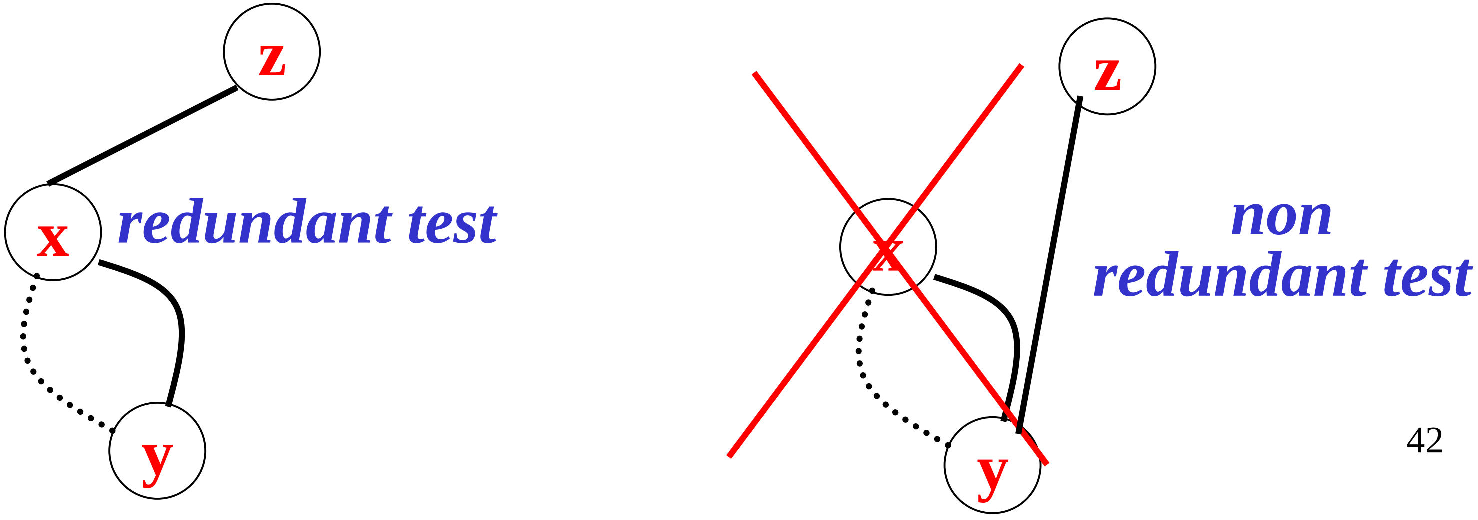
Share identical terminal nodes. (R1)



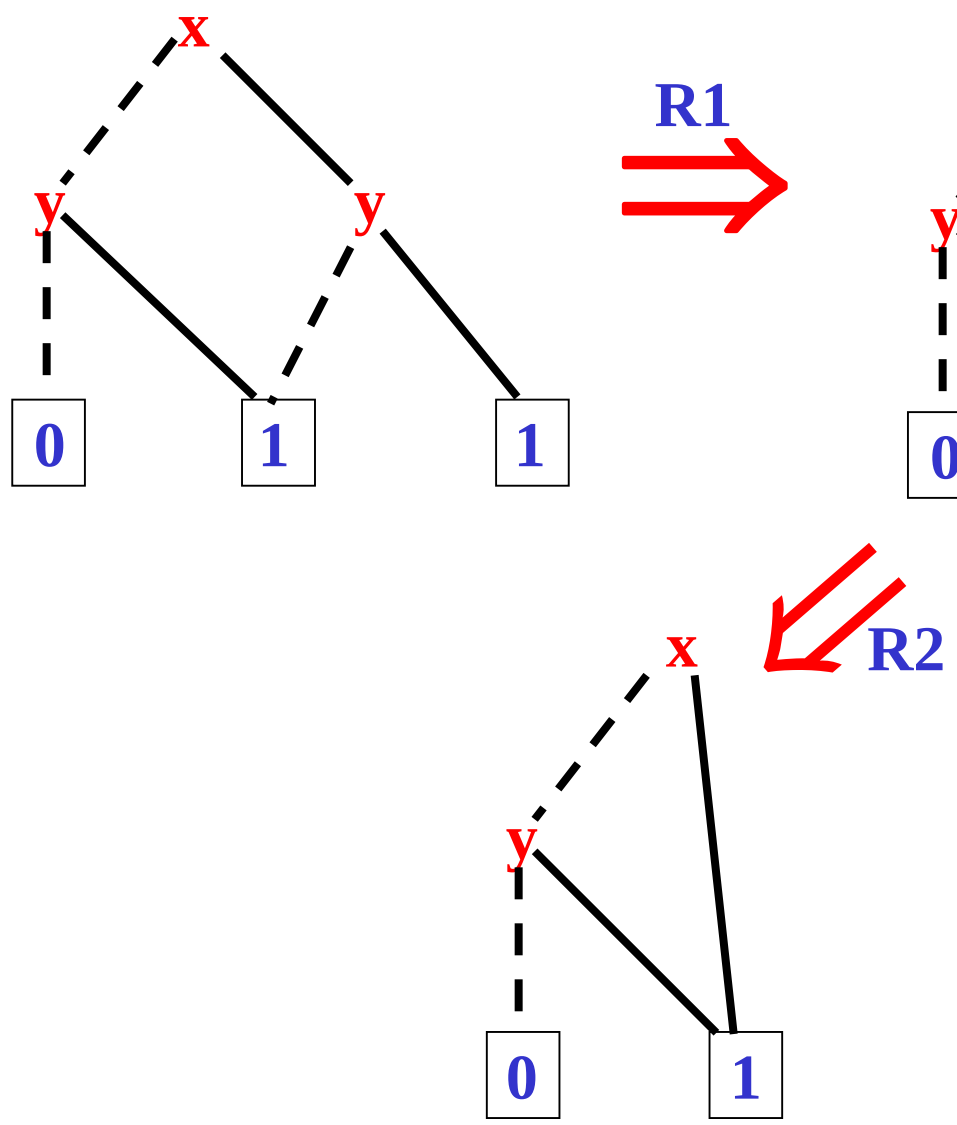
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Reduction Rules

- Three reduction rules:
  - Share identical terminal nodes. (**R1**)
  - **Remove redundant tests (R2)**
- If both ***successors of node* m *lead to the same node* n** then ***remove* m** and ***direct all incoming edges of* m *to* n**.

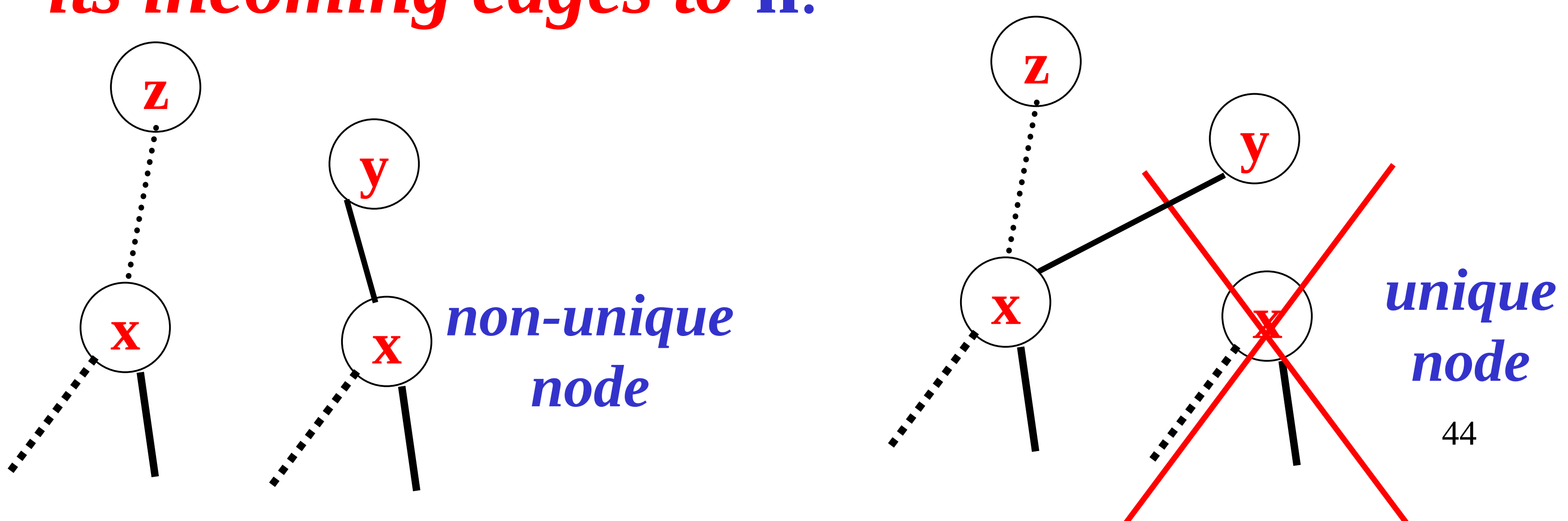


Remove redundant tests (R2)

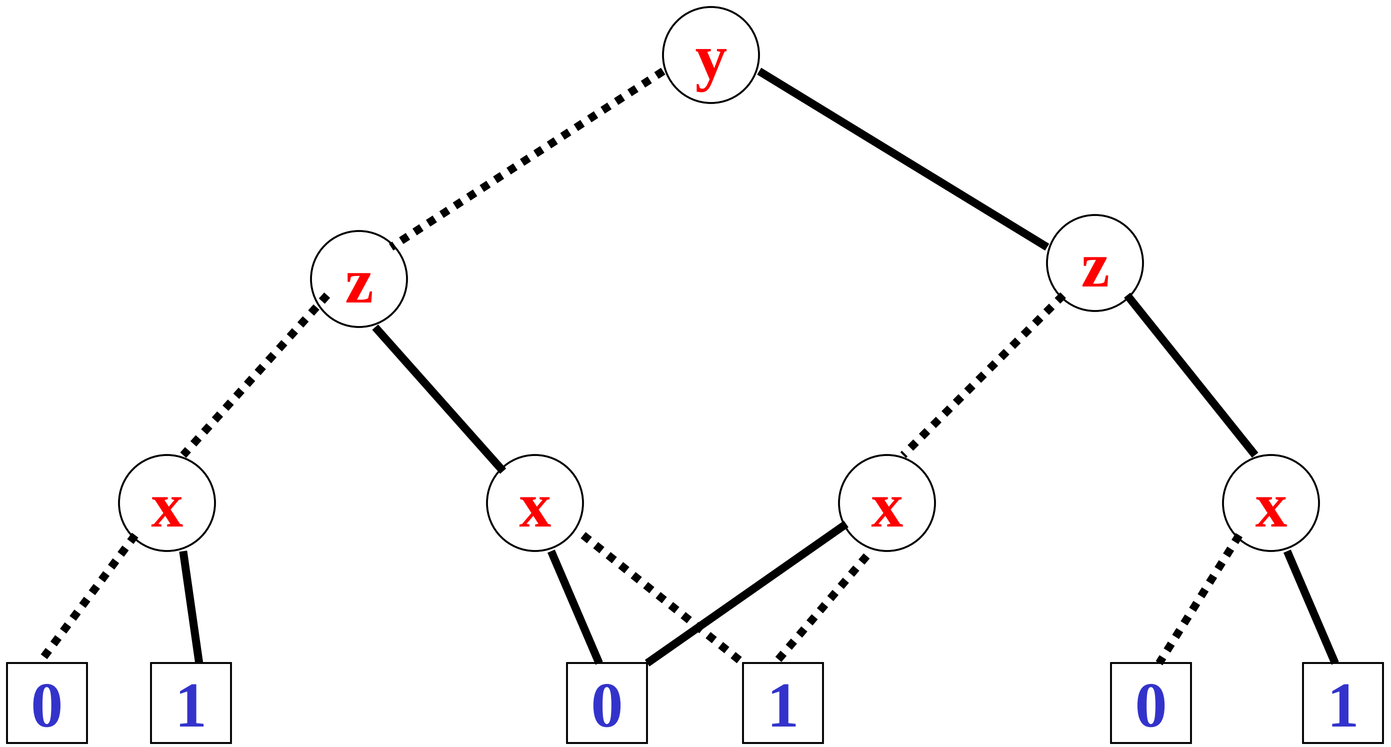


Reduction Rules

- Three reduction rules:
  - Share identical terminal nodes. (**R1**)
  - Remove redundant tests (**R2**)
  - **Share identical non-terminal nodes**. **(R3)**
- If the ***sub-BDDs rooted at the nodes*** **m** and **n** are "***identical***" then ***remove*** **m** and ***direct all its incoming edges to*** **n**.

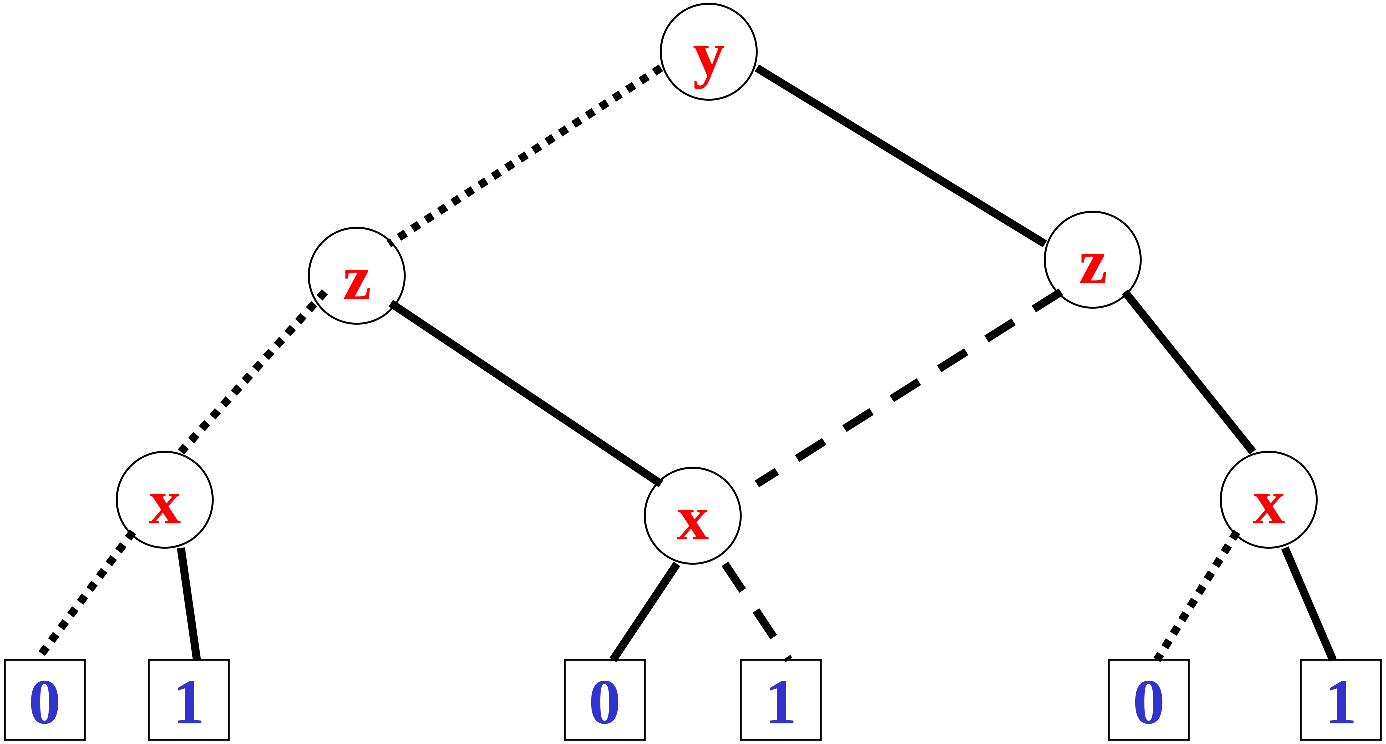


Share identical non-terminal nodes. (R3)



$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Share identical non-terminal nodes. (R3)



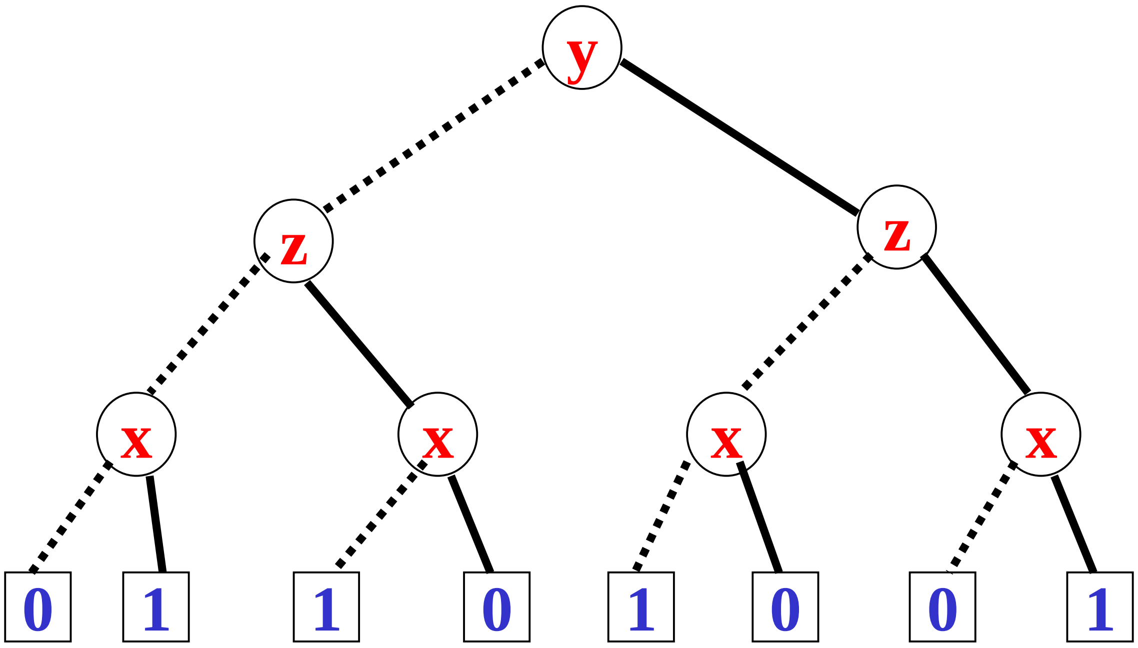
$$g = (y \land (x \Leftrightarrow z)) \lor (\neg y \land (x \Leftrightarrow \neg z))$$

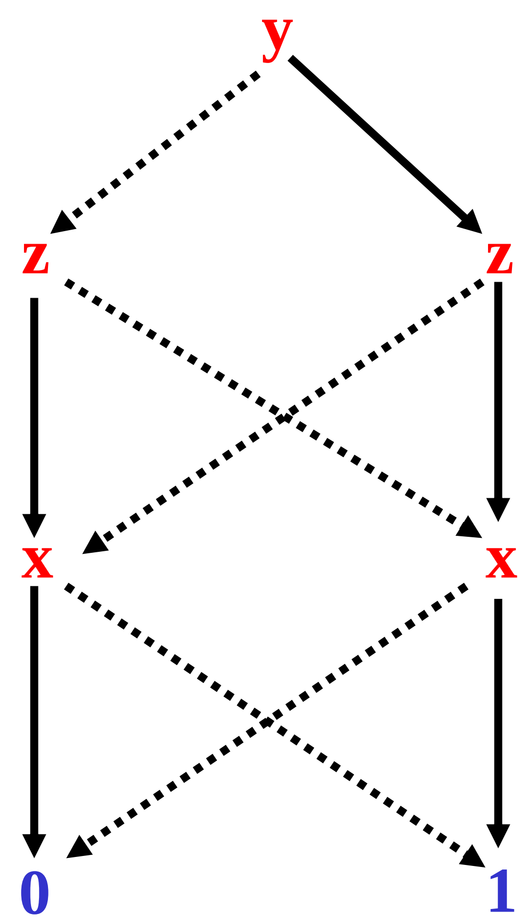
Reduced BDDs

- A **BDD** is ***reduced*** ***iff*** none of the three reduction rules can be applied to it.
- Start from the bottom layer (terminal nodes).
- ***Apply*** the ***rules*** repeatedly ***to level*** **i**. And then ***move to level*** **i-1** (in this way checking for applicability of R3 only needs testing whether **var(m)=var(n)**, **low(m)=low(n)** and **high(m)=high(n)**).
- Stop when the root node has been treated.
- This can be done efficiently.

Binary Decision Tree for

Reduced BDD





$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

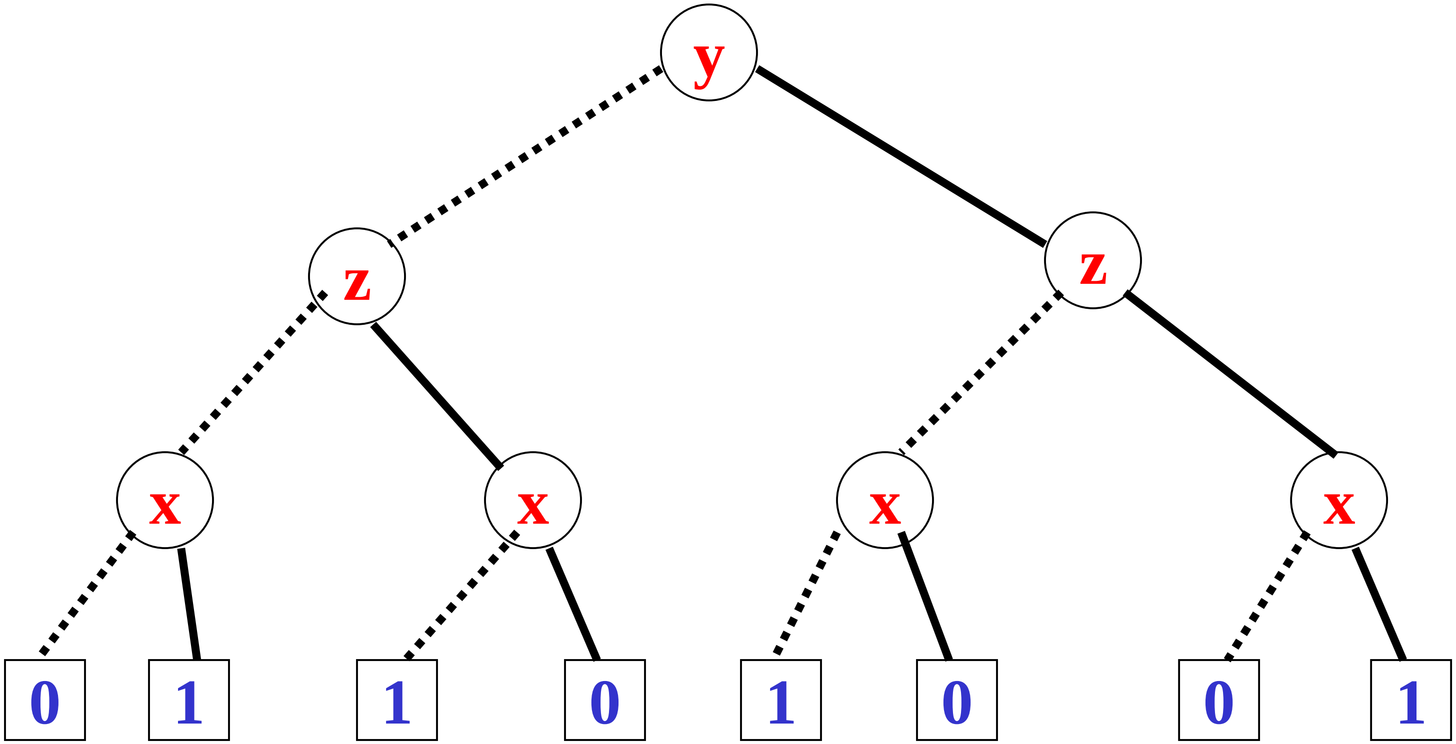
Ordered BDDs

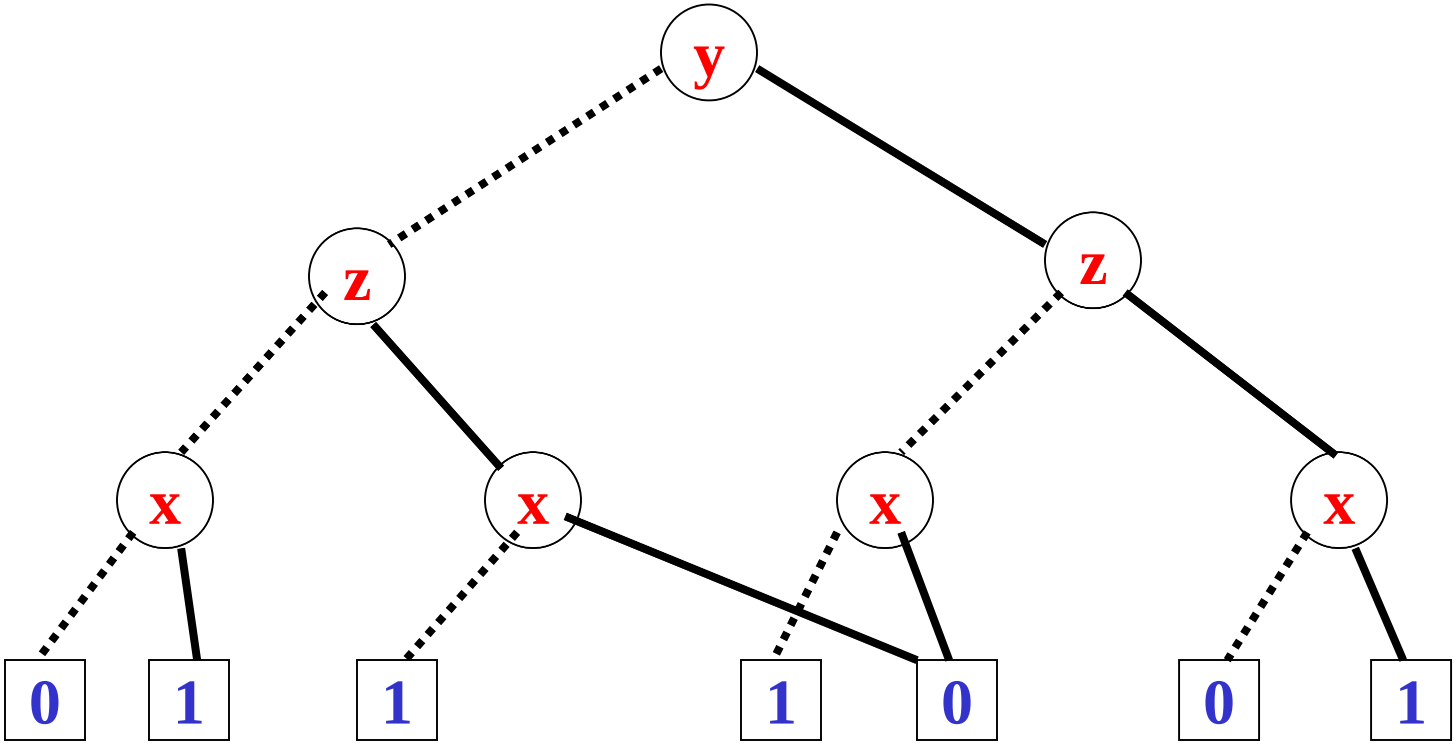
- $\{x_1, x_2, \ldots, x_n\}$
  - An indexed (ordered) set of boolean variables.
  - $x_1 < x_2 \ldots\ldots < x_n$
- **G** is an **ordered BDD** w.r.t. the above ***variable ordering iff***:
  - Each variable that appears in **G** is in the above set. (but the converse may not be true).
  - If $i < j$ and x_i and x_j appear on a path then **x_i appears before x_j**.

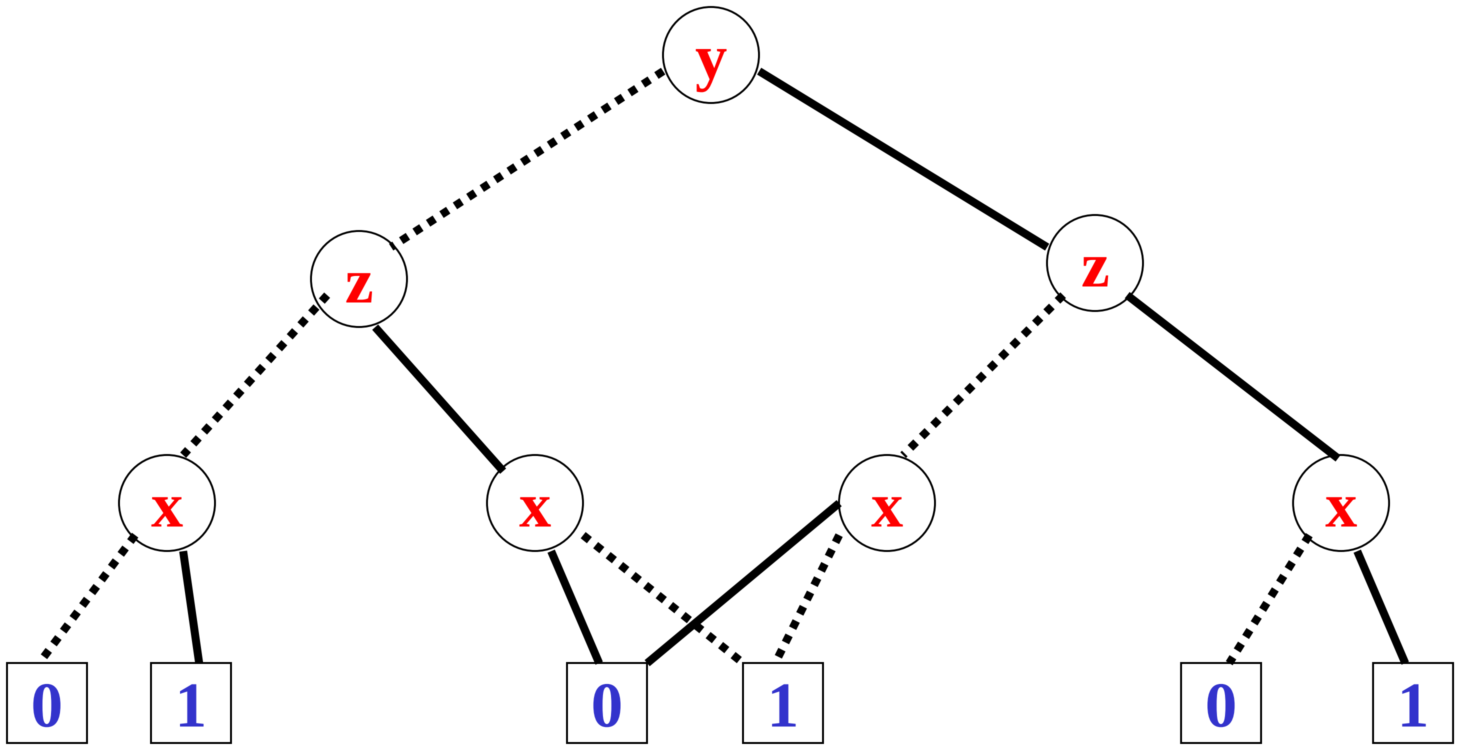
Ordered BDDS

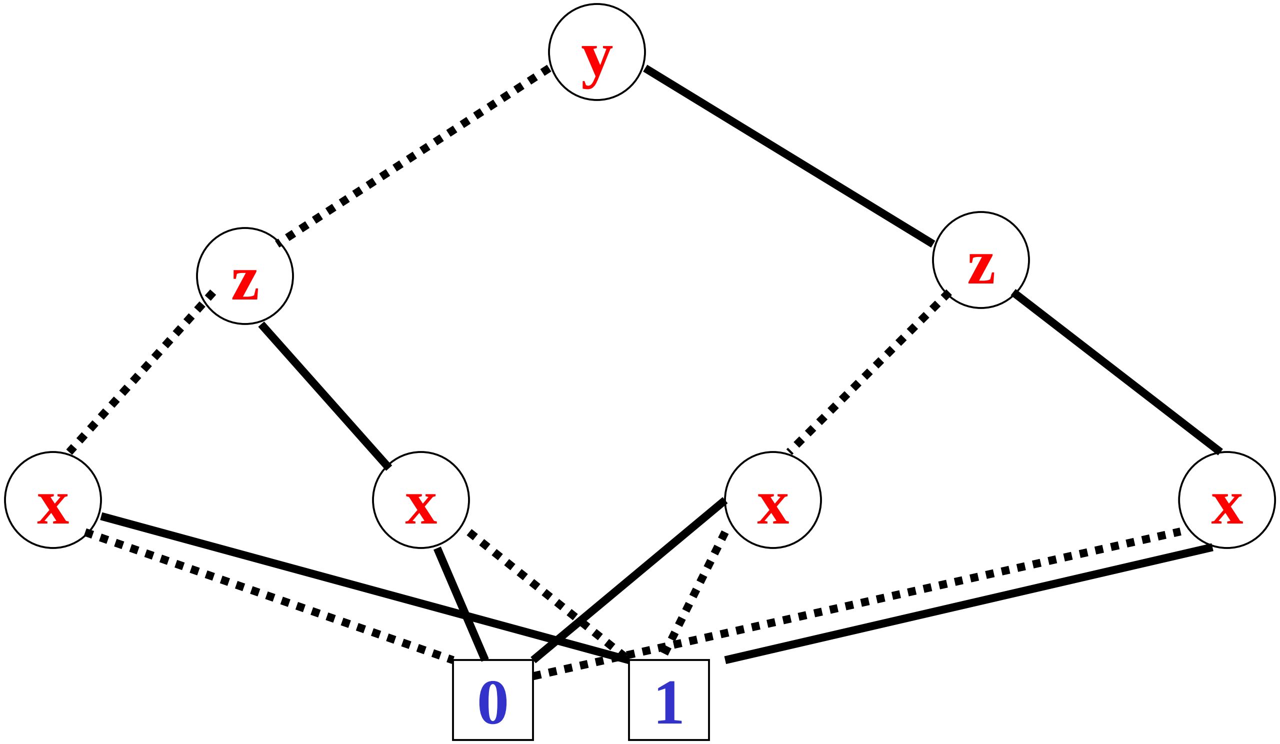
- Fundamental Fact:
  - For a fixed variable ordering, each boolean function has ***exactly one*** reduced Ordered BDD!
  - Reduced OBDDs are ***canonical objects***.
  - To test if ***f*** and ***g*** are equal, we just have to check if **their** reduced **OBDD**s are **identical**.
  - This will be crucial for model checking!

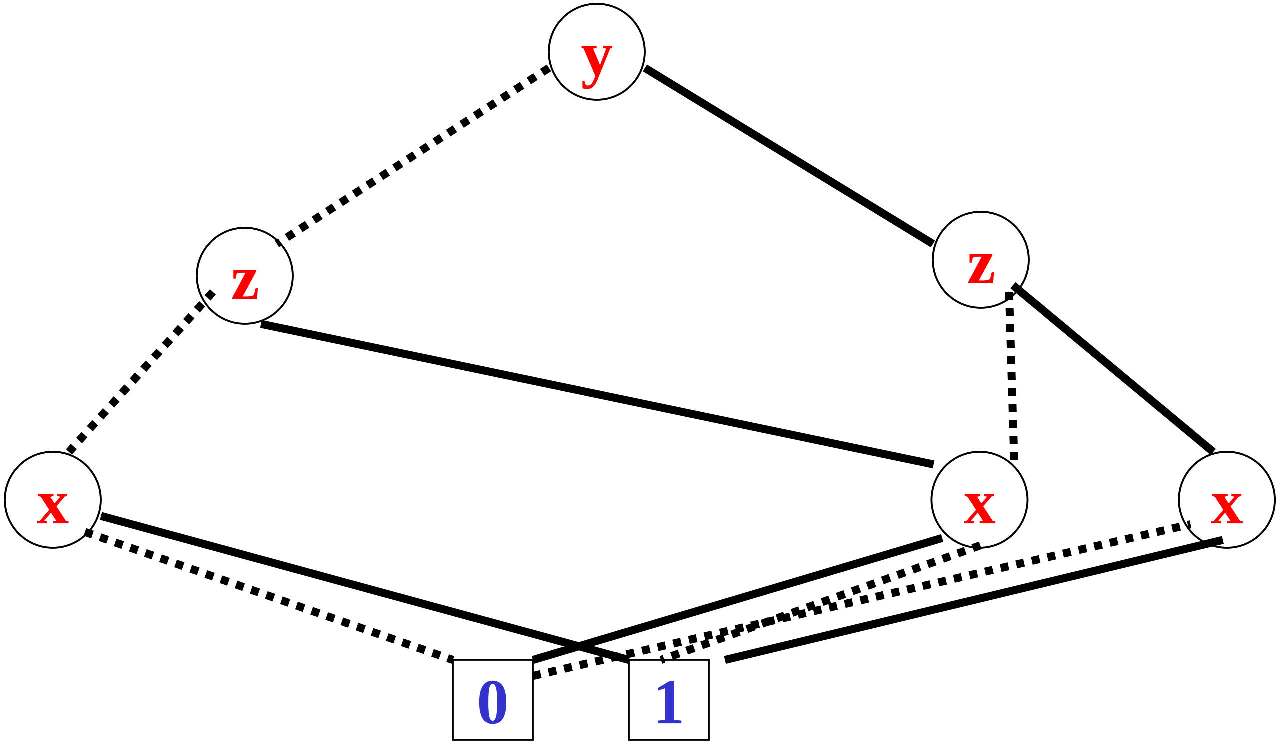
y < z < x

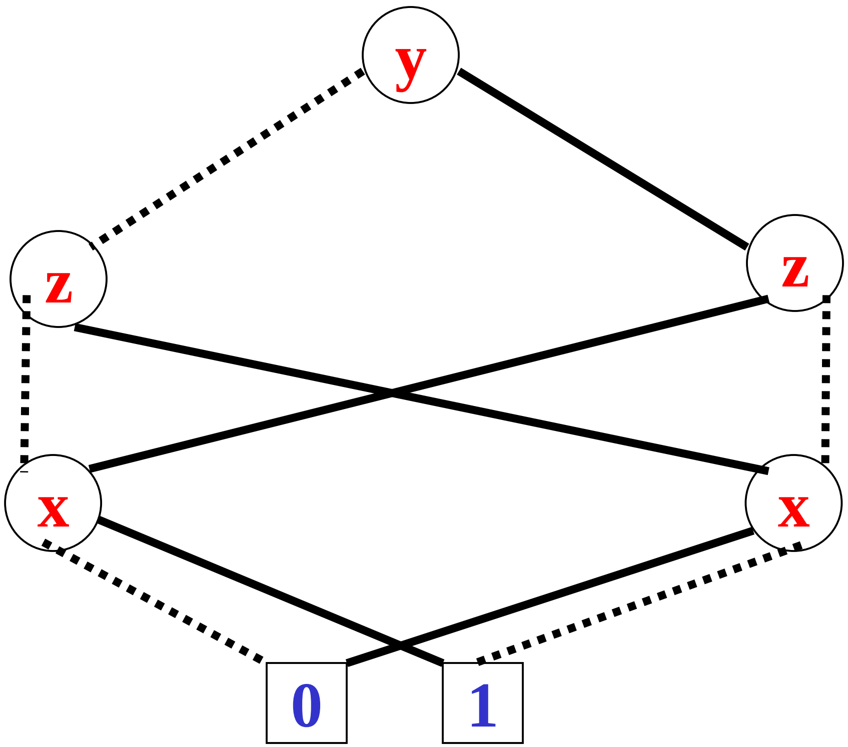



y
z
z
x
x
x
x
0
1
1
1
0
0
1


y
z
z
x
x
x
x
0
1
0
1
0
1


y
z
z
x
x
x
x
0
1


y
z
z
x
x
x
0
1


y
z
z
x
x
0
1

Reduced OBDD

- An **OBDD** is ***reduced*** (i.e. it is a **ROBDD**) if there are only ***two terminal vertices*** **0** and **1**, and for all ***non terminal vertices v***,***u***:
  - ***low(v) ≠ high(v)*** (***non-redundant tests***)
  - ***low(v) = low(u)***, ***high(v) = high(u)*** and ***var(v) = var(u)*** implies ***v = u*** (***uniqueness***)

Canonicity of ROBDD

Let us denote a **ROBDD** with its ***root node*** and the ***function*** represented by ***subgraph a rooted*** in node ***u*** with $\mathbf{f^u}$. Then:

Theorem: For any function $\mathbf{f:\{0,1\}^n \to \{0,1\}}$ ***there exists a unique*** **ROBDD** ***u*** with variable ordering $x_1, x_2, \ldots, x_n$ such that

$$\mathbf{f^u = f(x_1,\ldots,x_n)}$$

Consequences of canonicity

Theorem: For any function $\mathbf{f:\{0,1\}^n \rightarrow \{0,1\}}$ there exists a ***unique*** **ROBDD** ***u*** with variable ordering $x_1, x_2, \ldots, x_n$ such that

$$\mathbf{f^u = f(x_1,\ldots,x_n)}$$

Therefore we can say that:

- A function $\mathbf{f^u}$ is a ***tautology*** if its **ROBDD** ***u*** is ***equal*** to **1**.
- A function $\mathbf{f^u}$ is a ***satisfiable*** if its **ROBDD** ***u*** is ***not equal*** to **0**.

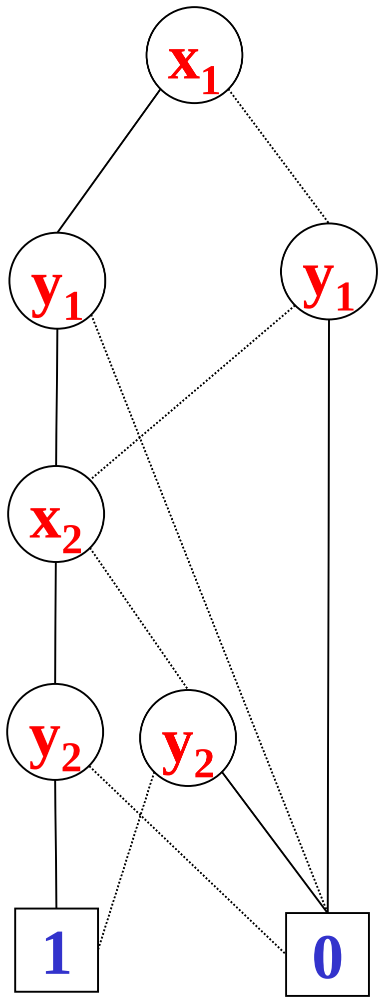
Reduced OBDDs

- ***The ordering is crucial!***
- $\{x_1, x_2, y_1, y_2\}$ $\quad x_1 \; x_2$
  - $f(x_1, x_2, y_1, y_2)$ $\quad y_1 \; y_2$
  - $f(x_1, x_2, y_1, y_2) = 1$ ***iff*** $(x_1 = y_1 \wedge x_2 = y_2)$
- If $x_1 < y_1 < x_2 < y_2$, then the **OBDD** is of size $3 \cdot 2 + 2 = 8$.
- If $x_1 < x_2 < y_1 < y_2$, then the **OBDD** is of size $3 \cdot 2^2 - 1 = 11$!

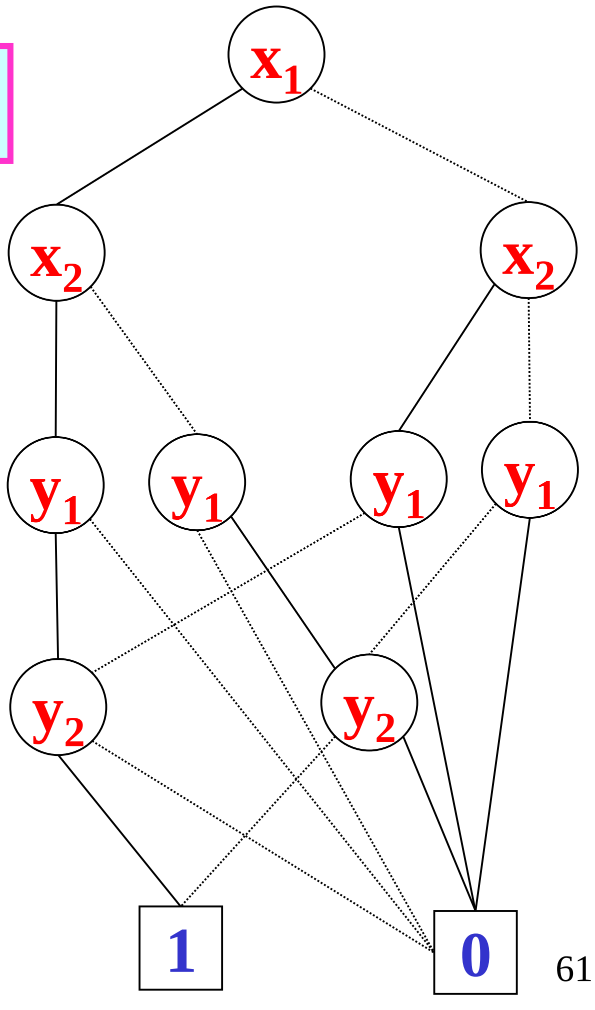
Reduced OBDDs

$x_1 < y_1 < x_2 < y_2$

$$(x_1 = y_1 \land x_2 = y_2)$$



$x_1 < x_2 < y_1 < y_2$



Reduced OBDDs

- ***The ordering is crucial!***
- $\{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$ $\qquad x_1 \;\; x_2 \ldots x_n$

  $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ $\qquad y_1 \; y_2 \ldots y_n$

  – $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = 1 \quad \textit{iff} \quad \bigwedge_{i=1}^{n} (x_i = y_i)$

- If $x_1 < y_1 < x_2 < y_2 \ldots < x_n < y_n$, then the **OBDD** is of size **$3n + 2$**.

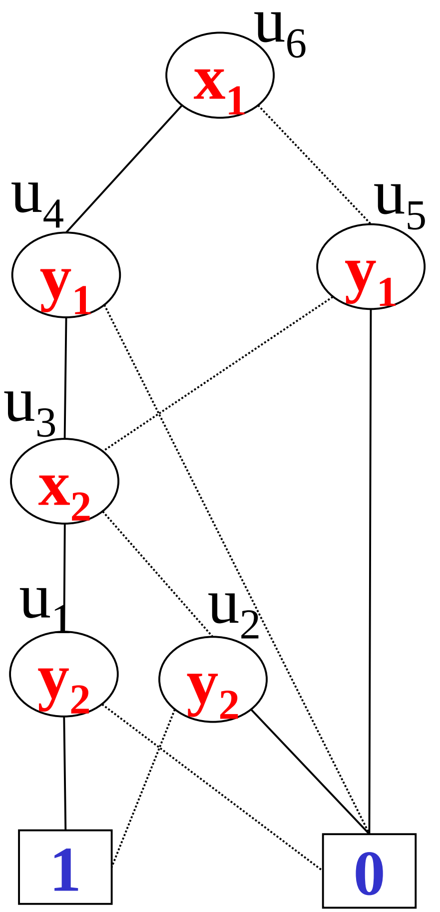
- If $x_1 < x_2 < \ldots < x_n < y_1 < \ldots < y_n$, then the **OBDD** is of size **$3 . 2^n - 1$** !

ROBDDs

- Finding the ***optimal variable ordering*** is ***computationally expensive*** (**NP-complete**).
- There are ***heuristics*** for finding “***good orderings***”.
- There exist boolean functions whose sizes are ***exponential*** (in the number of variables) for any ordering.
- Functions encountered in practice are **rarely** of this kind.

Implementation of ROBDDs

Array-based implementation



root = u_6

***T*[]=**

	Var	Low	High
0	?	?	?
1	?	?	?
u_1	y_2	0	1
u_2	y_2	1	0
u_3	x_2	u_2	u_1
u_4	y_2	0	u_3
u_5	y_1	0	u_3
u_6	x_1	u_5	u_4

The function MK

- The function **MK** searches for a node ***u*** with ***var(u)=x***$\boldsymbol{var(u)=x_i}$, ***low(u)=l*** and ***high(u)=h***. If the node does not exists, then creates the new node after inserting it. The running time is ***O(1)***.

H(i,l,h) **is a hash function mapping a triple *<i,l,h>* into a node index in T.**

```
Algorithm mk(i,l,h)
 if  l=h then
    return l
 else if T[H(i,l,h)] ≠ empty then
       return T[H(i,l,h)]
 else u = add(T,H(i,l,h),i,l,h)
        return u
```

Operations on ROBDDs.

- During model checking, boolean operations will have to be performed on **ROBDD**s.
- These operations can be implemented efficiently.
- **$f \vee g$ --------- G_f op$_\vee$ $G_g = G_{f \vee g}$**

- There is a procedure called **APPLY** to do this.

Operations on ROBDDs

- When performing an operation on **G** and **G'** we assume their variable orderings are ***compatible***.
- $\mathbf{X = X_G \cup X_{G'}}$
- There is an ordering $<$ on $\mathbf{X}$ such that:
  - $<$ restricted to $\mathbf{X_G}$ is $<_{\mathbf{G}}$
  - $<$ restricted to $\mathbf{X_{G'}}$ is $<_{\mathbf{G'}}$.

Operations on OBDDs

- The basic idea (Shannon Expansion):
- $\mathbf{f(x_1, x_2, \ldots, x_n)}$
  - $\mathbf{f|_{x_1 = 0} = f(0, x_2, \ldots, x_n)}$
    - $\mathbf{f = x_1 \vee (x_2 \wedge x_3)}$
    - $\mathbf{f|_{x_1 = 0} = x_2 \wedge x_3}$
  - Similarly, $\mathbf{f|_{x1 = 1} = f(1, x_2, \ldots, x_n)}$

$$\boxed{\mathbf{f(x_1, x_2, \ldots, x_n) = (\neg x_1 \wedge f|_{x_1 = 0}) \vee (x_1 \wedge f|_{x_1 = 1})}}$$

- This is true even if $\mathbf{x_1}$ does not appear in $\mathbf{f}$!

Operations on OBDDs: Negation

- The basic idea (Shannon Expansion):

$$f(x_1, x_2,\ldots,x_n) = (\neg x_1 \wedge f|_{x_1 = 0}) \vee (x_1 \wedge f|_{x_1 = 1})$$

- Therefore, assuming $x_1 < x_2 < \ldots < x_n$,

$$\neg f(x_1, x_2,\ldots,x_n) = \neg ((\neg x_1 \wedge f|_{x_1 = 0}) \vee (x_1 \wedge f|_{x_1 = 1}))$$

$$= (\neg(\neg x_1 \wedge f|_{x_1 = 0}) \wedge \neg(x_1 \wedge f|_{x_1 = 1}))$$

$$= ((x_1 \vee \neg f|_{x_1 = 0}) \wedge (\neg x_1 \vee \neg f|_{x_1 = 1})$$

$$= (x_1 \wedge \neg x_1) \vee (\neg x_1 \wedge \neg f|_{x_1 = 0}) \vee$$

$$\vee (x_1 \wedge \neg f|_{x_1 = 1}) \vee (\neg f|_{x_1 = 0} \wedge \neg f|_{x_1 = 1})$$

$$= (\neg x_1 \wedge \neg f|_{x_1 = 0}) \vee (x_1 \wedge \neg f|_{x_1 = 1})$$

Operations on ROBDDs.

- Let **x** be the top variable of $\mathbf{G_f}$ and **y** the top variable of $\mathbf{G_g}$.
- To compute $\mathbf{G_{f\ op\ g}}$ we consider:

  CASE1: x = y

  - $\mathbf{f\ op\ g = (\neg x \wedge (f|_{x=0}\ op\ g|_{x=0}) \vee}$
    $\mathbf{(x \wedge (f|_{x=1}\ op\ g|_{x=1})}$

  – We have to solve now two **smaller** problems!

Operations on ROBDDs.

- Let **x** be the top variable of $\mathbf{G_f}$ and **y** the top variable of $\mathbf{G_g}$.
- To compute $\mathbf{G_{f\ op\ g}}$ we consider:

CASE2: x < y.

- Then **x** does not appear in $\mathbf{G_g}$ (why?).
- $\mathbf{g|_{x=0} = g = g|_{x=1}}$
  - $\mathbf{f\ op\ g = (\neg x \wedge (f|_{x=0}\ op\ g\ ) \vee (x \wedge (f|_{x=1}\ op\ g)}$
- We have to solve now two **smaller** problems!

CASE2: x > y is symmetric.

Operations on ROBDDs.

- To compute $\mathbf{G_{f\ op\ g}}$ we consider:

  Base (terminal) cases depend upon **op**

  Eg.: if **op** $= \vee$ then $\{0,0 \to \mathbf{0};\ \mathbf{1}\}$

  if **op** $= \wedge$ then $\{1,1 \to \mathbf{1};\ \mathbf{0}\}$

  ….

Notice that $\neg\mathbf{f(x_1,x_2,\ldots,x_n) = f(x_1,x_2,\ldots,x_n) \oplus 1}$, therefore ***negation*** can be implemented with ***Apply***.

Algorithm for Apply

```
Algorithm Apply(op,u,v)

  Function App(u,v)
     if terminal_case(op,u,v) then return op(u,v)
     else if var(u) = var(v) then
         u = mk(var(u), App(op,low(u),low(v)),
                        App(op,high(u),high(v)))
     else if var(u) < var(v) then
         u = mk(var(u),App(op,low(u), v), App(op,high(u),v))
     else /* var(u) > var(v) */
         u = mk(var(u),App(op,u,low(v)), App(op,u,high(v)))
     return u

return App(u,v)
```

If n = number of variables, then *running time* = $O(2^n)$. Why?

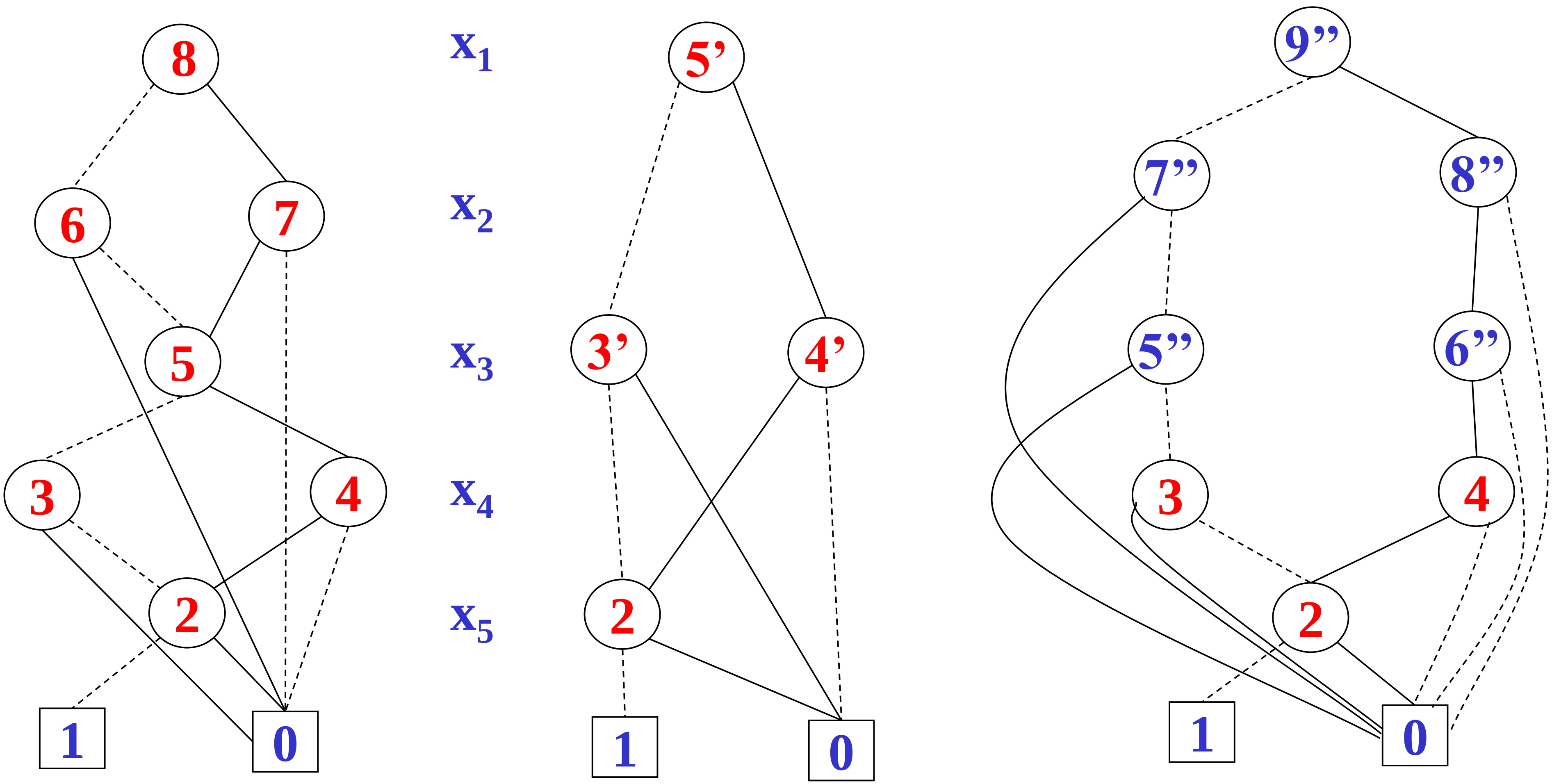
Efficient algorithm for Apply

```
Algorithm Apply(op,u,v)
   init(G_op)
  Function App(u,v)
     if  G_op(u,v) ≠ empty then return G_op(u,v)
     else if terminal_case(op,u,v) then return op(u,v)
     else if var(u)=var(v) then
          r = mk(var(u), App(op,low(u),low(v)),
                              App(op,high(u),high(v)))
     else if var(u) < var(v) then
          r = mk(var(u),App(op,low(u), v), App(op,high(u),v))
     else /* var(u) > var(v) */
          r = mk(var(u),App(op,u,low(v)), App(op,u,high(v)))
     G_op(u,v) = r
     return r
return App(u,v)
```

running time = $O(|G_u||G_v|)$. Why?

Exemple of Apply $\wedge$

$(x_1 \equiv x_2) \wedge (x_3 \equiv x_4) \wedge \neg x_5$ $\wedge$ $(x_1 \equiv x_3) \wedge \neg x_5$ $=$ $((x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee$ $\vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4)) \wedge \neg x_5$



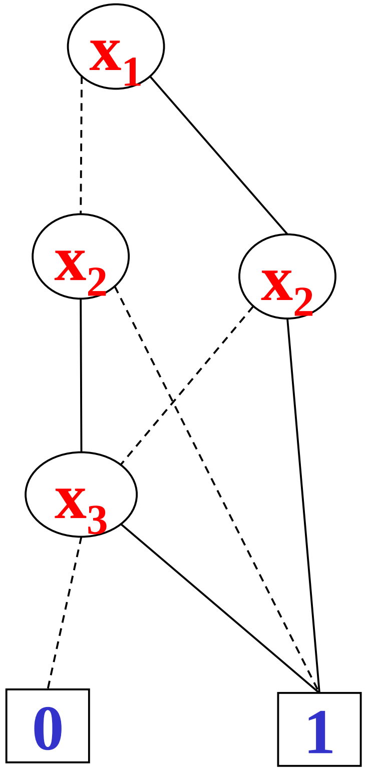
The Restrict operation

- ***Problem***: Given a (partial) truth assignment $x_1=b_1,\ldots,x_k=b_k$ (where $b_j=0$ or $b_j=1$), and a ROBDD t^u, compute the restriction of t^u under that assignment.
- E.G.: if $f(x_1,x_2,x_3) = ((x_1 \Leftrightarrow x_2) \vee x_3)$ we want to compute $f(x_1,x_2,x_3)[0/x_2] = f(x_1,0,x_3)$

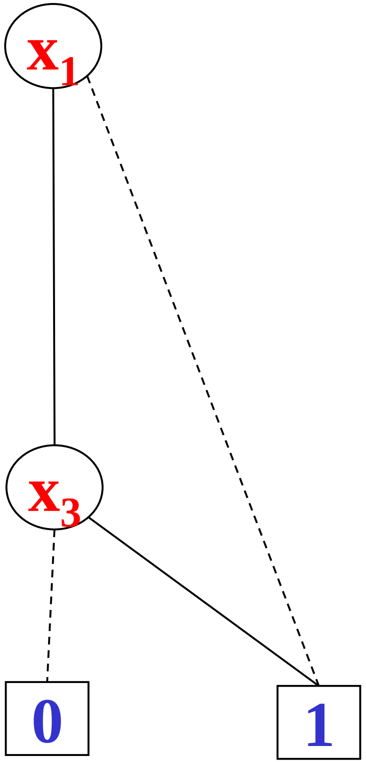
  i.e.: $f(x_1,0,x_3) = \neg x_1 \vee x_3$

Restrict Operation: example

$f(x_1,x_2,x_3) = ((x_1 \Leftrightarrow x_2) \vee x_3)$



$f(x_1,x_2,x_3)[0/x_2] = \neg x_1 \vee x_3$



Restrict Operation

- Let **x** be the root of $\mathbf{G_f}$
- To compute $\mathbf{G_f|_{y=b}}$ we consider:

CASE1: x = y

- $\mathbf{f|_{y=b} = low(G_f)}$ **if b=0**
- $\mathbf{f|_{y=b} = high(G_f)}$ **if b=1**

Restrict Operation

- Let **x** be the root of $\mathbf{G_f}$
- To compute $\mathbf{G_f|_{y=b}}$ we consider:

CASE2: x > y

- $\mathbf{f|_{y=b} = f}$

Restrict Operation

- Let **x** be the root of $\mathbf{G_f}$
- To compute $\mathbf{G_f|_{y=b}}$ we consider:

CASE2: x < y

- $\mathbf{f|_{y=b} = (\neg\, x \wedge (f|_{x=0})|_{y=b}) \vee (x \wedge (f|_{x=1})|_{y=b})}$

- We have to solve now two **smaller** problems!

Algorithm for Restrict

```
Algorithm Restrict(u,i,b)
  Function Res(u)
    if var(u) > i then return u
    else if var(u) < i then
        return mk(var(u),Res(low(u)),Res(high(u)))
    else /* var(u) = i */
        if b = 0 then
          return Res(low(u))
        else /* var(u) = i and b = 1 */
          return Res(high(u))
return Res(u)
```

running time = $O(2^n)$. Why?

Efficient algorithm for Restrict

```
Algorithm Restrict(u,i,b)
  init(G_res)
 Function Res(u)
   if  G_res(u) ≠ empty then return G_res(u)
   if var(u) > i then return u
   else if var(u) < i then
       r = mk(var(u),Res(low(u)),Res(high(u)))
   else /* var(u) = var(v) */
       if b = 0 then
           r = Res(low(u))
       else /* var(u) = var(v) and b = 1 */
           r = Res(high(u))
   G_res(u) = r
   return r
return Res(u)
```

running time = $O(|G_u|)$. Why?

Quantification

- Extend the boolean language with

$$\exists x.t \mid \forall x.t$$

- They can be defined in terms of ROBDD operations:

$$\exists x.t = t[0/x] \vee t[1/x]$$

$$\forall x.t = t[0/x] \wedge t[1/x]$$

We can use an appropriate combination of ***Restrict*** and ***Apply***

Symbolic CTL Model Checking

- Represent the required **subsets of states** as boolean functions and hence as **ROBDD**s.
- Represent the **transition relation** as a boolean function and hence as a **ROBDD**.
- Reduce the iterative **fixed point computations** of the model checking process to **operations on OBDDs**.
- Check for the **termination** of the **fixpoint** computation by checking **ROBDD equivalence**.

Symbolic Model Checking

- **$K = (S, S_0, R, AP, L)$**
- Assume that if **$L(s) = L(s')$** then **$s = s'$**.
  - If not, ***add*** a few ***new atomic propositions*** if necessary, so as to distinguish states only based on the labeling.
- **$AP = \{p, q, r\}$**
- **$L(s) = \{p\}$**
  - $f_s = p \wedge \neg q \wedge \neg r$
- $f_{\{s_1, s_2, s_5\}} = f_{s_1} \vee f_{s_2} \vee f_{s_5}$

Symbolic Model Checking

- **$K = (S, S_0, R, AP, L)$**
- **$AP = \{p, q, r\}$**
- ***Add*** the next-state boolean variables **$\{p', q', r'\}$**
- Suppose **(s_1, s_2)** in **R** (i.e. **$R(s_1, s_2)$**)

  with **$L(s_1) = \{p, q\}$** and **$L(s_2) = \{r\}$**.

  Then $f_{R(s_1,s_2)} = f_{s_1} \wedge f'_{s_2}$.
  - where $f_{s_1} = p \wedge q \wedge \neg r$ and $f'_{s_2} = \neg p' \wedge \neg q' \wedge r'$

- $f_R = \bigvee_{(s_1, s_2) \in R} (f_{R(s_1,s_2)})$
- **Choose the ordering** $p < p' < q < q' < r < r'$ **!**

CTL symbolic Model Checking

- $\|[x_i]\| = f_{x_i}(x_1,\ldots,x_n) = x_i$

  (the OBDD for the *boolean variable* x_i)
- $\|[\neg\phi]\| = \neg f_{\phi}(x_1,\ldots,x_n)$

  (apply negation to the OBDD for ϕ)
- $\|[\phi \vee \psi]\| = f_{\phi}(x_1,\ldots,x_n) \vee f_{\psi}(x_1,\ldots,x_n)$

  (apply $\vee$ operation to the OBDDs for ϕ and ψ)
- $\|[\phi \wedge \psi]\| = f_{\phi}(x_1,\ldots,x_n) \wedge f_{\psi}(x_1,\ldots,x_n)$

  (apply $\wedge$ operation to the OBDDs for ϕ and ψ)

CTL Symbolic Model Checking

- **$\|[\mathrm{EX}\ \phi]\| =$**

  $\exists x'_1,\ldots,x'_n(f_\phi(x'_1,\ldots,x'_n) \wedge f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n))$

This is also called the *relational product*, or the *pre-image of* $\|[\phi]\|$ by *R* (see ***Section* 6.6** in ***Clarke's book*** for a more ***efficient algorithm*).**

- **$\|[\mathrm{EU}(\phi,\psi)]\| = \mu Z.(f_\psi(x_1,\ldots,x_n) \vee (f_\phi(x_1,\ldots,x_n) \wedge \mathrm{EX}\ Z))$**

- **$\|[\mathrm{EG}\ \phi]\| = \nu Z.(f_\phi(x_1,\ldots,x_n) \wedge \mathrm{EX}\ Z)$**

Symbolic model checking: example

Let $\mathbf{V=\{x_1,\ldots,x_n\}}$, then $\mathbf{\|[EG\ \psi]\|}$ can be computed as follows:

1. Assume the ROBDD $\mathbf{f_\psi(x_1,\ldots,x_n)}$ has been computed.
2. Set $\mathbf{X_0 = f_\psi (x'_1,\ldots,x'_n)}$ [computed from $\mathbf{f_\psi(x_1,\ldots,x_n)}$ by *variable substitution*]
3. We need to compute $\mathbf{X_{i+1} = X_i \cap Y_i}$ where:

   $\mathbf{Y_i = \exists x'_1,\ldots,x'_n(f_\psi(x'_1,\ldots,x'_n) \wedge f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n))}$

   $\mathbf{X_{i+1}}$ can easily be computed as $\mathbf{X_i \wedge Y_i}$
4. Check **whether $X_{i+1} = X_i$** by checking whether the corresponding ROBDDs are **identical**.
5. If **not**, substitute the ***next-state*** variables for the ***state-variables*** in $\mathbf{X_{i+1}}$, and repeat from **step 3**.

Algorithm Compute_EG(β)

$\mathbf{f_1}(\boldsymbol{x}) := \mathbf{f_\beta}(\boldsymbol{x})$;

j=1;

repeat

$\mathbf{j} := \mathbf{j+1}$;

$\mathbf{f_j} := \mathbf{f_\beta}(\boldsymbol{x}) \wedge \exists \boldsymbol{x}'.(\mathbf{f_R}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathbf{f_{j-1}}(\boldsymbol{x}'))$;

until $\mathbf{f_j}(\boldsymbol{x}) = \mathbf{f_{j-1}}(\boldsymbol{x})$;

Algorithm Compute_EU(β_1,β_2)

$\mathbf{f_1}(\boldsymbol{x}) := \mathbf{f_{\beta_2}}(\boldsymbol{x})$;

j=1;

repeat

$\mathbf{j} := \mathbf{j+1}$;

$\mathbf{f_j} := \mathbf{f_{\beta_2}}(\boldsymbol{x}) \vee (\mathbf{f_{\beta_1}}(\boldsymbol{x}) \wedge \exists \boldsymbol{x}'.(\mathbf{f_R}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathbf{f_{j-1}}(\boldsymbol{x}')))$;

until $\mathbf{f_j}(\boldsymbol{x}) = \mathbf{f_{j-1}}(\boldsymbol{x})$;

CTL Symbolic model checking

Finally, assuming boolean variables $\mathbf{V=\{x_1,\ldots,x_n\}}$, and the ROBDD for $\|[\phi]\|$ already computed.

- Checking whether

$$\mathbf{K} \models \phi$$

amounts to checking whether the ROBDD for $\mathbf{f_{Init}} \wedge \mathbf{f_{\neg\phi}}$ is **identical** to the ROBDD for **0**, where $\mathbf{f_{Init}}$ is the ROBDD for the set $\|[\mathbf{Init}]\|$ of **initial states** of **K**.

(recall that $\mathbf{K} \models \phi$ iff $\|[\mathbf{Init}]\| \subseteq \|[\phi]\|$ iff $\|[\mathbf{Init}]\| \cap \|[\neg\phi]\| = \varnothing$)

Pre-image computation with BDD

Let us consider the ***Pre-image*** operation

$$\exists x'_1,\ldots,x'_n(f_\psi(x'_1,\ldots,x'_n) \wedge f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n))$$

Pre-image is a special case of the ***Relational Product***

$$\exists x' (R_1(x,x') \wedge R_2(x',z))$$

where R_1 is R, R_2 is ψ and z is empty.

Pre-image can easily be computed by applying $\wedge$ to the BDD's for ψ and R, and then existential elimination of the primed variables.

However, the intermediate BDD for

$$f_\psi(x'_1,\ldots,x'_n) \wedge f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n)$$

is usually far bigger than the final result.

This be avoided by exploiting ***early quantification***, whenever possible.

Pre-image computation with BDD

Early quantification is based on the fact that:

- If $\mathbf{x_1 < x_2}$ and the top variable of $\mathbf{f}$ is $\mathbf{x_1}$ then

$$\exists x_2\, (x_1 \rightarrow f|_{x_1=1}, f|_{x_1=0}) \equiv (x_1 \rightarrow \exists x_2\, f|_{x_1=1}, \exists x_2\, f|_{x_1=0})$$

(recall that $\exists x\, (f \text{ op } g) \equiv f \text{ op } \exists x\, g$, whenever $\mathbf{f}$ does not depend on $\mathbf{x}$)

- If the top variable of $\mathbf{g}$ is $\mathbf{x_2}$ then

$$\exists x_2\, (x_2 \rightarrow f|_{x_2=1}, f|_{x_2=0}) \equiv (f|_{x_2=1} \vee f|_{x_2=0})$$

This means that we can devise an algorithm that computes the ***pre-image*** by applying quantification as soon as it is possible.

This avoids computing the ***conjunction*** (which is usually bigger than the final result) during the computation of the ***pre-image***.

Pre-image computation with BDD

```
Algorithm RelationalProduct(u,v,𝓘)  /* ∃𝓘 (f^u ∧ f^v) */
  init(G)
  Function RelPrd(u,v,𝓘)
     if u = 0 or v = 0 then return 0
     if u = 1 and v = 1 then return 1
     if  G(u,v) ≠ NIL then return G(u,v)
     z = min(var(u), var(v))
     if var(v) = var(u) then
          r1 = RelPrd(high(u), high(v), 𝓘) ; r2 = RelPrd(low(u), low(v), 𝓘)
    else if z = var(u) then
          r1 = RelPrd(high(u), v, 𝓘) ; r2 = RelPrd(low(u), v, 𝓘))
    else /* z = var(v) */
          r1 = RelPrd(u, high(v), 𝓘) ; r2 = RelPrd(u, low(v), 𝓘))
    if z ∉ 𝓘 then  /* z is not a quantified variable */
        r = mk(z, r1, r2)
    else /* z ∈ 𝓘 : z is a quantified variable */
        r = Apply(∨, r1, r2)
     G(u,v) = r
     return r

return RelPrd(u,v,𝓘)
```

Symbolic Model Checking

- The actual Kripke structure will be, in general, too large.
  - **State explosion**.
- So one must try to **compute the ROBDDs directly from the system model (NuSMV program)** and run the model checking procedure with the help of this **implicit** representation.
  - **Symbolic model checking**.
- This may not be sufficient, though! **Additional techniques may be needed** (e.g., **abstraction**).