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

- **$TS_1 = (S_1, S_1^0,$ S$_1$, $R_1)$** **R_1** µ **S_1** £ S$_1$ £ **S_1**
- **$TS_2 = (S_2, S_2^0,$ S$_2$, $R_2)$** **R_2** µ **S_2** £ S$_2$ £ **S_2**

- **a** 2 S$_1$ and **a** Ï S$_2$
  - An “***internal***” action of **TS_1**.
- **a** 2 S$_1$ Å S$_2$
  - A ***common (synchronizing)*** action of **TS_1** and **TS_2**.

Parallel Composition

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

Parallel Composition

- $TS_1 = (S_1, S_1^0, S_1, R_1)$ R_1 µ S_1 £ S_1 £ S_1
- $TS_2 = (S_2, S_2^0, S_2, R_2)$ R_2 µ S_2 £ S_2 £ S_2
- $TS = (TS_1$ k $TS_2) = (S, S^0, S, R)$.

– R µ S £ S £ S

  - $S = S_1$ £ S_2.

– R((s1, s2), a, (t1, t2)) ?

– if a 2 S_1 and a Ï S_2

– then R_1(s1, a, t1) and s2 = t2.

Parallel Composition

- $TS_1 = (S_1, S_1^0, S_1, R_1)$ R_1 µ S_1 £ S_1 £ S_1
- $TS_2 = (S_2, S_2^0, S_2, R_2)$ R_2 µ S_2 £ S_2 £ S_2
- $TS = (TS_1$ k $TS_2) = (S, S^0, S, R)$.

  - R µ S £ S £ S
    - S = S_1 £ S_2.
  - R((s1, s2), a, (t1, t2)) ?
  - if a 2 S_2 and a Ï S_1
  - then R_2(s2, a, t2) and s1 = t1.

Parallel Composition

- $TS_1 = (S_1, S_1^0, S_1, R_1)$ R_1 µ S_1 £ S_1 £ S_1
- $TS_2 = (S_2, S_2^0, S_2, R_2)$ R_2 µ S_2 £ S_2 £ S_2
- $TS = (TS_1$ k $TS_2) = (S, S^0, S, R)$.

- R µ S £ S £ S
  - S = S_1 £ S_2.
- R((s1, s2), a, (t1, t2)) ?
- if a 2 S_1 and a 2 S_2
- then R_1(s1, a, t1) and R_2(s2, a, t2)

Parallel Composition

- $\mathbf{TS} = (\mathbf{TS}_1 \text{ k } \mathbf{TS}_2) \text{ k } \mathbf{TS}_3$
- $\mathbf{TS} = \mathbf{TS}_1 \text{ k } (\mathbf{TS}_2 \text{ k } \mathbf{TS}_3)$
- $\mathbf{TS} = \mathbf{TS}_1 \text{ k } \mathbf{TS}_2 \text{ k } \mathbf{TS}_3$

Parallel Composition

- **$TS = TS_1$ k TS_2 …k TS_n**
- **Size(TS_i) ¼ jS_ij = k_i > 2**
- **Description of *TS* ¼ $k_1 + k_2$ …+ k_n**
- **Size(TS) = k_1 £ k_2…£ k_n**
  > 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 that:
  - **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)**
- y a **CTL** formula
- To check whether:
  - **K, s** 2 y
- We need to
  - compute ||[y]|| = **states(**y **) = {s** j **K, s** ⊨ y **}**.
  - then check whether **s** 2 ||[y]||.

Symbolic Model checking

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

- **S' µ S** can be represented as a ***boolean function***.
- **R** can be represented as a ***boolean function***.
- ||[y]|| 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 ! Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = {0,1} £ ….£ {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 ! Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = {0,1} £ ….£ {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

g : {0, 1} £ {0, 1} £ {0, 1} ! {0, 1}

Boolean Expressions

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

  t ::= ***x*** | **0** | **1** | Ø***t*** | ***t*** Ù ***t*** | ***t*** Ú ***t*** | ***t*** Þ ***t*** | ***t*** Û ***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** Ù **y** represents the function:
  - **f : {0, 1} £ {0, 1} ! {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** Ù **y** represents the function:
  - **f : {0, 1} £ {0, 1} ! {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

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

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

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 Ù y Ù Øz) Ú (x Ù Øy Ù z) Ú (Øx Ù 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 Ù y Ù Øz) Ú (x Ù Øy Ù z) Ú (Øx Ù y)

g : {0, 1} £ {0, 1} £ {0, 1} ! {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**(y), **EU**(y_1, y_2) and **EG**(y) to manipulating boolean functions and checking for **boolean function equality**.
- Go from **NuSMV** (program) representation ***directly*** to its **BDD** representation!

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**.

Boolean decision trees.

If-Then-Else representation

x Ù **y = x ® y , 0**

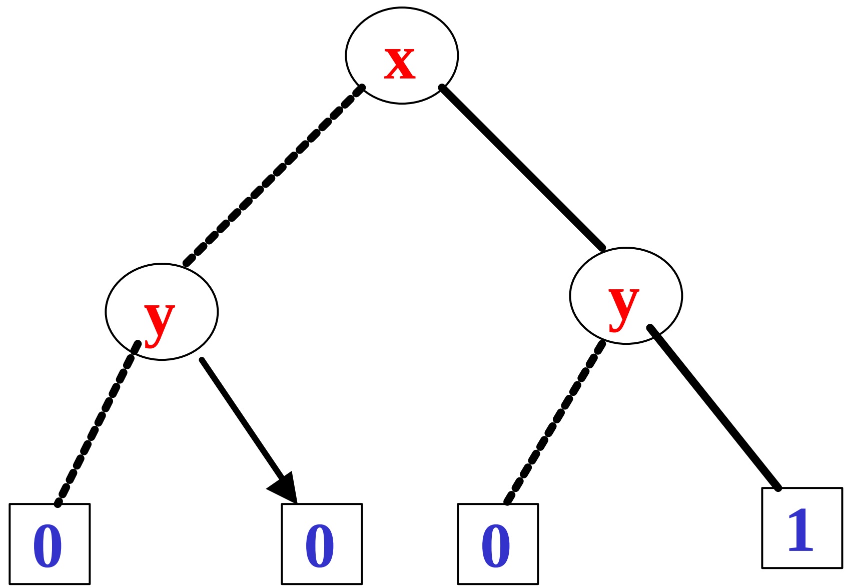
Shannon Expansion:

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

where

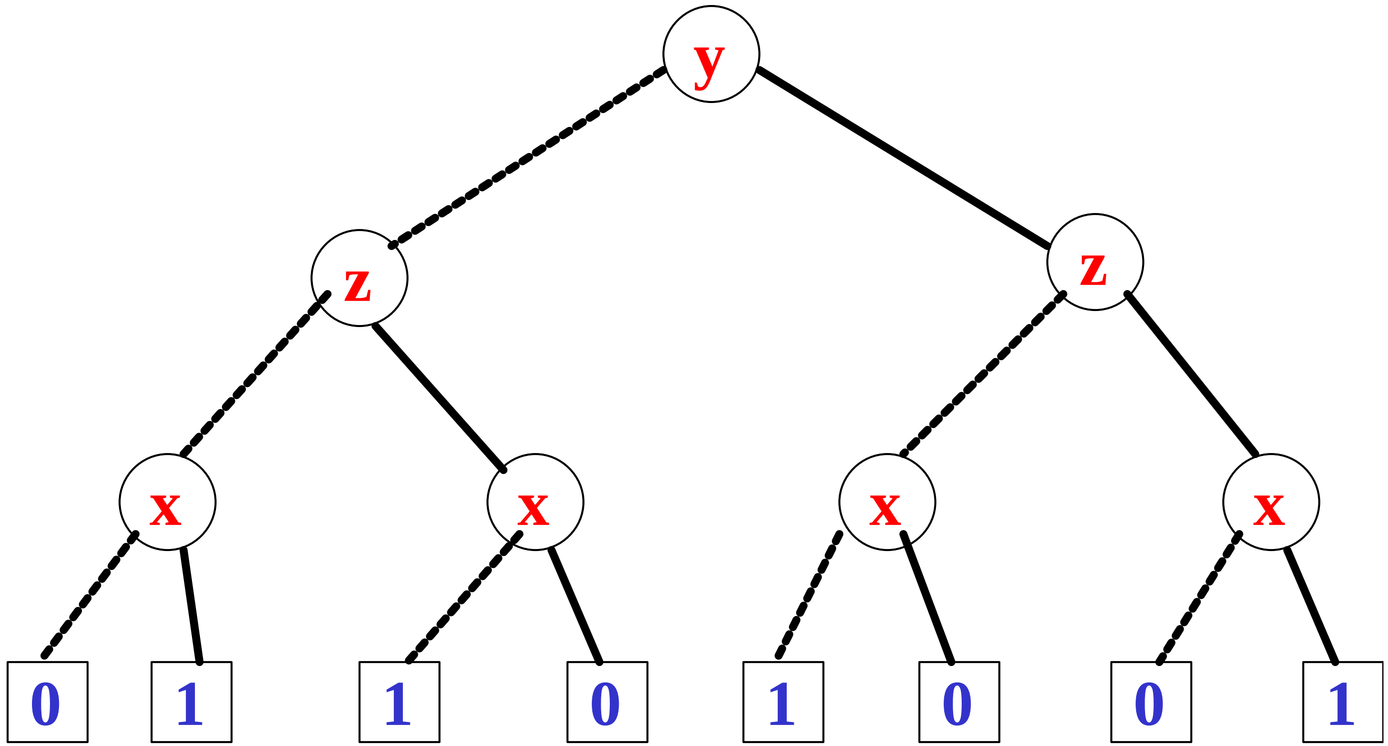
$$\mathbf{f_{[a/x]}(\ldots,x,\ldots) = f(\ldots,a,\ldots)}$$

for **a = 0,1**.



x Ù **y**

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

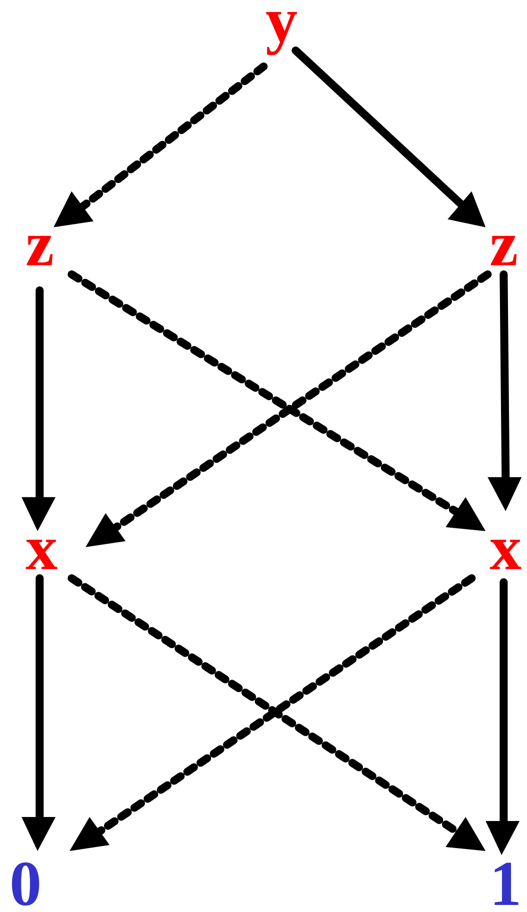


g = (y Ù (x Û z)) Ú (Øy Ù (x Û Øz))

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)***).



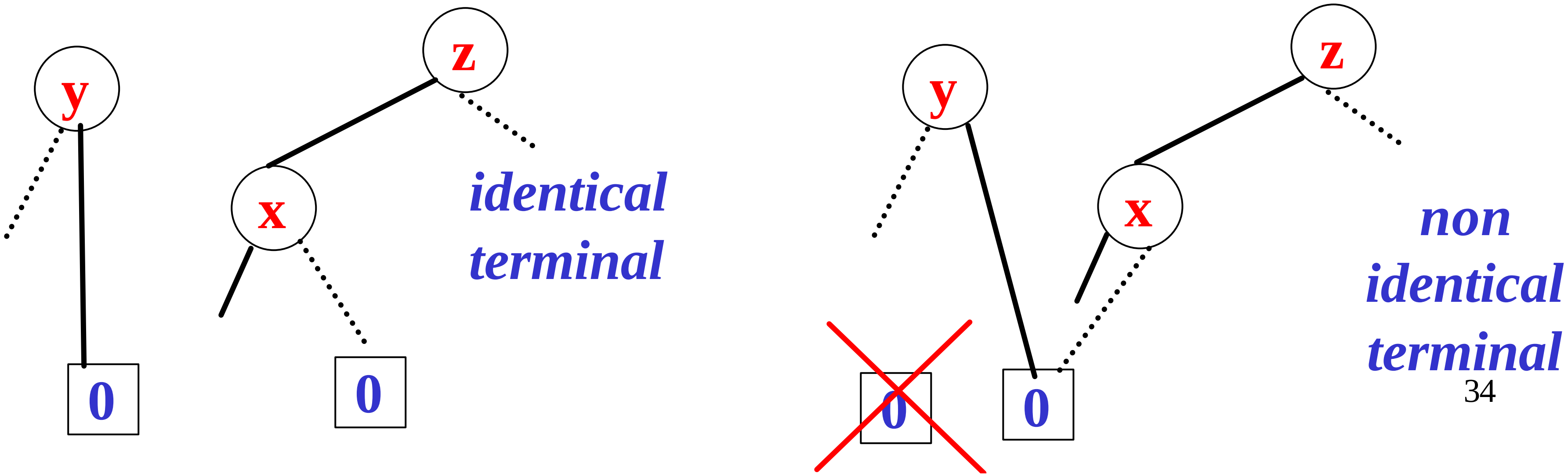
g = (y Ù (x Û z)) Ú (Øy Ù (x Û Øz))

Reduction Rules

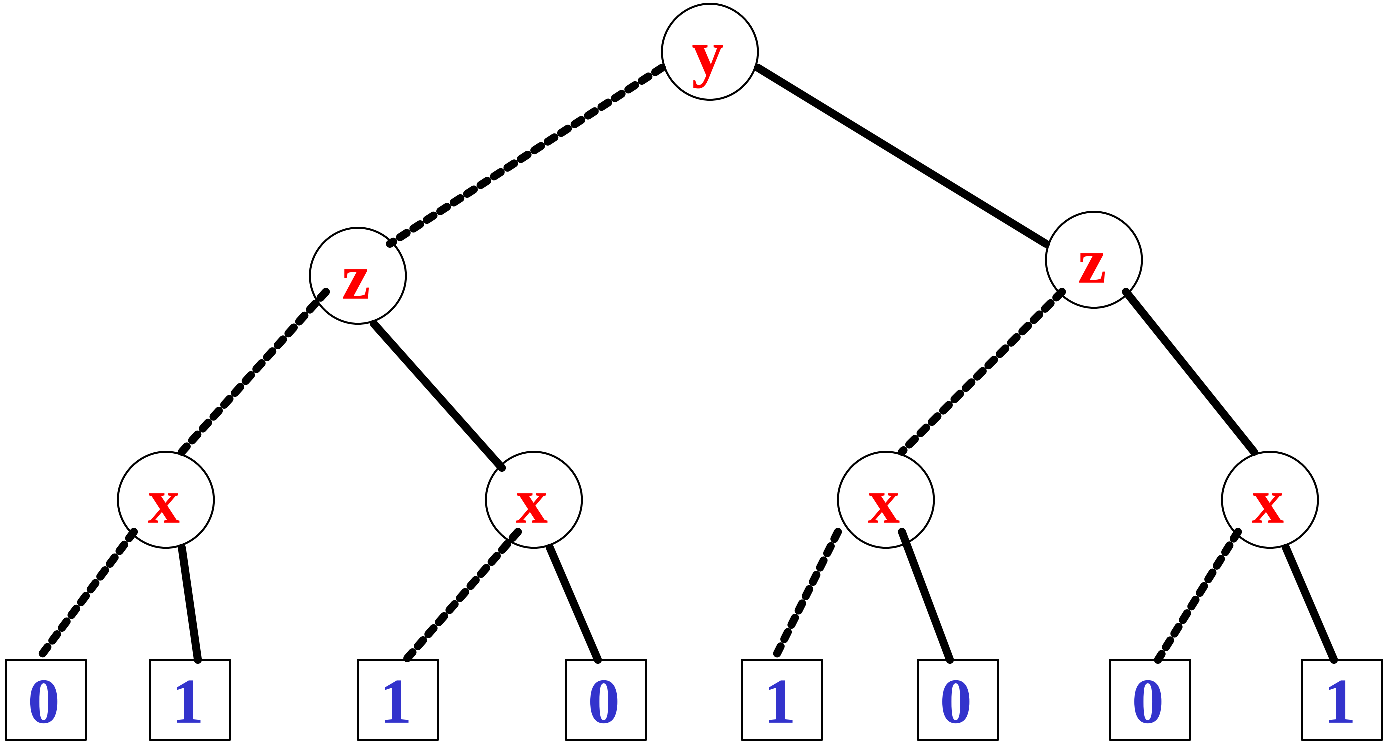
- Three reduction rules:
  - **Share identical terminal nodes. (R1)**
  - **Remove redundant tests (R2)**
  - **Share identical non-terminal nodes. (R3)**

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.**

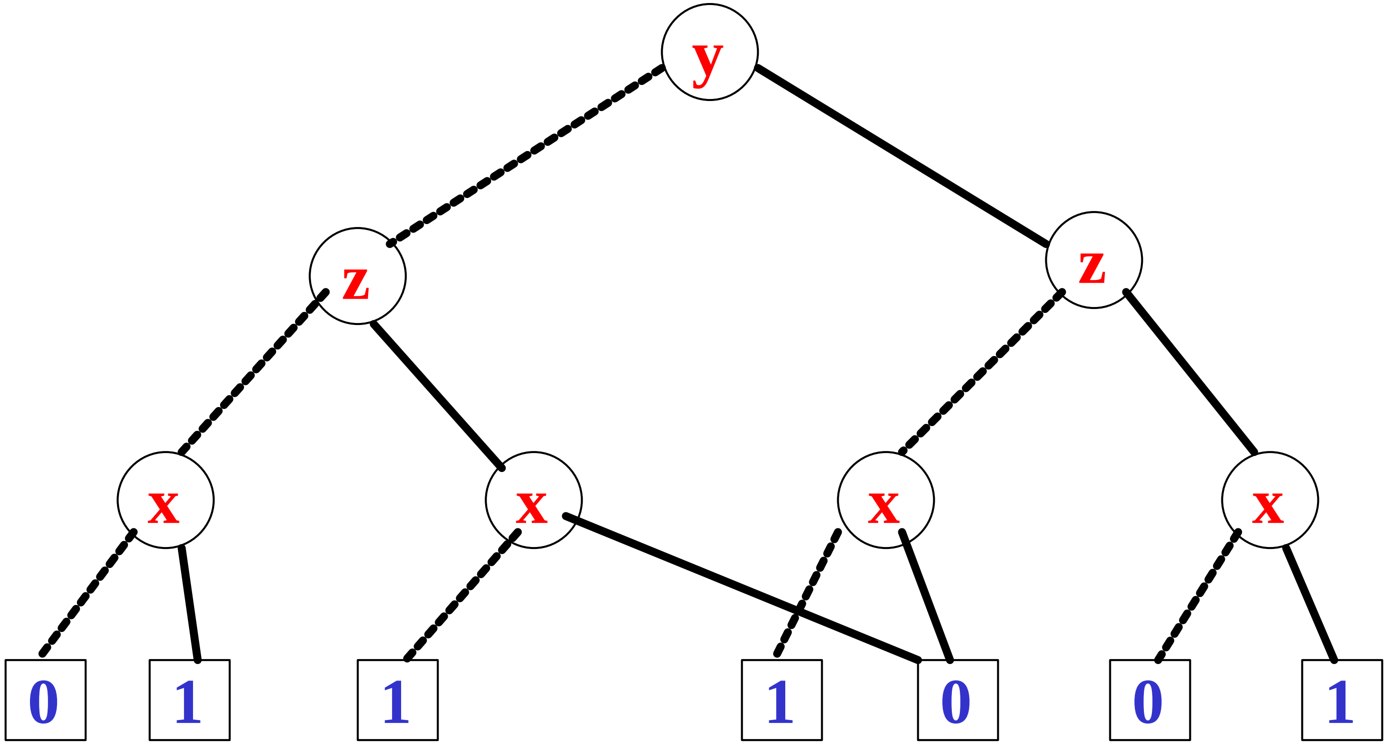


Share identical terminal nodes. (R1)



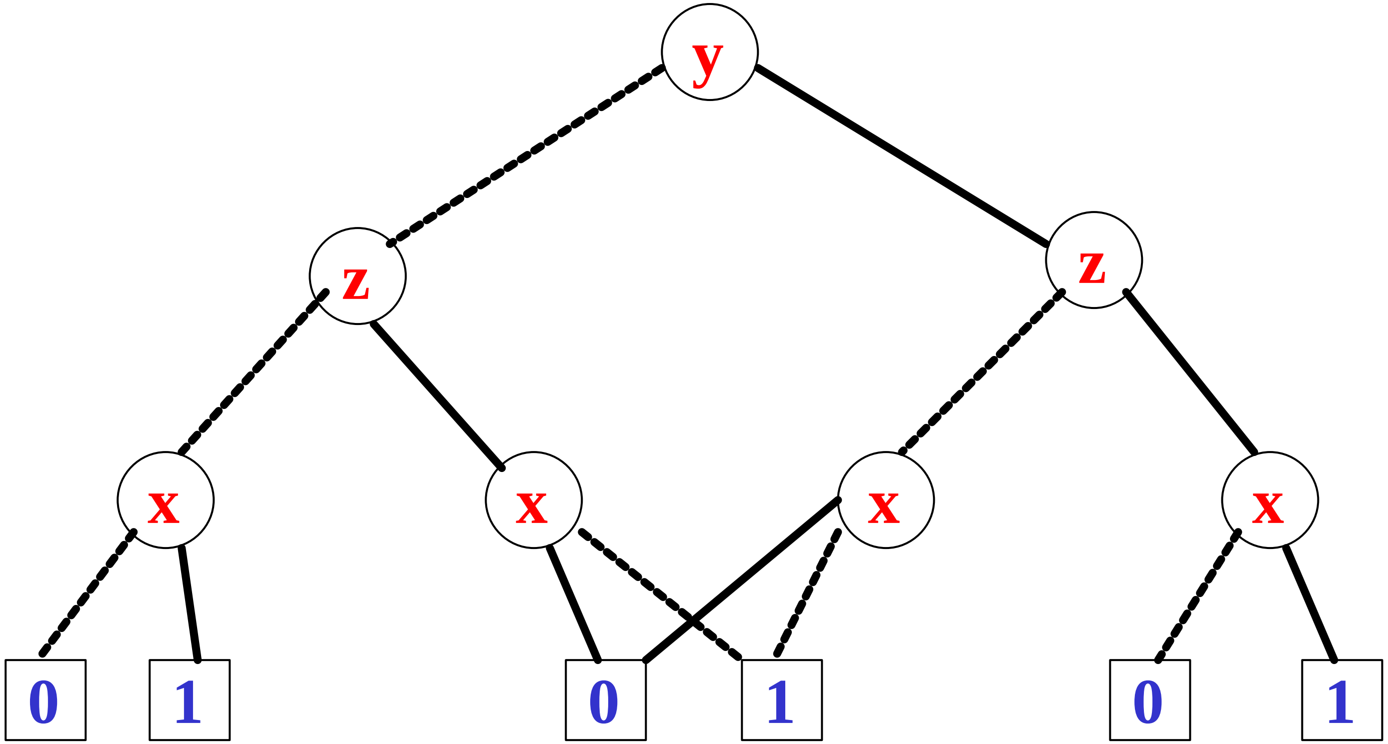
g = (y Ù (x Û z)) Ú (Øy Ù (x Û Øz))

Share identical terminal nodes. (R1)



g = (y Ù (x Û z)) Ú (Øy Ù (x Û Øz))

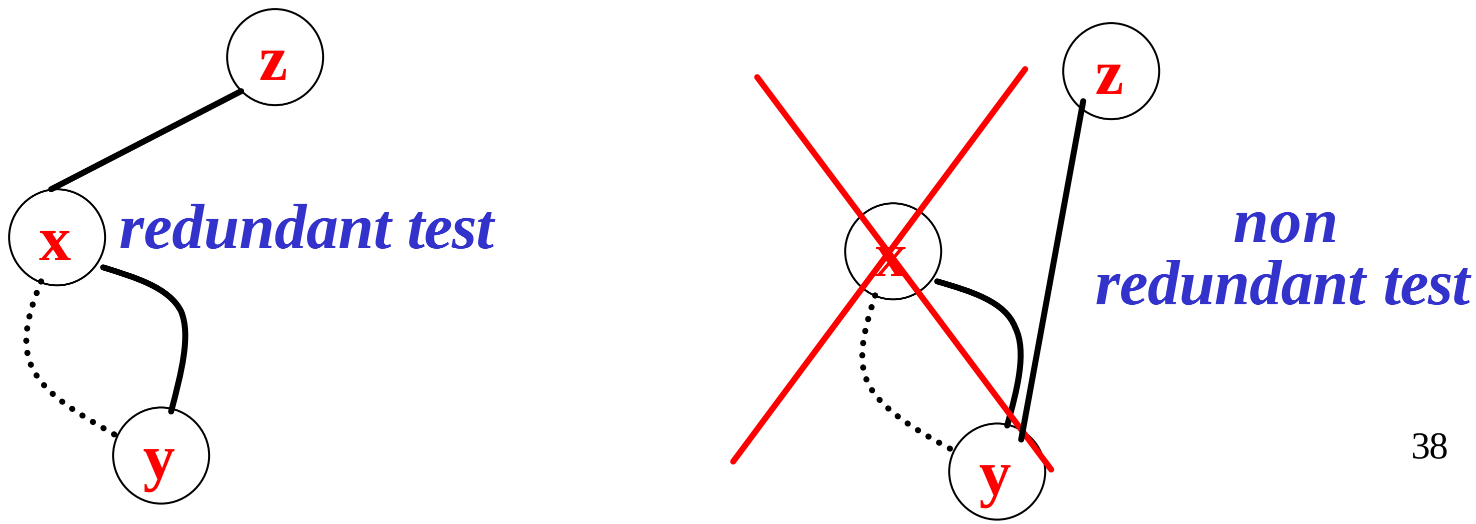
Share identical terminal nodes. (R1)



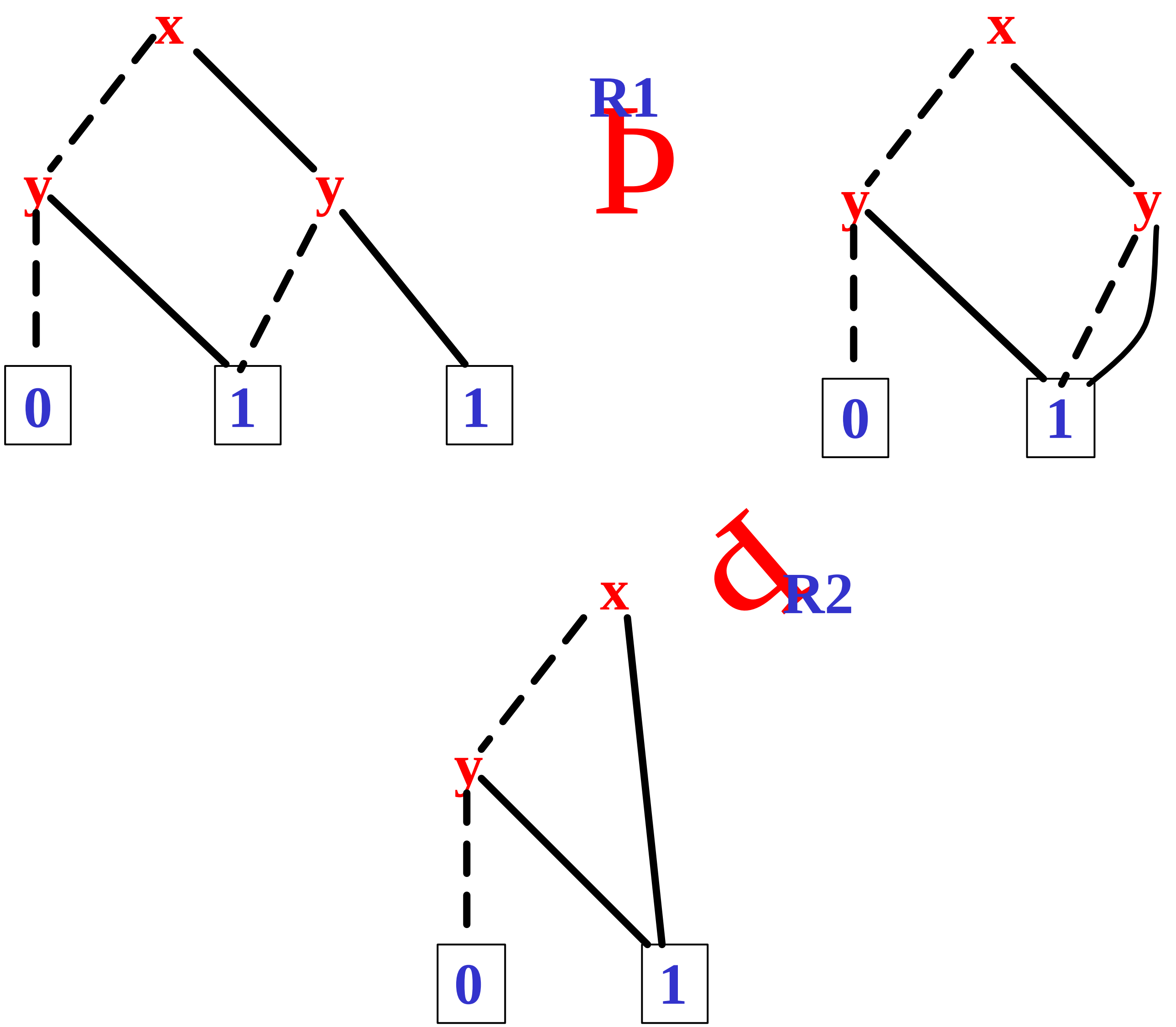
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

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**.

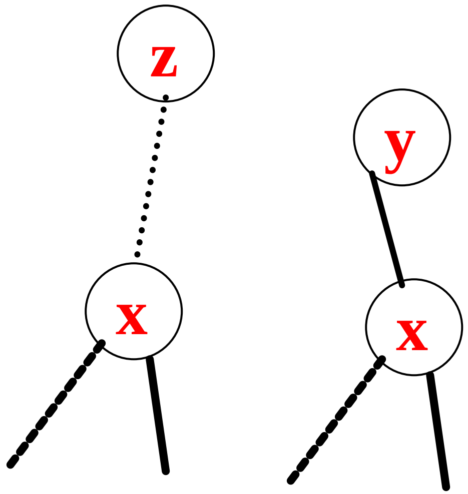


Remove redundant tests (R2)

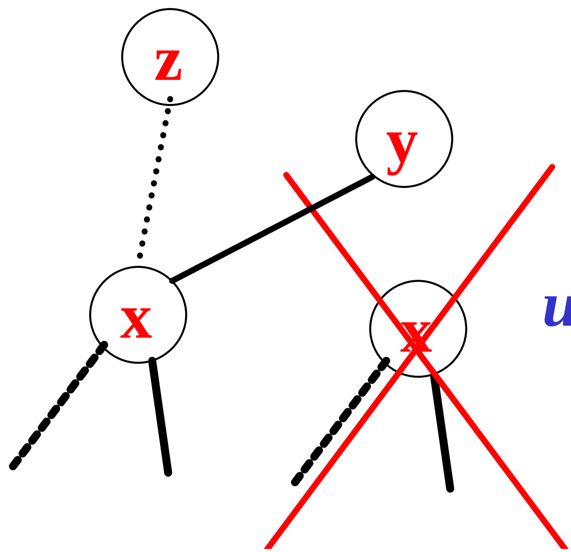


Reduction Rules

- 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**.

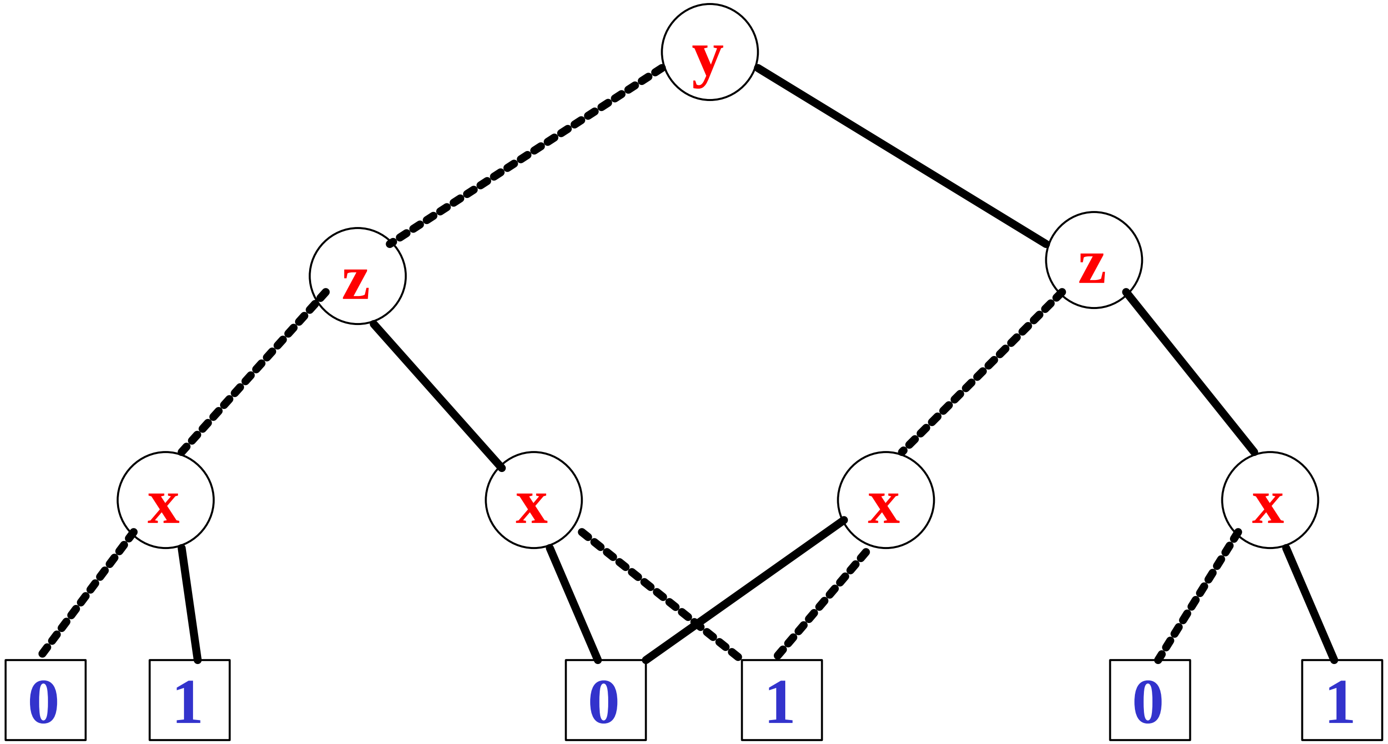


non-unique node



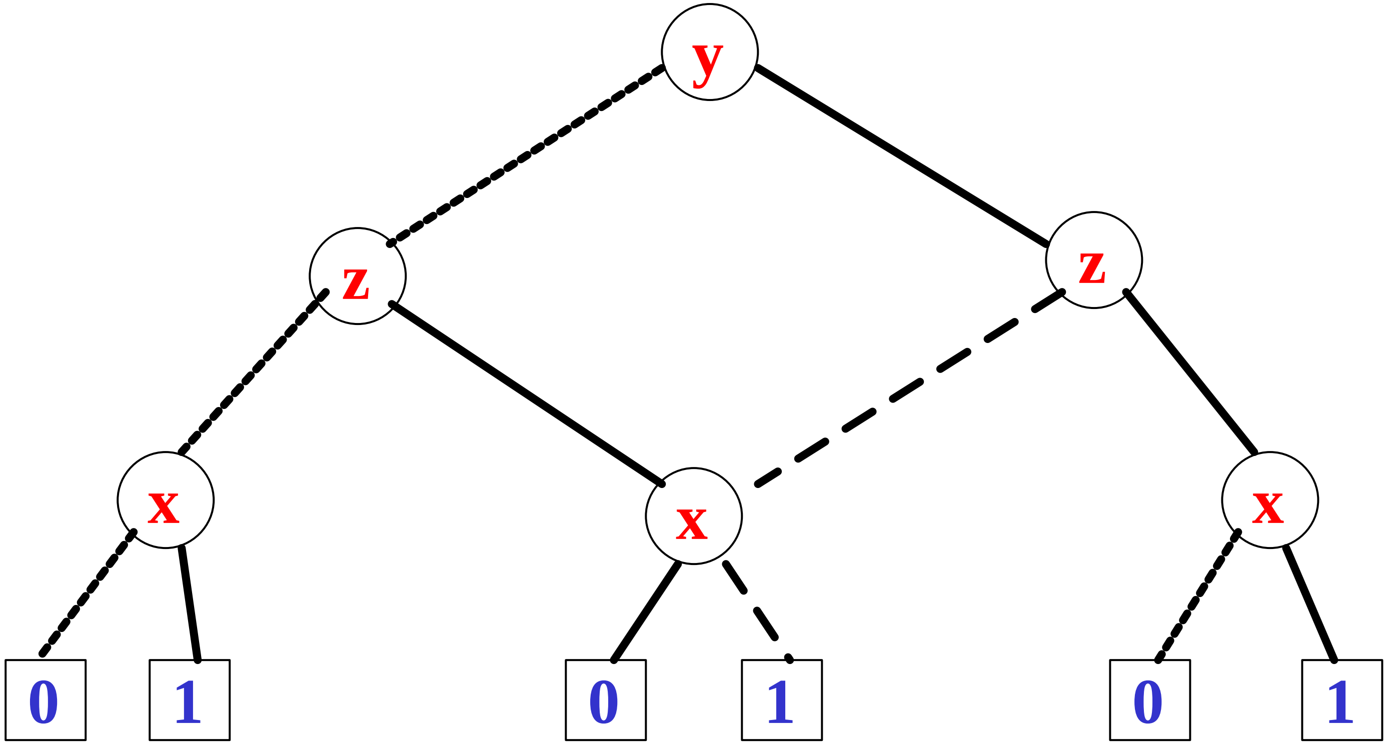
unique node

Share identical non-terminal nodes. (R3)



g = (y Ù (x Û z)) Ú (Øy Ù (x Û Øz))

Share identical non-terminal nodes. (R3)



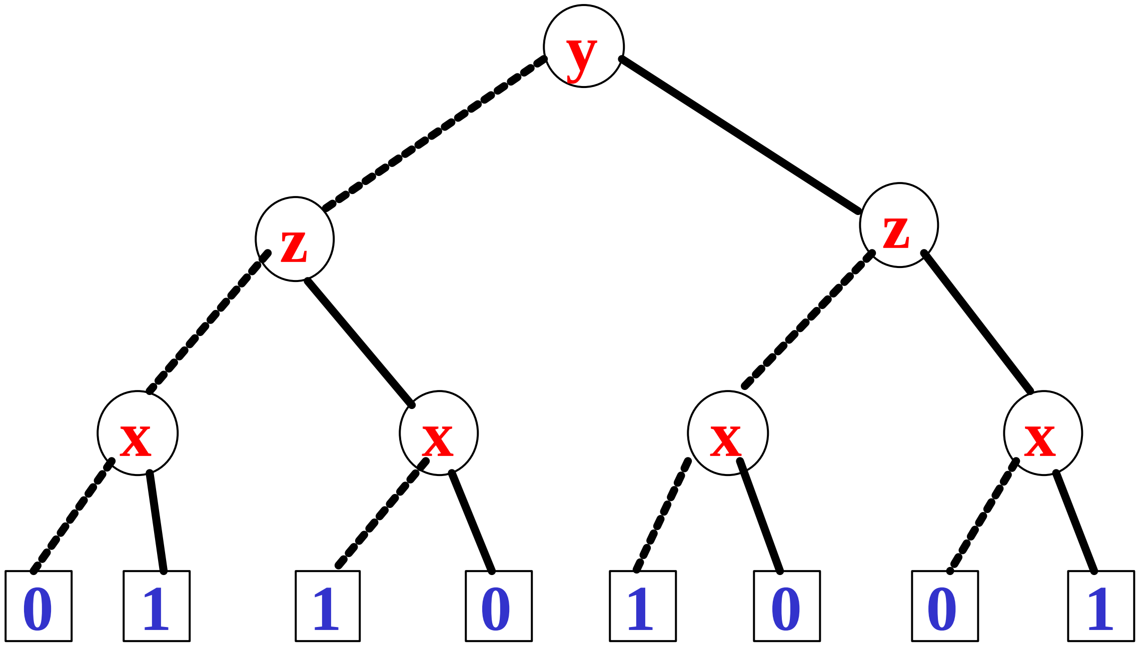
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Reduced BDDs

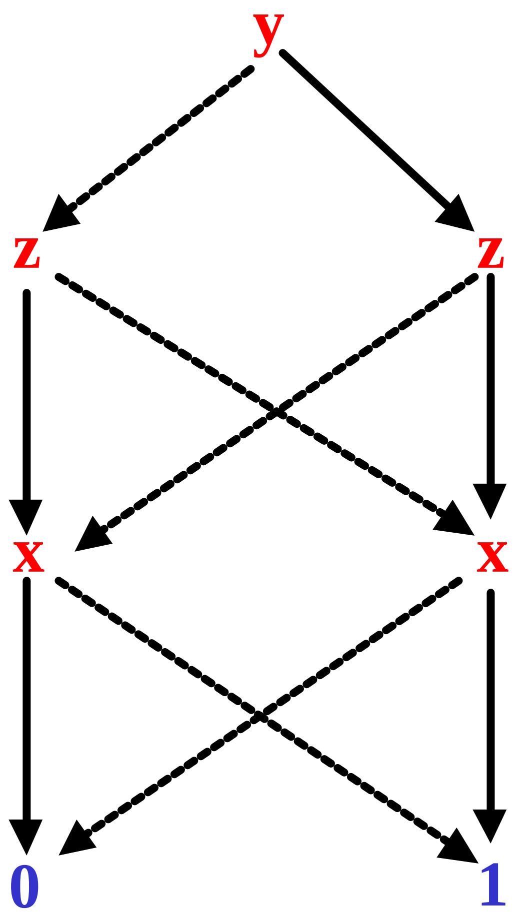
- A **BDD** is ***reduced*** ***iff*** none of the three reduction rules can be applied to it.
- Start from the bottom layer (terminal nodes).
- ***Apply*** the ***rules*** repeatedly ***to level*** **i**. And then ***move to level*** **i-1** (in this way checking for applicability of R3 only needs testing 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 Ù (x Û z)) Ú (Øy Ù (x Û Øz))

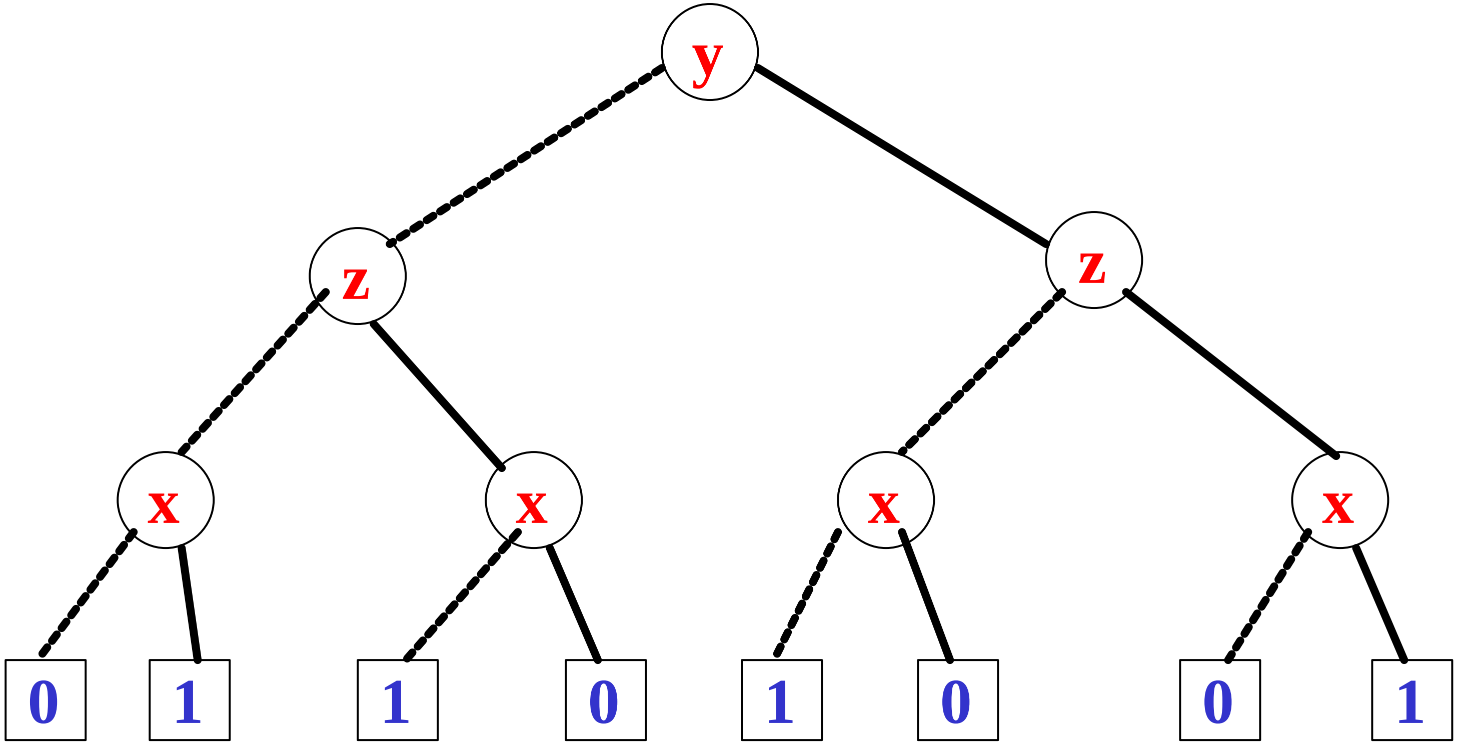
Ordered BDDs

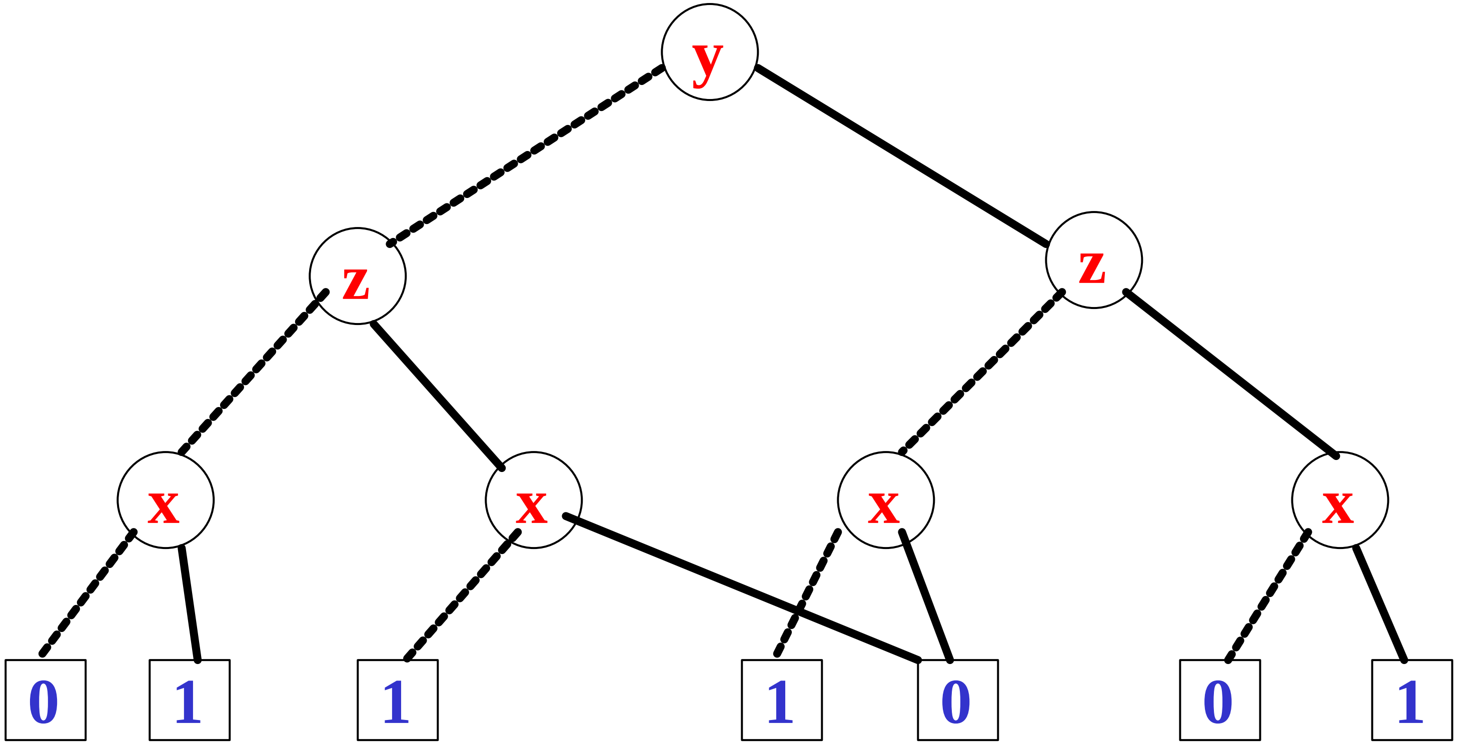
- **$\{x_1, x_2, \ldots, x_n\}$**
  - An indexed (ordered) set of boolean variables.
  - **$x_1 < x_2 \ldots\ldots < x_n$**
- **G** is an **ordered BDD** w.r.t. the above ***variable ordering iff***:
  - Each variable that appears in **G** is in the above set. (but the converse may not be true).
  -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

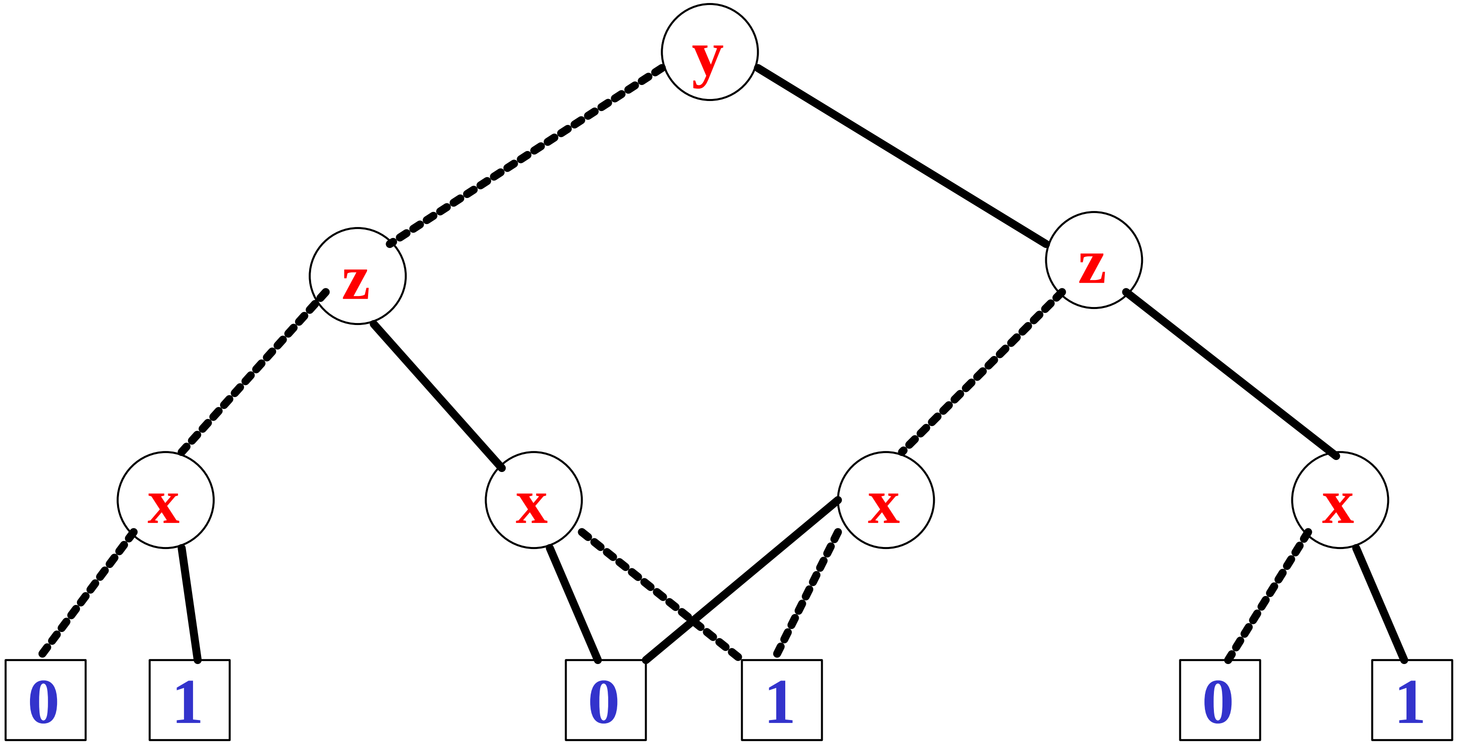
Ordered BDDS

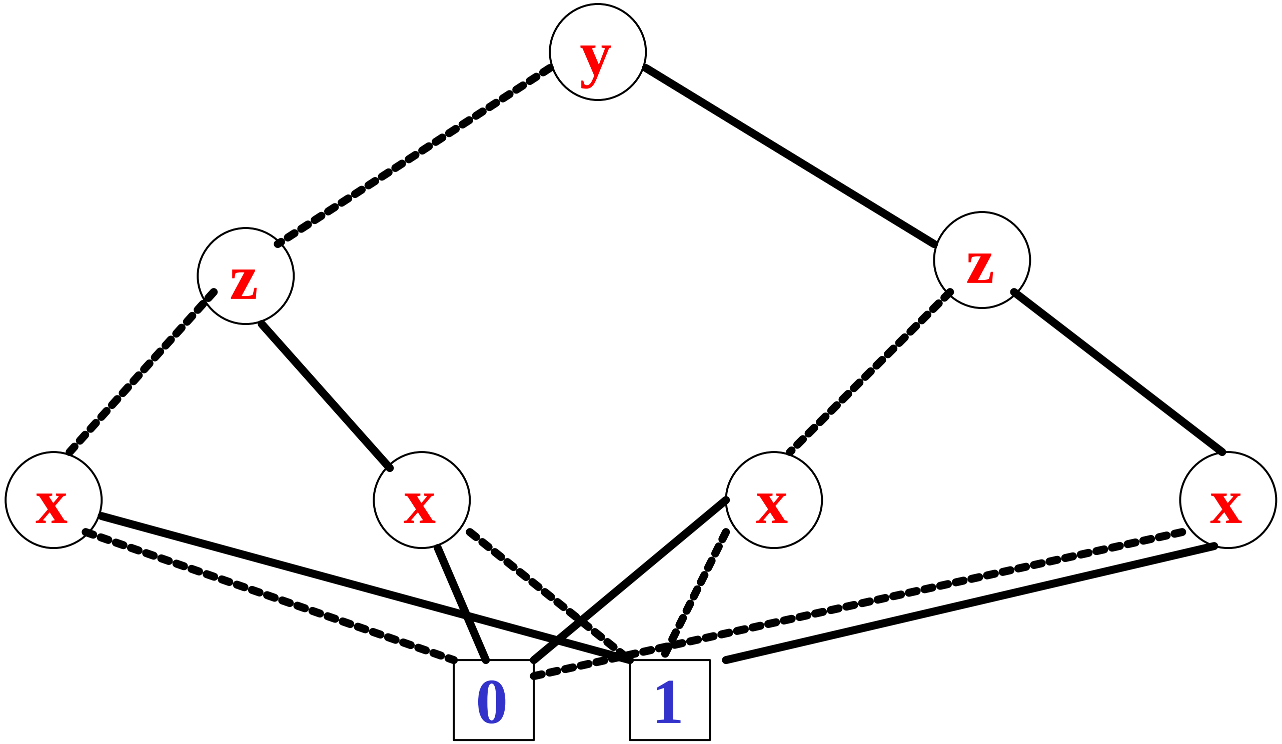
- Fundamental Fact:
  - For a fixed variable ordering, each boolean function has ***exactly one*** reduced Ordered BDD!
  - Reduced OBDDs are ***canonical objects***.
  - To test if ***f*** and ***g*** are equal, we just have to check if **their** reduced **OBDD**s are **identical**.
  - This will be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

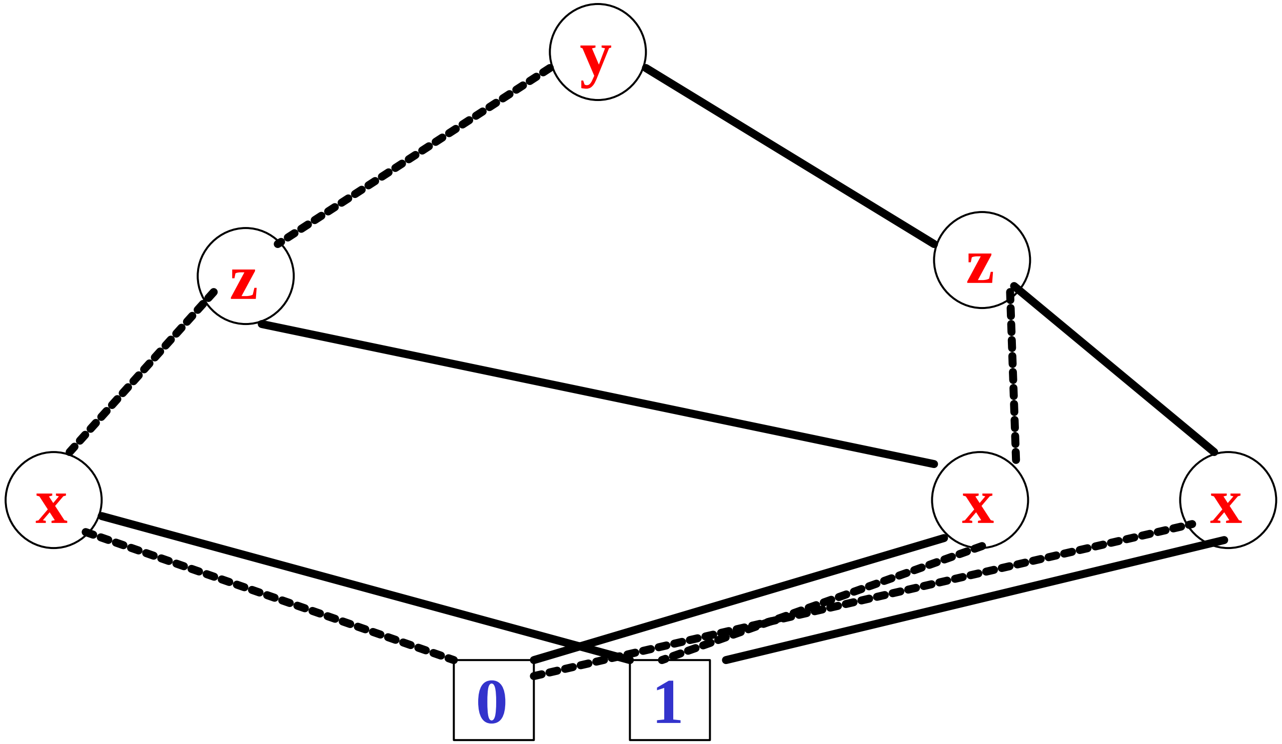
y < z < x

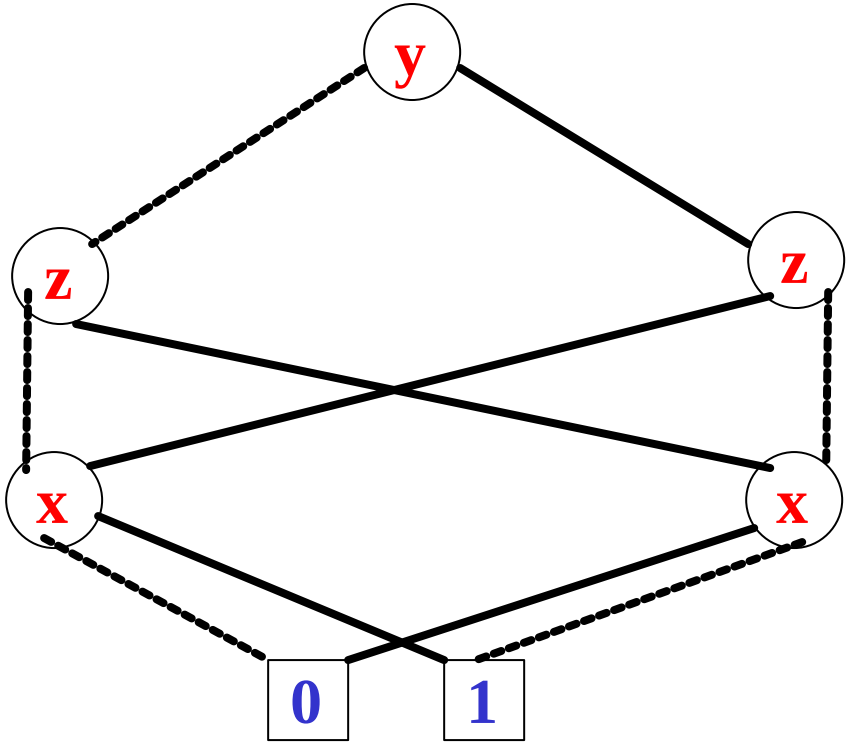



y
z
z
x
x
x
x
0
1
1
1
0
0
1


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

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

Therefore we can say that:

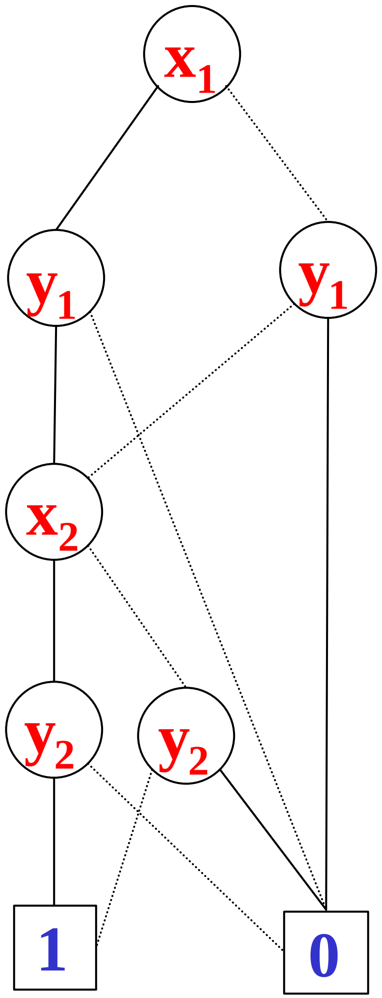
- A function f^u is a ***tautology*** if its **ROBDD** ***u*** is ***equal*** to **1**.
- A function f^u is a ***satisfiable*** if its **ROBDD** ***u*** is ***not equal*** to **0**.

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*** $(x_1 = y_1$ Ù $x_2 = y_2)$
- 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 $3{\cdot}2^2 - 1 = 11$!

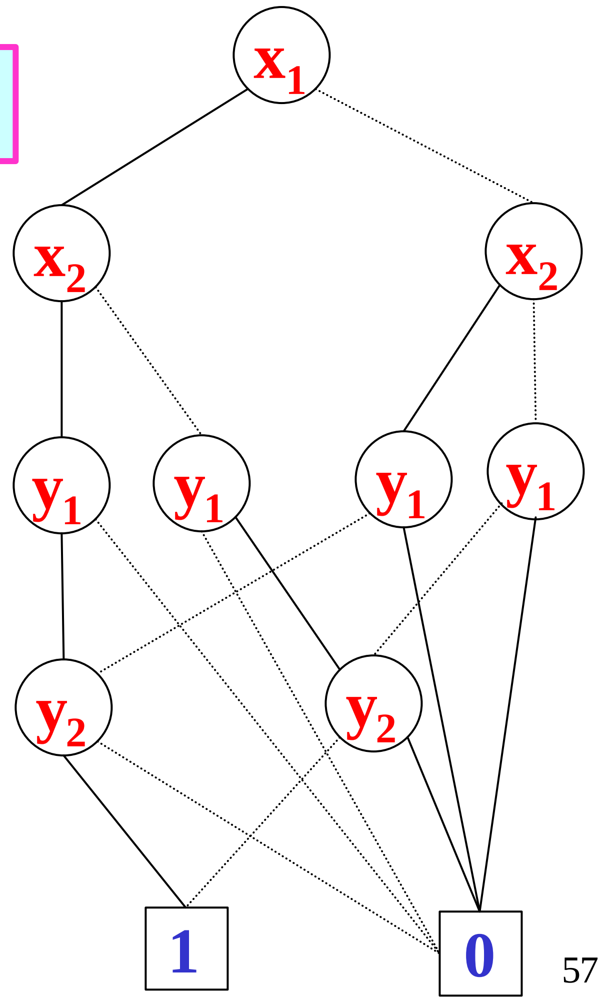
Reduced OBDDs

$x_1 < y_1 < x_2 < y_2$



$(x_1 = y_1$ Ù $x_2 = y_2)$

$x_1 < x_2 < y_1 < y_2$



Reduced OBDDs

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

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

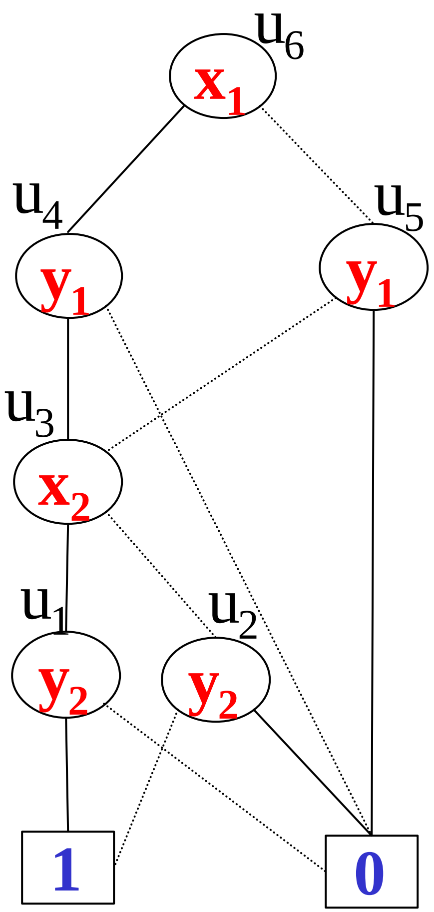
  – $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = 1 \quad \textit{iff} \quad \bigwedge_{i=1}^{n} (x_i = y_i)$
- If $x_1 < y_1 < x_2 < y_2 \ldots < x_n < y_n$, then the **OBDD** is of size $3n + 2$.
- If $x_1 < x_2 < \ldots < x_n < y_1 < \ldots < y_n$, then the **OBDD** is of size $3 \,.\, 2^n - 1$!

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$_i$***, ***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** Ú **g** --------- $\mathbf{G_f}$ $\mathbf{op}_{\text{Ú}}$ $\mathbf{G_g} = \mathbf{G}_{\mathbf{f}\,\text{Ú}\,\mathbf{g}}$
- 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***.
- $\mathbf{X = X_G \; [\; X_{G'}}$
- There is an ordering $<$ on **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):
- **$f(x_1, x_2, \ldots, x_n)$**
  - **$fj_{x_1 = 0} = f(0, x_2,\ldots,x_n)$**
    - **$f = x_1$ Ç $(x_2$ Æ $x_3)$**
    - **$fj_{x_1 = 0} = x_2$ Ù x_3**
  - Similarly, **$f\ j_{x1 = 1} = f(1, x_2,\ldots,x_n)$**

$f(x_1, x_2,\ldots,x_n) = ($Ø$x_1$ Ù $fj_{x_1 = 0})$ Ú $(x_1$ Ù $fj_{x_1 = 1})$

- This is true even if **x_1** does not appear in **f** !

Operations on OBDDs: Negation

- The basic idea (Shannon Expansion):

$$\boxed{f(x_1, x_2,\ldots,x_n) = (\text{Ø}x_1 \text{ Ù } fj_{x_1 = 0}) \text{ Ú } (x_1 \text{ Ù } fj_{x_1 = 1})}$$

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

$$\text{Ø } f(x_1, x_2,\ldots,x_n) = \text{Ø } ((\text{Ø}x_1 \text{ Ù } fj_{x_1 = 0}) \text{ Ú } (x_1 \text{ Ù } fj_{x_1 = 1}))$$

$$= (\text{Ø}(\text{Ø}x_1 \text{ Ù } fj_{x_1 = 0}) \text{ Ù } \text{Ø}(x_1 \text{ Ù } fj_{x_1 = 1}))$$

$$= ((x_1 \text{ Ú Ø } fj_{x_1 = 0}) \text{ Ù } (\text{Ø } x_1 \text{ Ú Ø } fj_{x_1 = 1})$$

$$= (x_1 \text{ Ù Ø } x_1) \text{ Ú } (\text{Ø } x_1 \text{ Ù Ø } fj_{x_1 = 0}) \text{ Ú}$$

$$\text{Ú } (x_1 \text{ Ù Ø } fj_{x_1 = 1}) \text{ Ú } (\text{Ø } fj_{x_1 = 0} \text{ Ù Ø } fj_{x_1 = 1})$$

$$= (\text{Ø } x_1 \text{ Ù Ø } fj_{x_1 = 0}) \text{ Ú } (x_1 \text{ Ù Ø } fj_{x_1 = 1})$$

Operations on ROBDDs.

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

  CASE1: $x = y$

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

  – 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?).

  – **g** $\text{j}_{x= 0}$ **= g = g** $\text{j}_{x= 1}$

  ▪ **f op g = (Øx Ù (f j**$_{x = 0}$ **op g) Ú (x Ù (f j**$_{x = 1}$ **op g)**

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

  CASE2: x > y is symmetric.

Operations on ROBDDs.

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

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

  Eg.: if **op =** Ú then {0,0 ® **0**; **1**}

  if **op =** Ù then {1,1 ® **1**; **0**}

  ….

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)
```

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

n = number of variables.

Efficient algorithm for Apply

Algorithm Apply(op,u,v)

init(G_{op})

Function App(u,v)

if $G_{op}(u,v)$ ^{1} 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

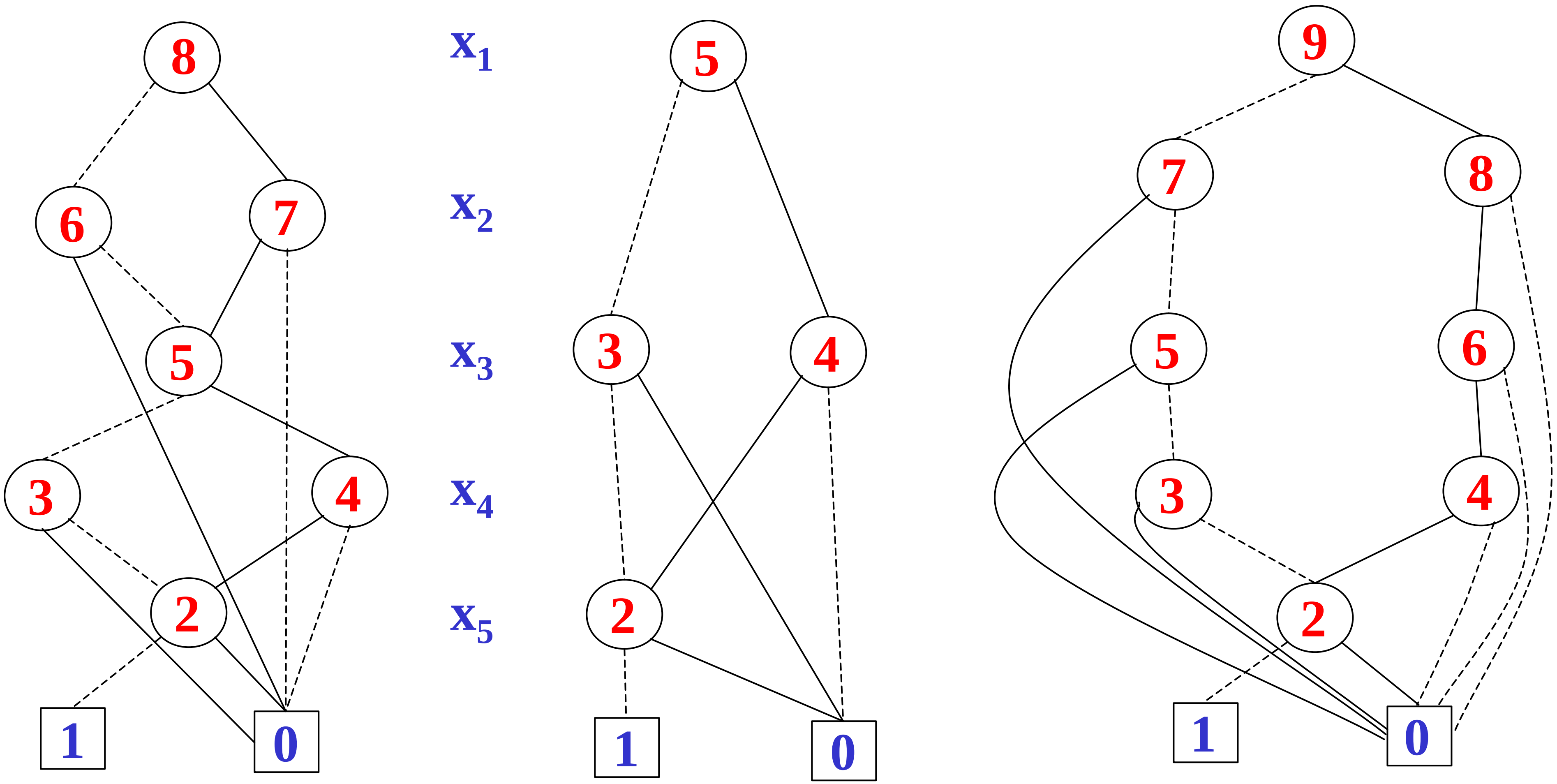
return r

return App(u,v)

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

Exemple of Apply Æ

(x_1 o x_2) Ù(x_3 o x_4) Ù Øx_5 Ù (x_1 o x_3) Ù Øx_5 = ((x_1Ùx_2Ùx_3Ùx_4) Ú Ú (Øx_1ÙØx_2ÙØx_3ÙØx_4)) ÙØx_5



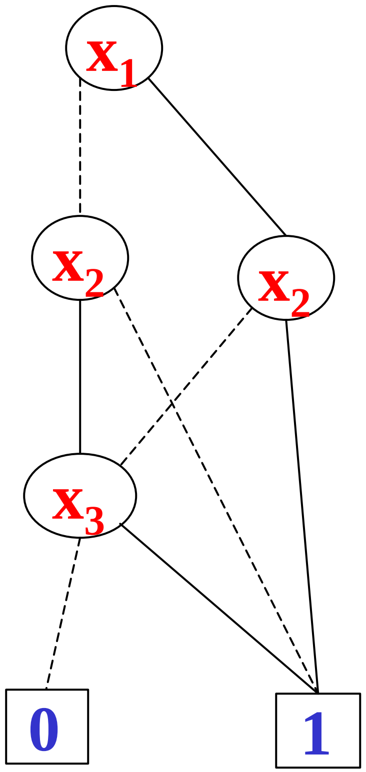
The Restrict operation

- ***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 the assignment.
- E.G.: if $f(x_1,x_2,x_3) = ((x_1$ Û $x_2)$ Ú $x_3)$ we want to compute $f(x_1,x_2,x_3)[0/x_2] = f(x_1,0,x_3)$

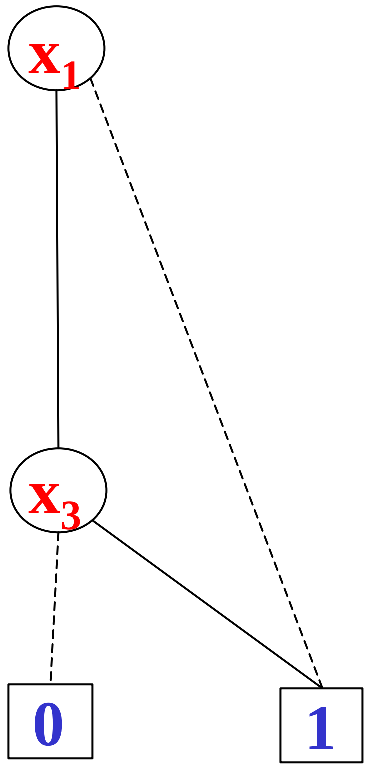
  i.e.: $f(x_1,0,x_3)=$ Øx_1 Ú x_3

Restrict Operation: example

$f(x_1,x_2,x_3) = ((x_1$ Û $x_2)$ Ú $x_3)$



$f(x_1,x_2,x_3)[0/x_2] =$ Øx_1 Ú 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

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

- 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

$x.t | " x.t

- They can be defined in terms of ROBDD operations:

$x.t = t[0/x] Ú t[1/x]

" x.t = t[0/x] Ù 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 labeling.
- **$AP = \{p, q, r\}$**
- **$L(s) = \{p\}$**
  - **f_s = p** Ù **Øq** Ù **Ør**
- **$f_{\{s_1, s_2, s_5\}}$ = f_{s_1}** Ú **f_{s_2}** Ú **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 $\mathbf{f}_{\mathbf{R(s_1,s_2)}}$ = $\mathbf{f}_{\mathbf{s_1}}$ Ù $\mathbf{f'}_{\mathbf{s_2}}$.
  – where $\mathbf{f}_{s_1}$ = **p** Ù **q** Ù Ø**r** and $\mathbf{f'}_{s_2}$ = Ø**p’** Ù Ø**q’** Ù **r’**
- $\mathbf{f}_{\mathbf{R}}$ = $\acute{\text{Ú}}_{(s_1, s_2)\ 2\ R}$ ($\mathbf{f}_{\mathbf{R(s_1,s_2)}}$)
- **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)
- \|[Ø f]\| = Ø$f_f(x_1,\ldots,x_n)$

  (apply negation to the OBDD for f)
- \|[f Ú y]\| = $f_f(x_1,\ldots,x_n)$ Ú $f_y(x_1,\ldots,x_n)$

  (apply Ú operation to the OBDDs for f and y)
- \|[f Ù y]\| = $f_f(x_1,\ldots,x_n)$ Ù $f_y(x_1,\ldots,x_n)$

  (apply Ù operation to the OBDDs for f and y)

CTL Symbolic Model Checking

- ||[EX f]|| =

  $\$x'_{1},\ldots,x'_{n}(f_{f}(x'_{1},\ldots,x'_{n})\ \text{Ù}\ f_{R}(x_{1},\ldots,x_{n},x'_{1},\ldots,x'_{n}))$

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

- ||[EU(f,y)]|| = $\text{m}Z.(f_{y}(x_{1},\ldots,x_{n})\ \text{Ú}$
  $(f_{f}(x_{1},\ldots,x_{n})\ \text{Ù}\ \text{EX}\ Z))$

- ||[EG f]|| = $\text{n}Z.(f_{f}(x_{1},\ldots,x_{n})\ \text{Ù}\ \text{EX}\ Z)$

Symbolic model checking: example

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

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

   $\mathbf{Y_i} = \mathbf{\$x'_1,\ldots,x'_n(f_y (x'_1,\ldots,x'_n)}$ Ù $\mathbf{f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n))}$

   $\mathbf{X_{i+1}}$ can easily be computed as $\mathbf{X_i}$ Ù $\mathbf{Y_i}$
4. Check **whether** $\mathbf{X_{i+1} = X_i}$ 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)

$\mathbf{f}_1(x) := \mathbf{f}_b(x);$

j=1;

repeat

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

$\mathbf{f}_j := \mathbf{f}_b(x)\ \text{Ù}\ \$x'.(\ \mathbf{f}_R(x, x')\ \text{Ù}\ \mathbf{f}_{j-1}(x')\ );$

until $\mathbf{f}_j(x) = \mathbf{f}_{j-1}(x);$

Algorithm Compute_EU(β_1,β_2)

$\mathbf{f}_1(x) := \mathbf{f}_{b_2}(x);$

j=1;

repeat

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

$\mathbf{f}_j := \mathbf{f}_{b_2}(x)\ \text{Ú}\ (\mathbf{f}_{b_1}(x)\ \text{Ù}\ \$x'.(\ \mathbf{f}_R(x, x')\ \text{Ù}\ \mathbf{f}_{j-1}(x')\ )\ );$

until $\mathbf{f}_j(x) = \mathbf{f}_{j-1}(x);$

CTL Symbolic model checking

Finally, assuming boolean variable **$V=\{x_1,\ldots,x_n\}$**, and the ROBDD for **|[** f **]|** already computed.

- Checking whether

  K ² f

  amounts to checking whether the ROBDD for **f_{Init}** Ù **f**$_{\text{Ø f}}$ is **identical** to the ROBDD for **0**, where **f_{Init}** is the ROBDD for the set **Init** of **initial states** of **K**.

  (remember that **K** ² f iff **Init** Í **|[** f **]|**)

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**).