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.
- Links will be created soon to these documents.

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

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

Symbolic Model checking

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

- **S'** $\subseteq$ **S** can be represented as a ***boolean function***.
- **R** can be represented as a ***boolean function***.
- **States(ψ)** can be represented as a ***boolean function***.

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

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

Boolean Expressions

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

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

- 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.
- $\mathbf{x} \wedge \mathbf{y}$ represents the function:
  - **f: {0, 1} $\times$ {0, 1} $\rightarrow$ {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}** $\rightarrow$ **{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

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

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

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**	

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

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

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

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

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**.

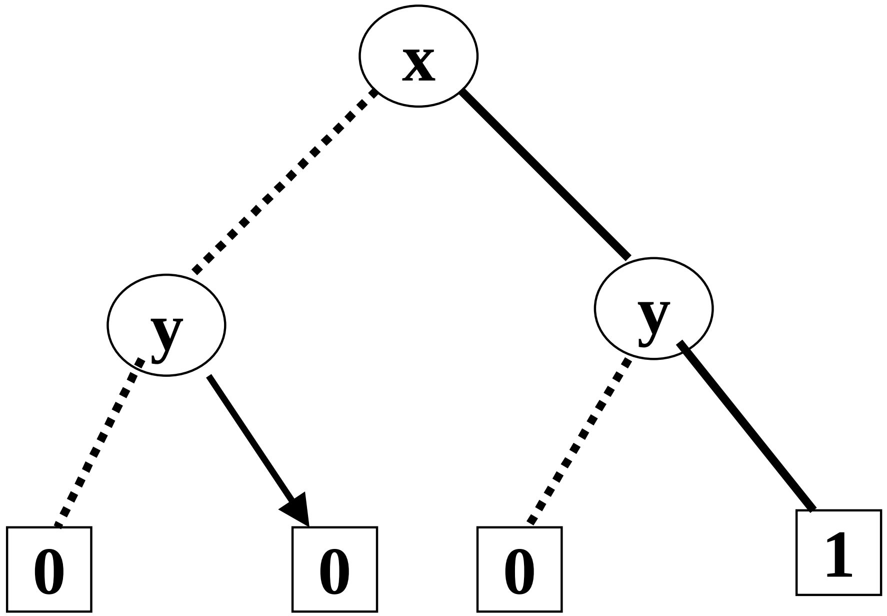
Boolean Decision Diagrams

- 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!

Boolean Decision Tree

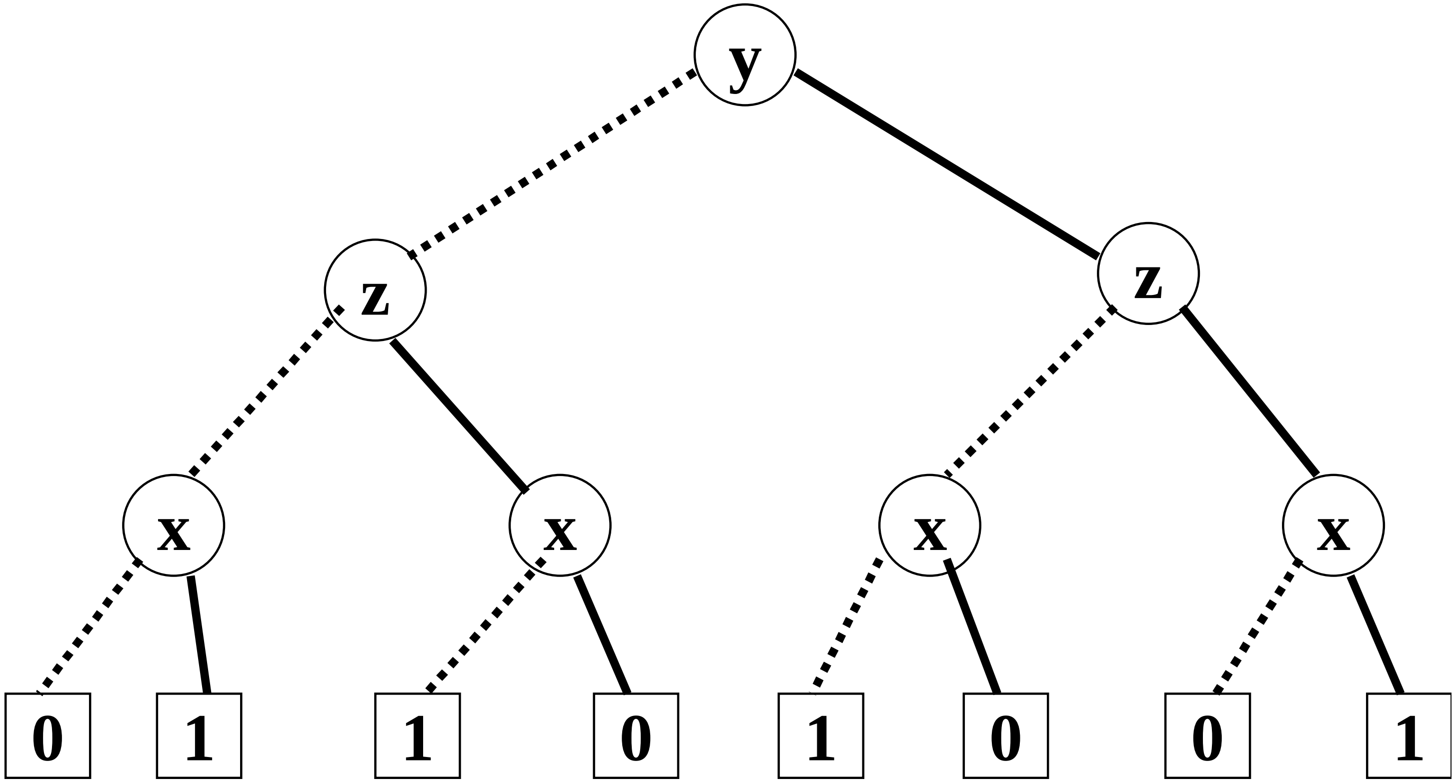
- A ***boolean function*** is represented as a (***binary***) ***tree***.
- Each ***node*** is ***labeled*** with a (boolean) ***variable***.
- Each ***node*** has a ***positive*** (***full line***) and a ***negative*** (***dotted line***) ***successor***.
- The ***terminal nodes*** are labeled with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

Boolean decision trees.



$\mathbf{x \wedge y}$

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

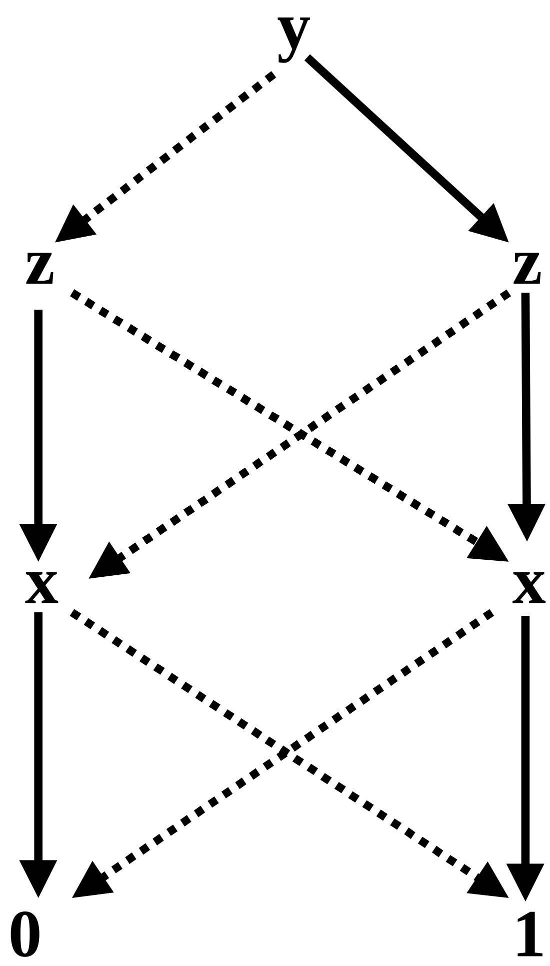


$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

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** (***full edge***, ***high(v)***).



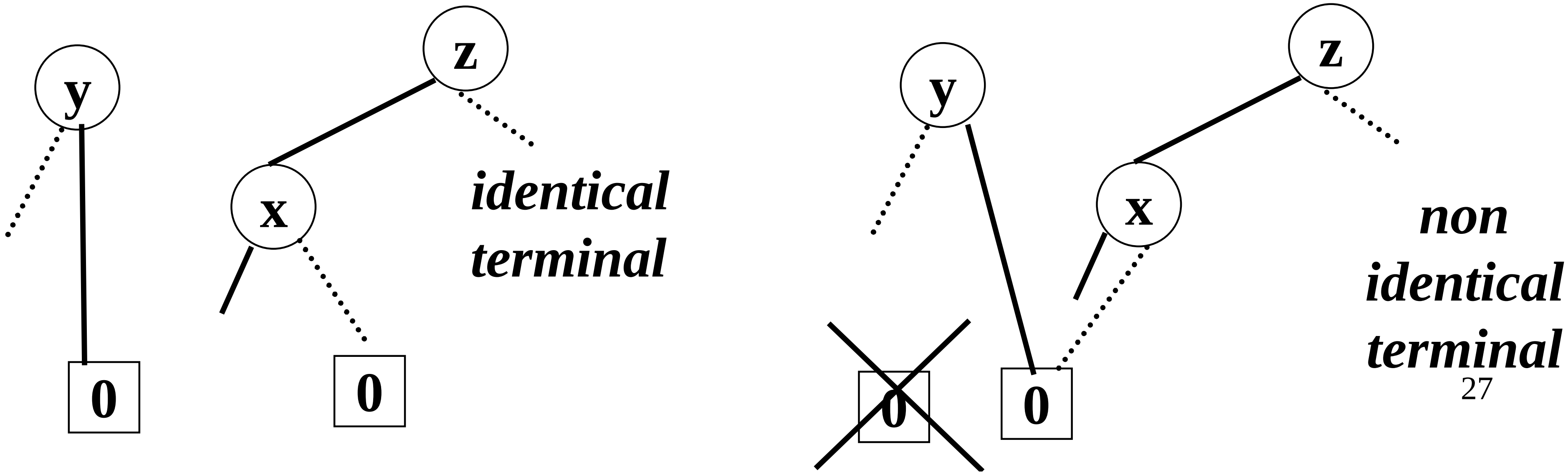
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Reduction Rules

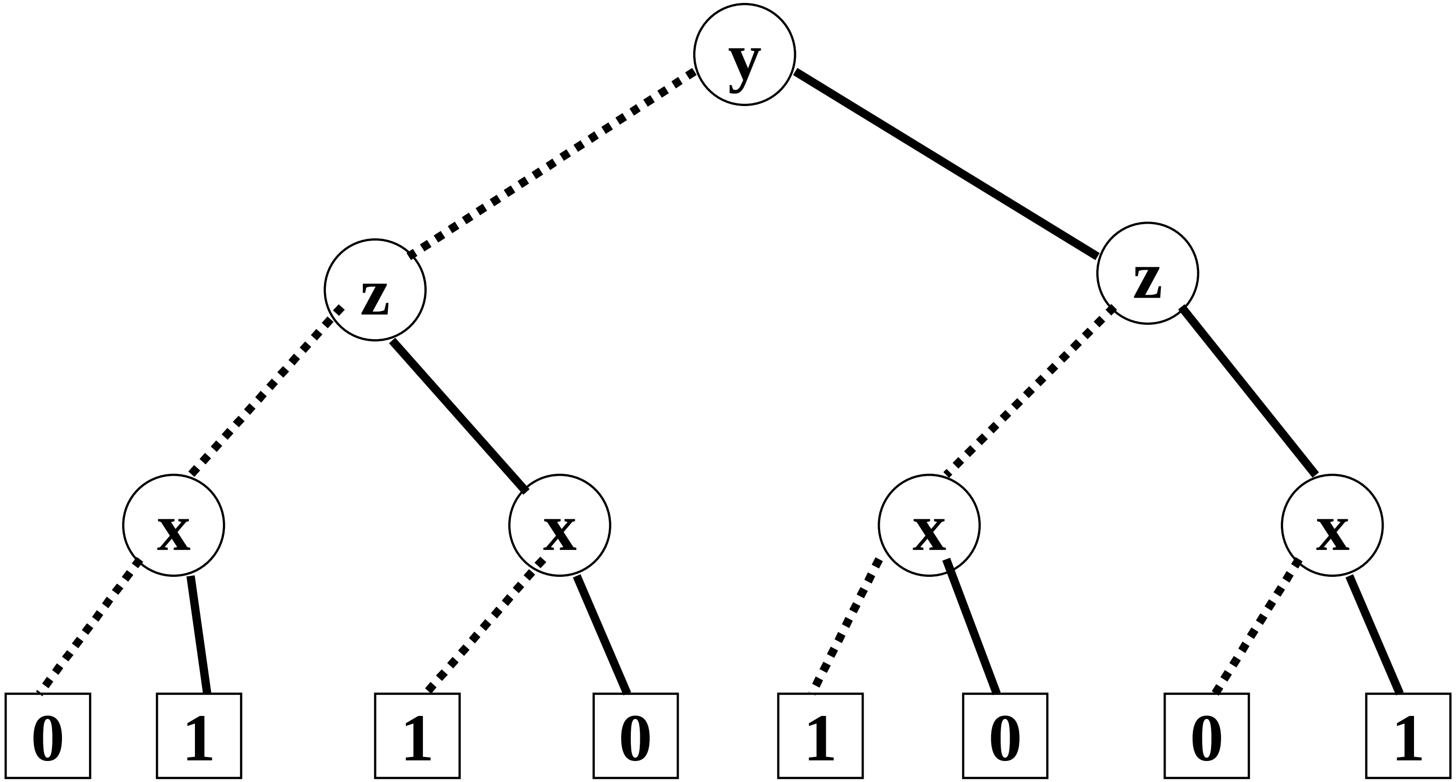
- Three reduction rules:
  - **Share identical terminal nodes. (R1)**
  - **Remove redundant tests (R2)**
  - **Share identical non-terminal nodes. (R3)**

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.**

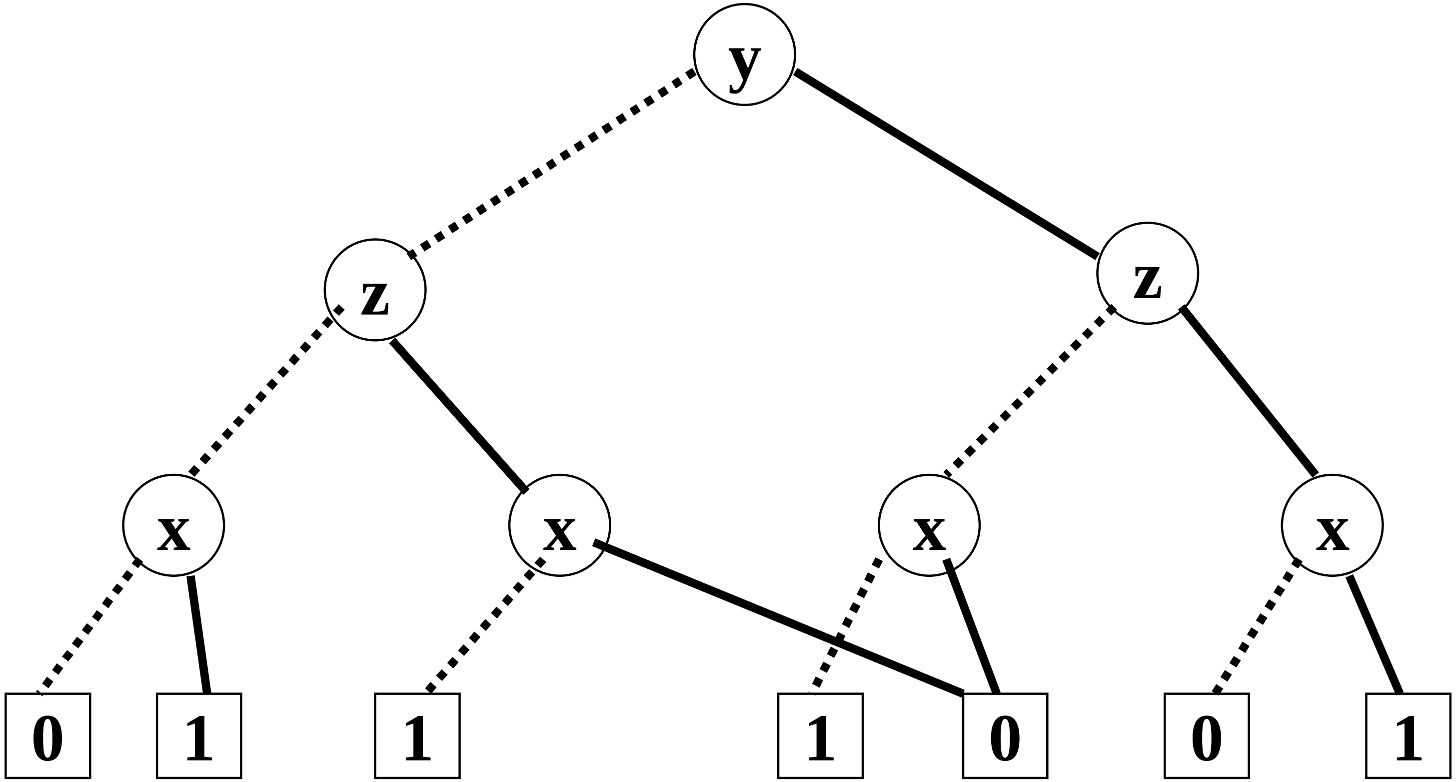


Share identical terminal nodes. (R1)



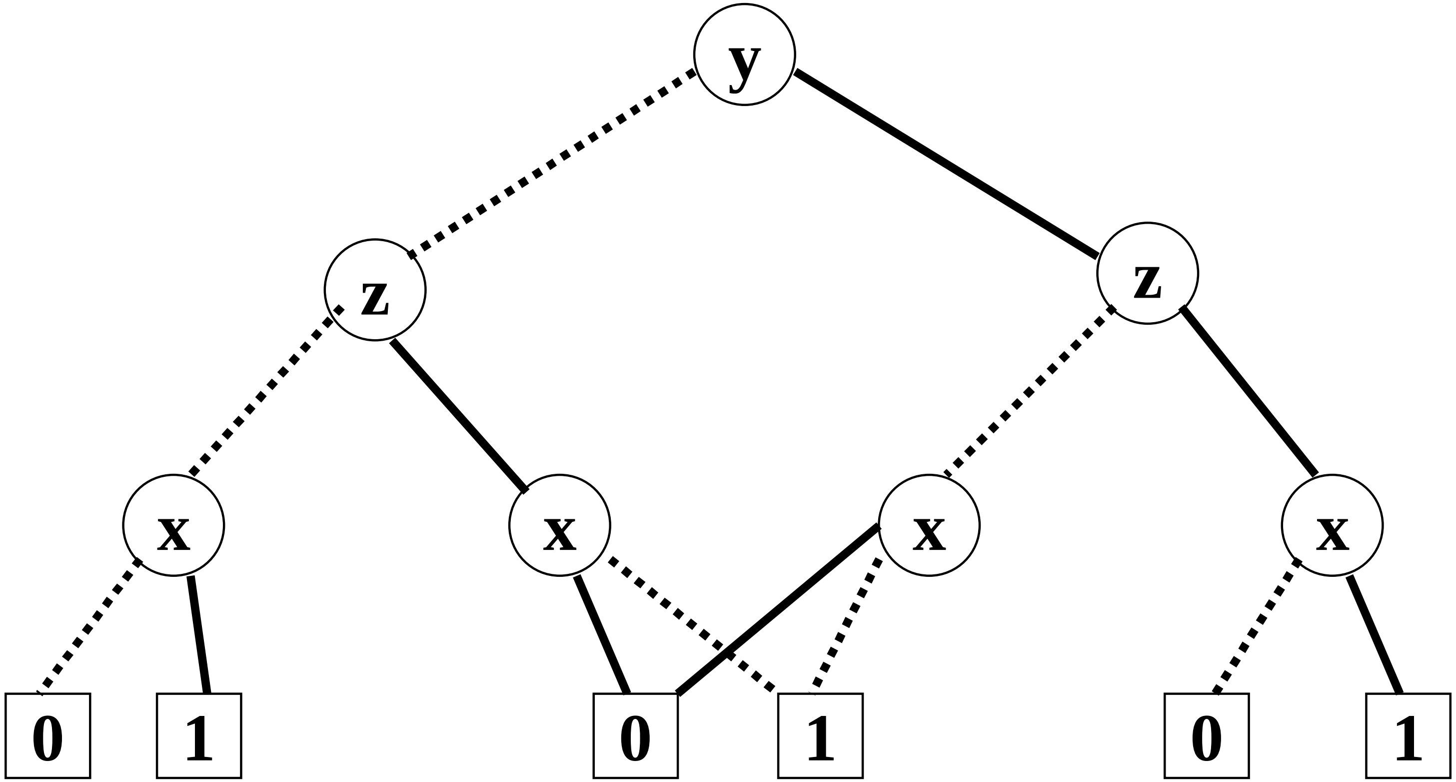
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

Share identical terminal nodes. (R1)



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

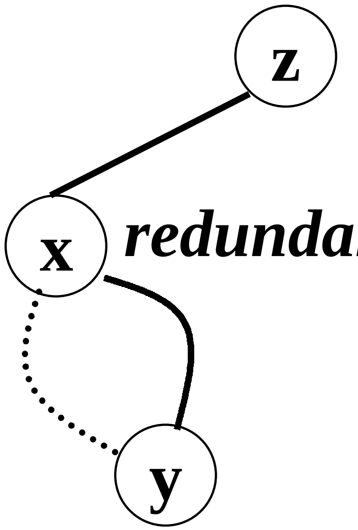
Share identical terminal nodes. (R1)



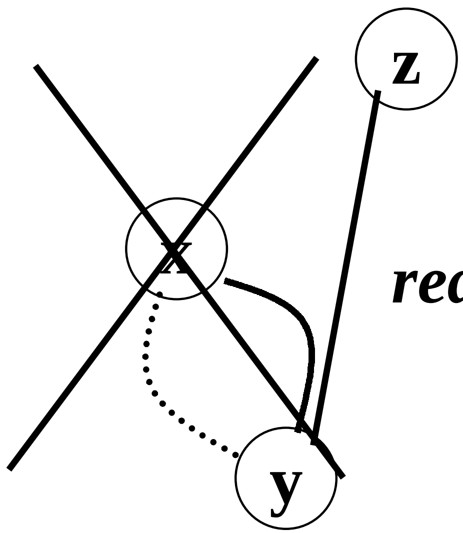
$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

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**.

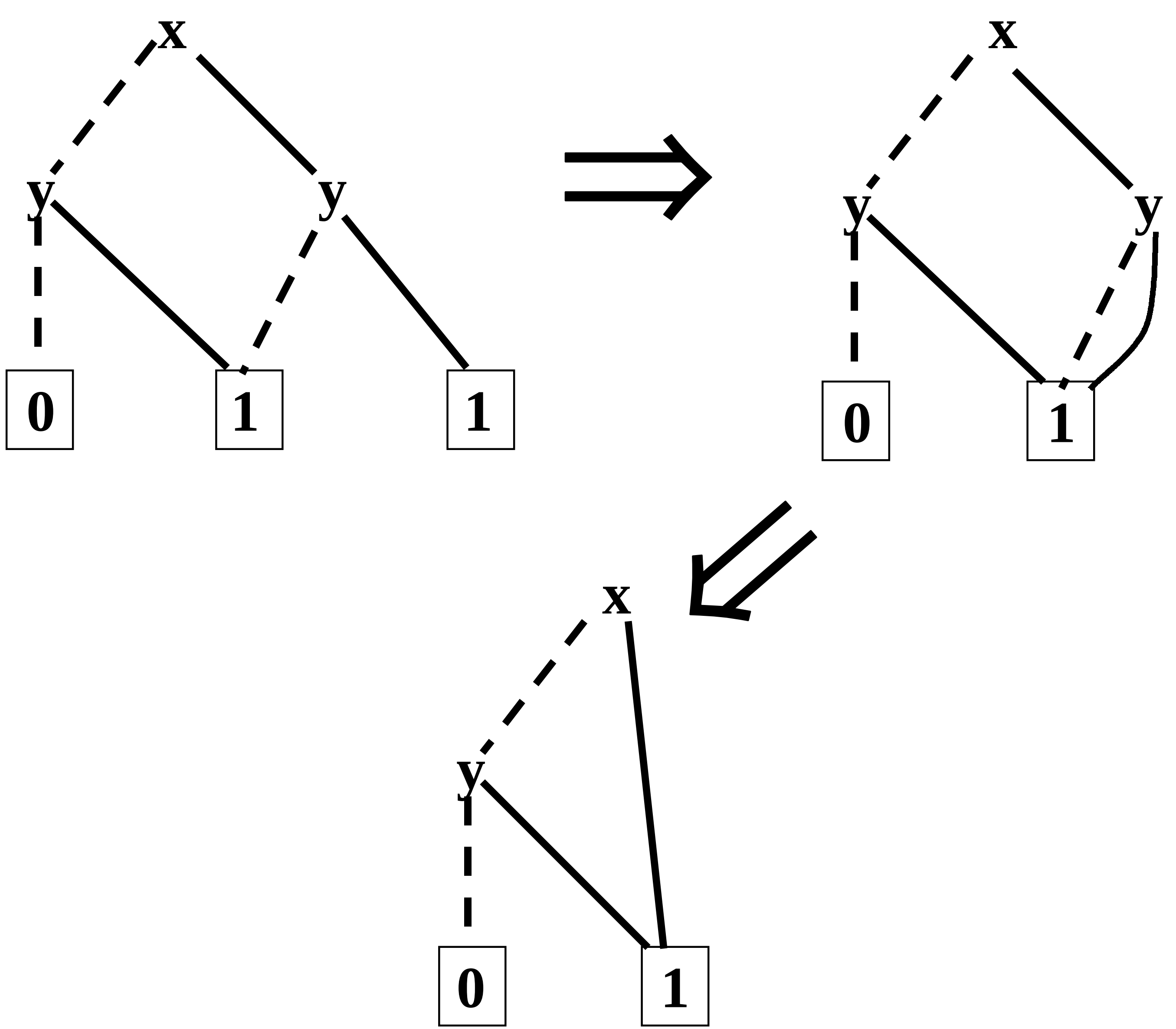


redundant test



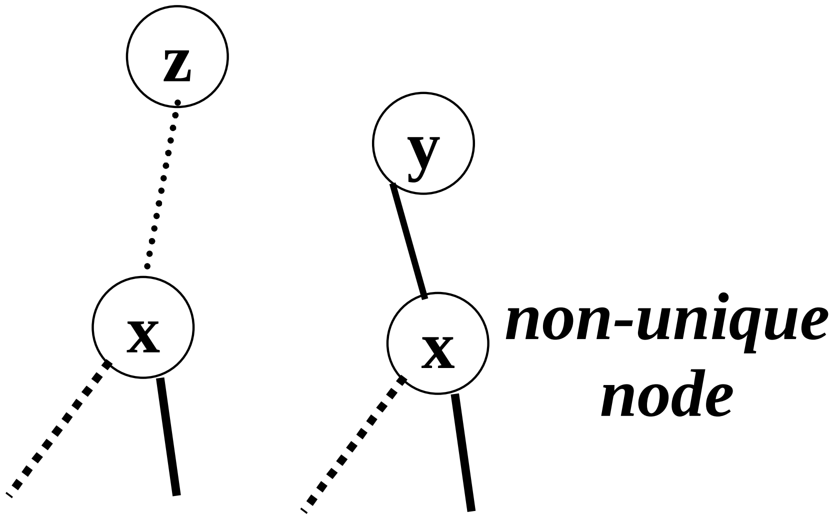
non redundant test

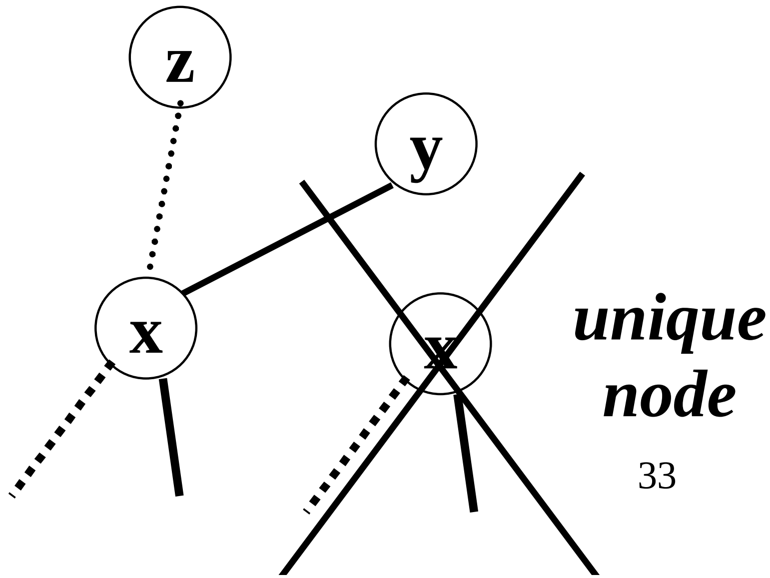
Remove redundant tests (R2)



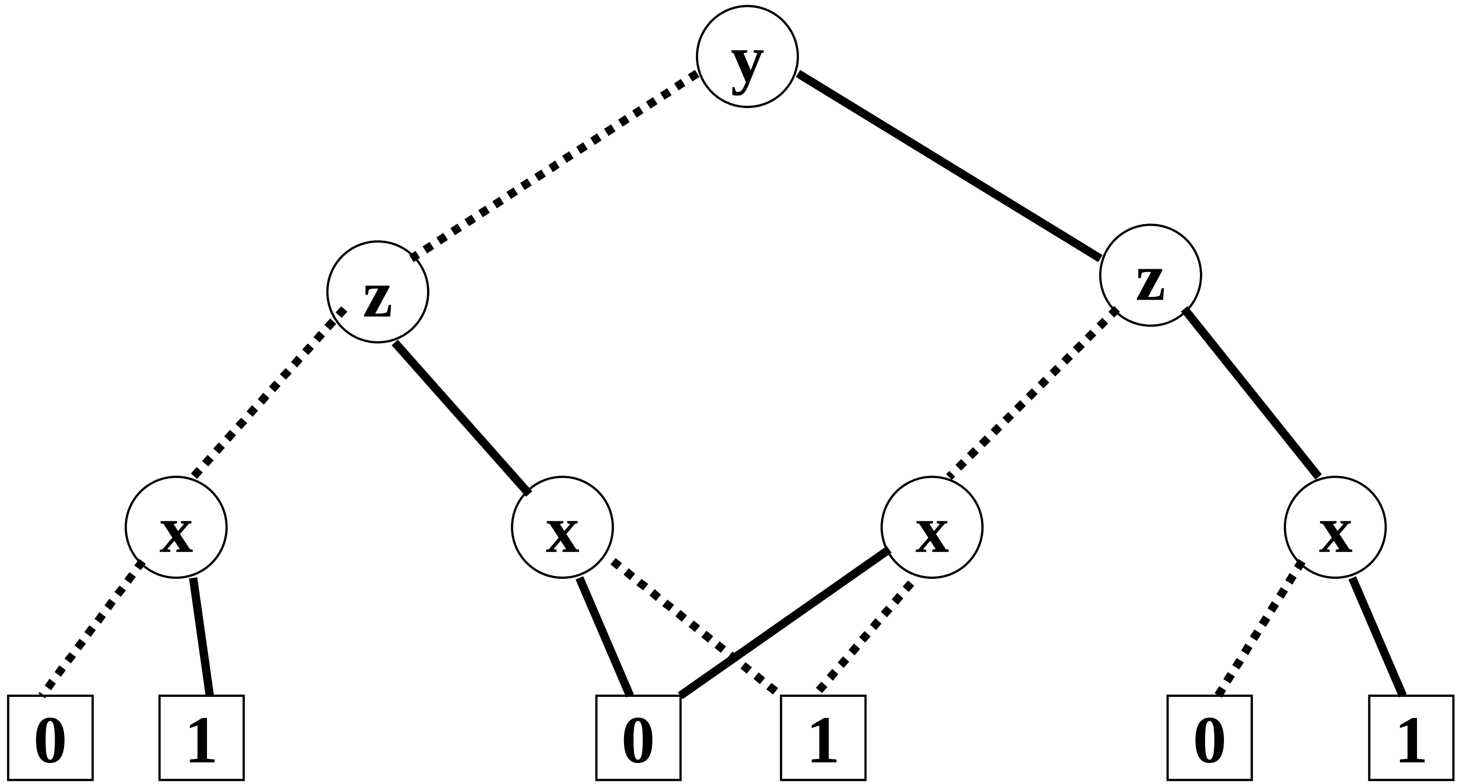
Reduction Rules

- 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*** **n** and ***direct all its incoming edges to*** **m**.



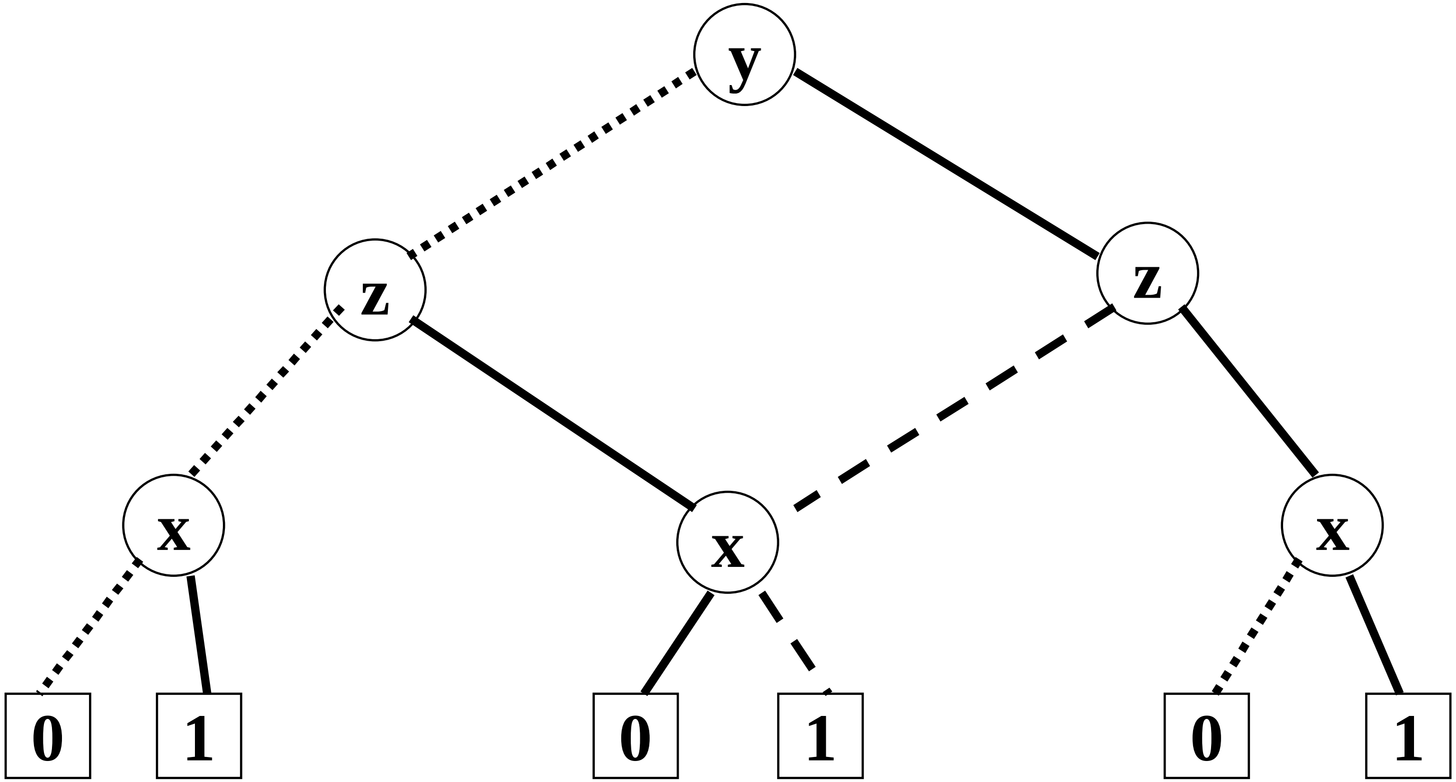


Share identical non-terminal nodes. (R3)



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

Share identical non-terminal nodes. (R3)

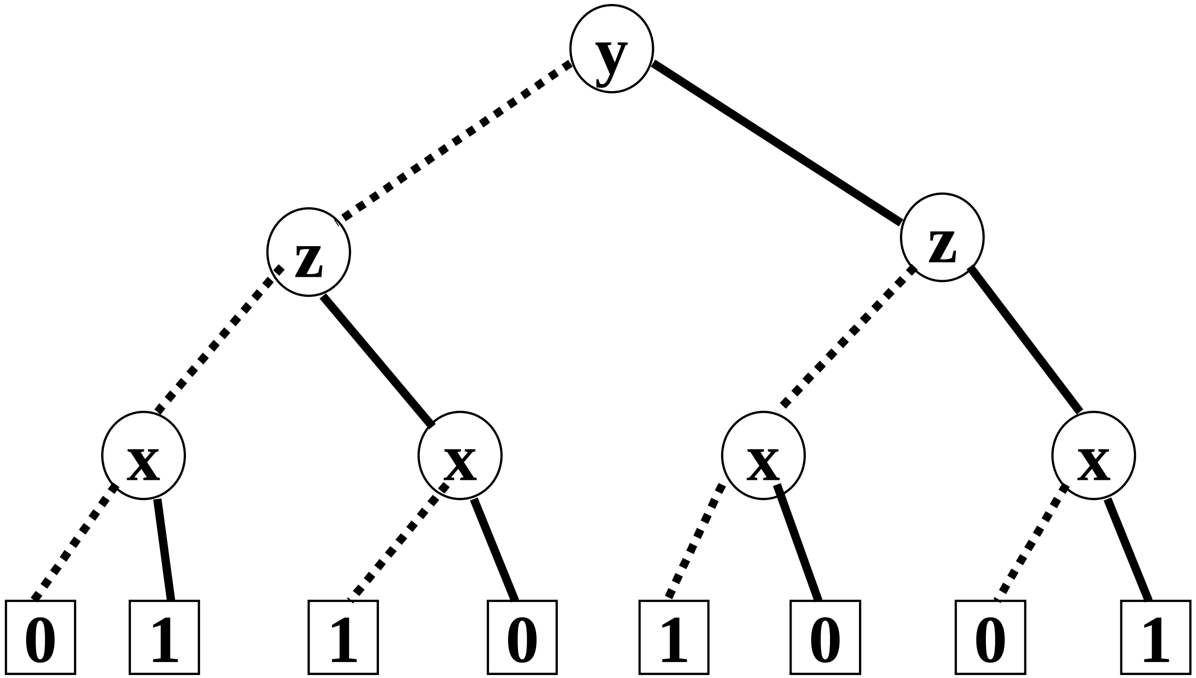


$$g = (y \wedge (x \Leftrightarrow z)) \vee (\neg y \wedge (x \Leftrightarrow \neg z))$$

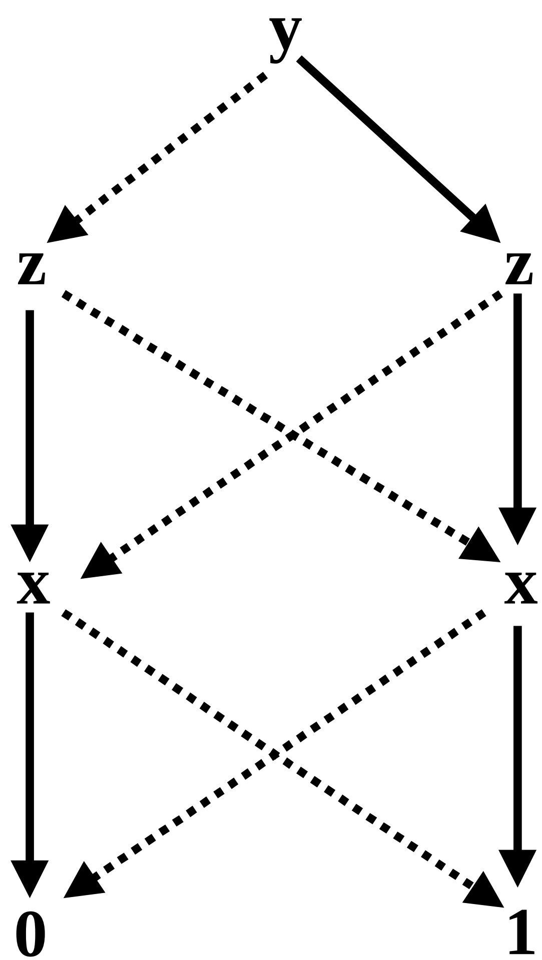
Reduced BDDs

- 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**.
- Stop when the root node has been treated.
- This can be done efficiently.

Binary Decision Tree for



Reduced BDD



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

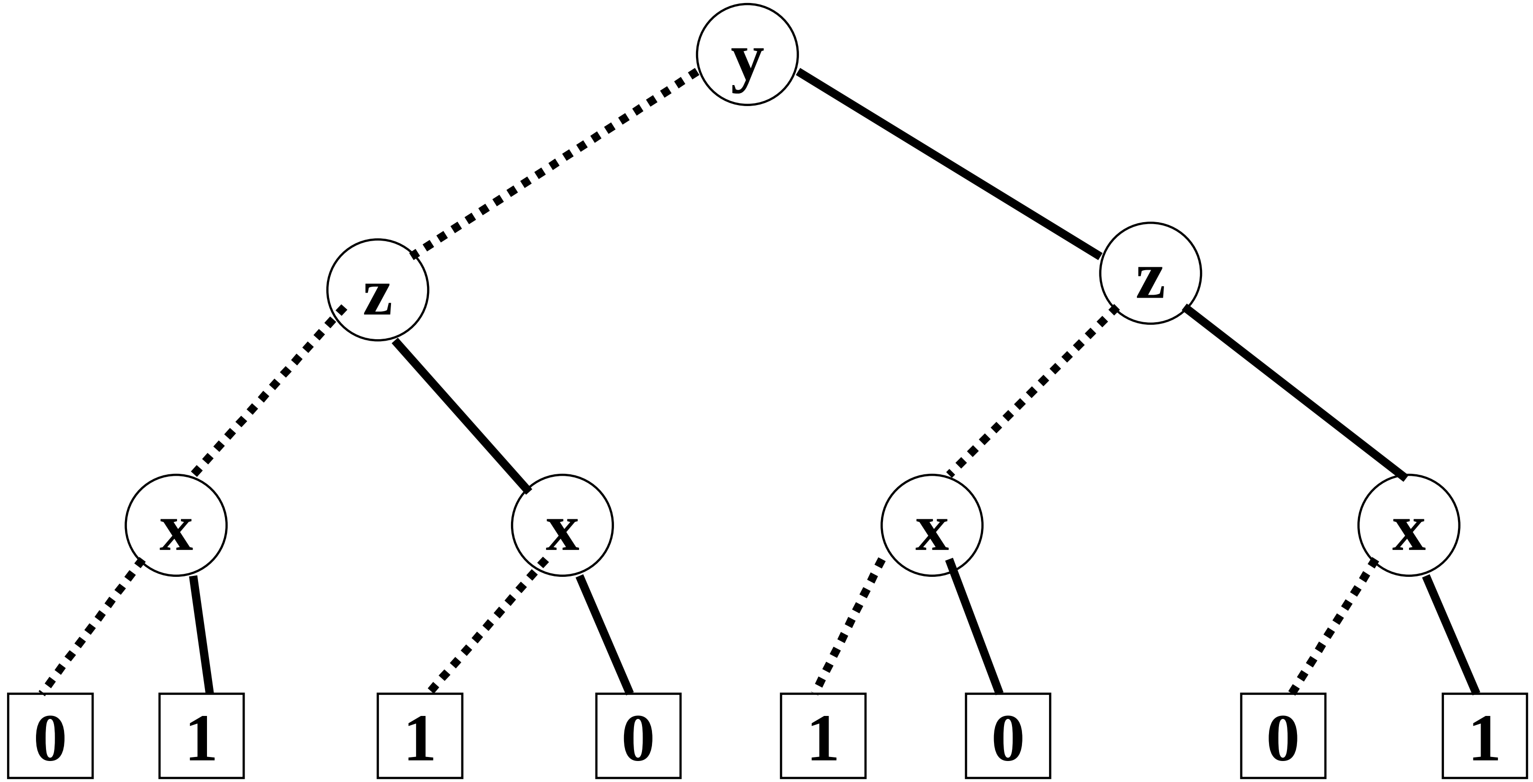
Ordered BDDs

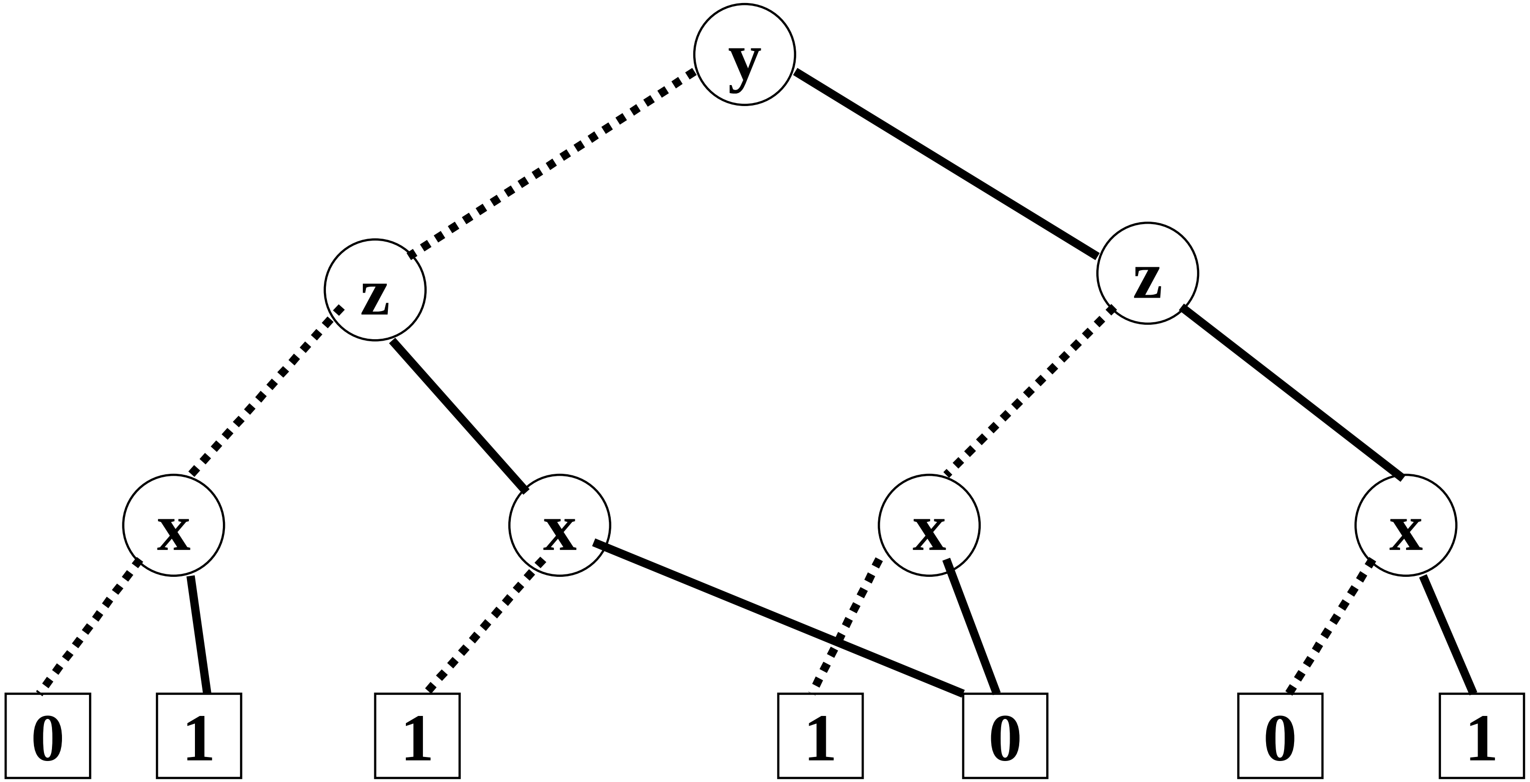
- $\{x_1, x_2, \ldots, x_n\}$
  - An indexed (ordered) set of boolean variables.
  - $x_1 < x_2 \ldots\ldots < x_n$
- **G** is an **ordered BDD** w.r.t. the above ***variable ordering iff***:
  - Each variable that appears in **G** is in the above set. (but the converse may not be true).
  - If $i <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

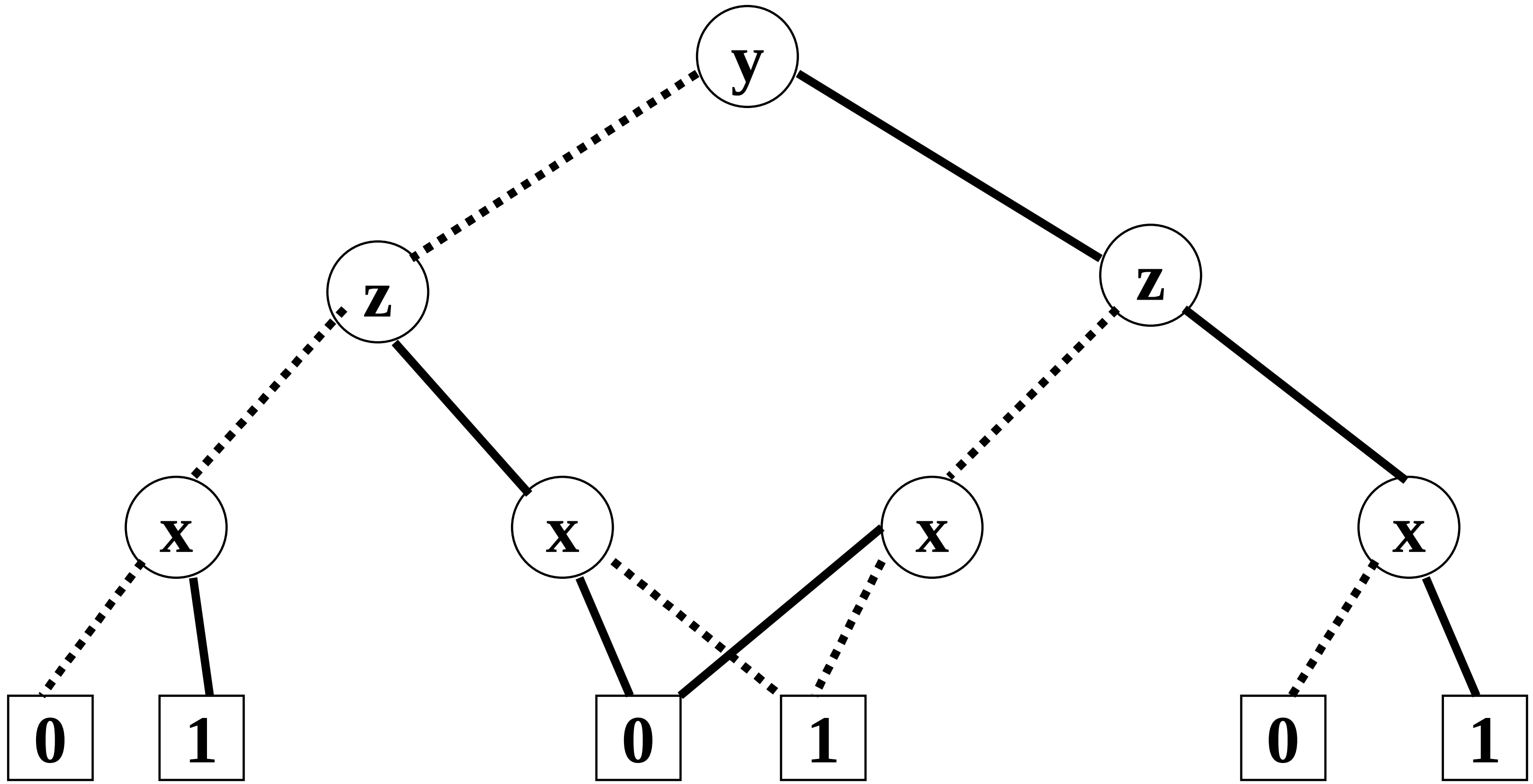
Ordered BDDS

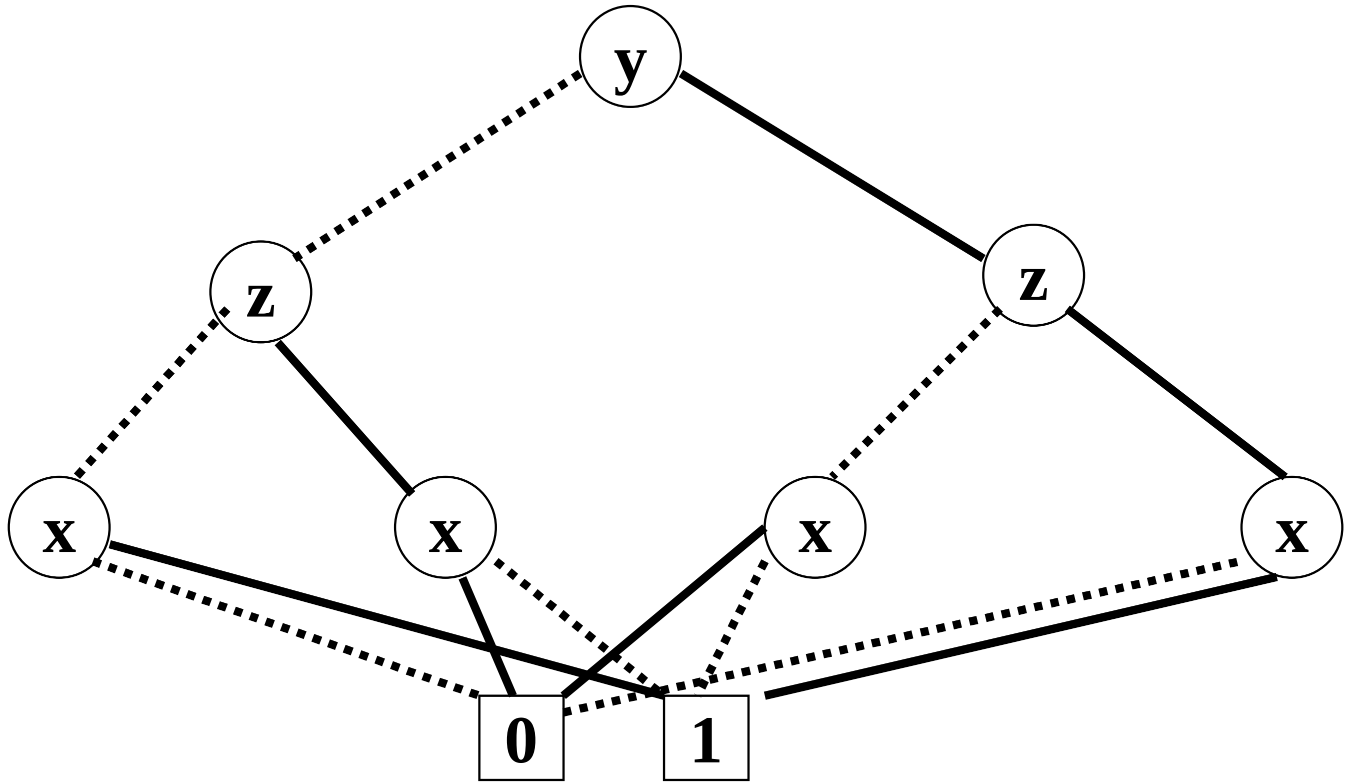
- Fundamental Fact:
  - For a fixed variable ordering, for each boolean function there is ***exactly one*** reduced ordered BDD!
  - Reduced OBDDs are ***canonical objects***.
  - To test if ***f*** and ***g*** are equal, we just have to check if **their** reduced **OBDD**s are **identical**.
  - This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

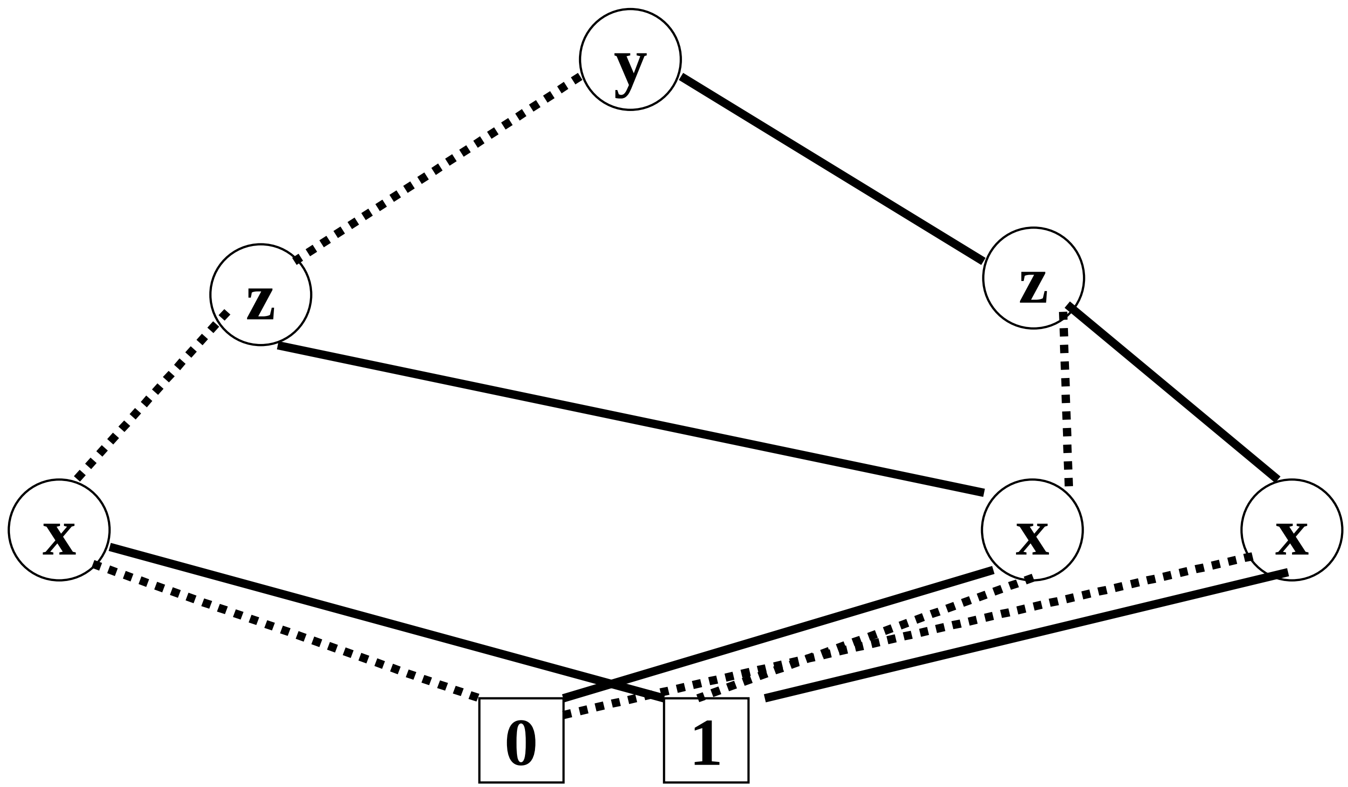
$y < z < x$

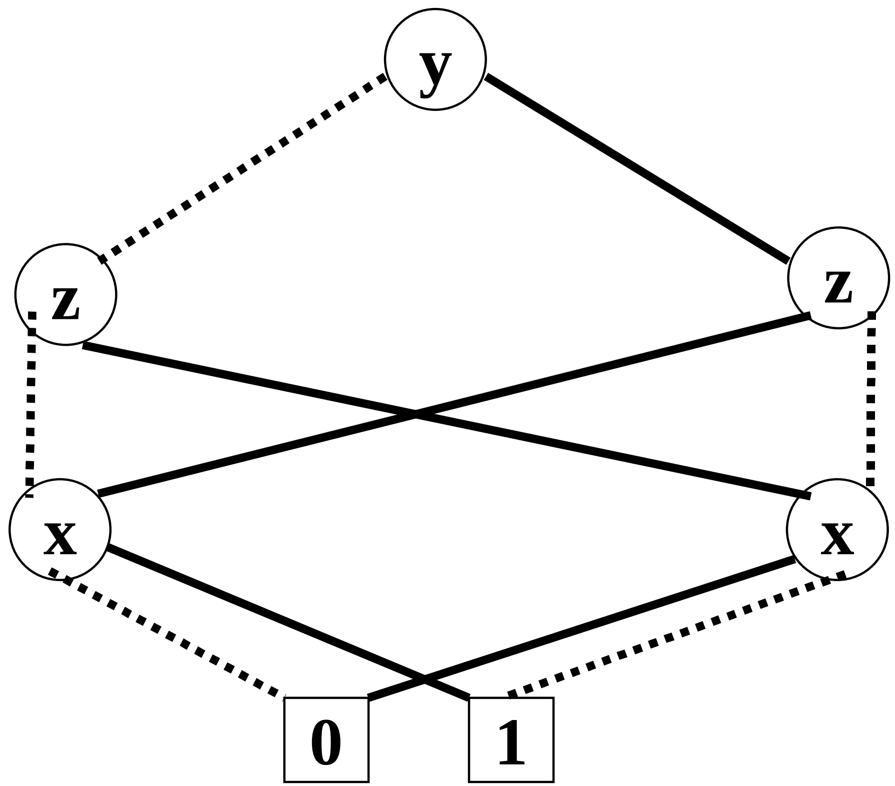



y
z
z
x
x
x
x
0
1
1
1
0
0
1


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

$$\mathbf{f^u = f(\mathit{x}_1,\ldots,\mathit{x}_n)}$$

Consequences of canonicity

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

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

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

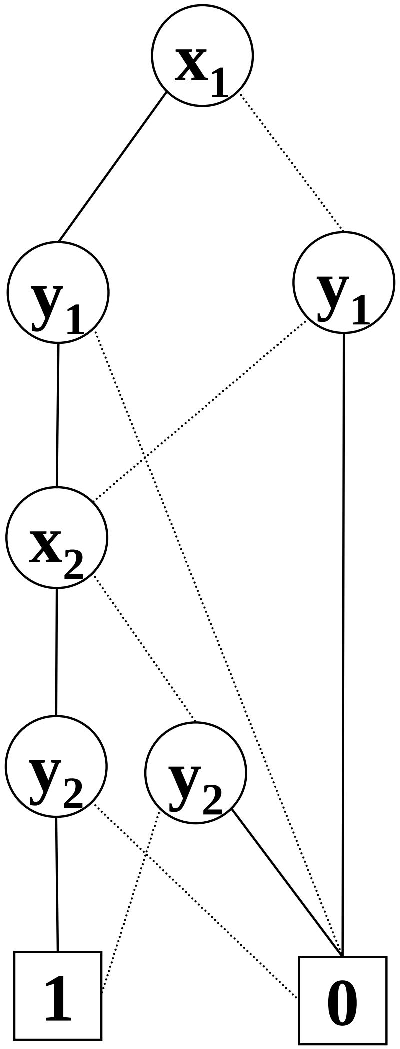
- ***The ordering is crucial!***
- $\{x_1, x_2, y_1, y_2\}$ x_1 x_2
  - $f(x_1, x_2, y_1, y_2)$ y_1 y_2
  - $f(x_1, x_2, y_1, y_2) = 1$ *iff* $(x_1 = y_1 \wedge x_2 = y_2)$
- If $x_1 < y_1 < x_2 < y_2$, then the **OBDD** is of size $3 \cdot 2 + 2 = 8$.
- If $x_1 < x_2 < y_1 < y_2$, then the **OBDD**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

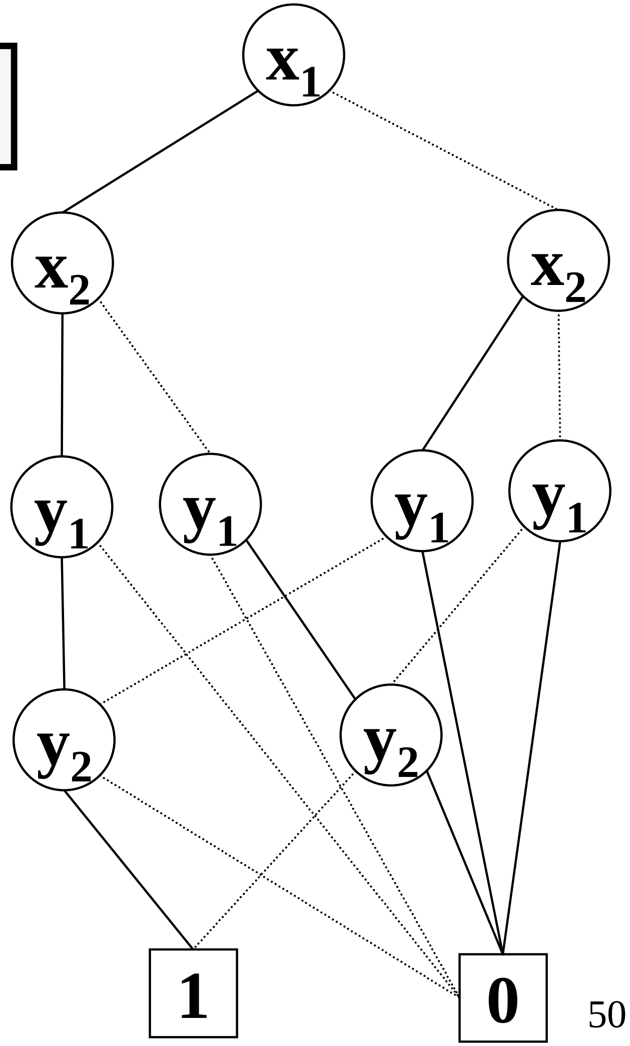
Reduced OBDDs

$x_1 < y_1 < x_2 < y_2$

$x_1 < x_2 < y_1 < y_2$

$$\boxed{(x_1 = y_1 \wedge x_2 = y_2)}$$





Reduced OBDDs

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

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

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

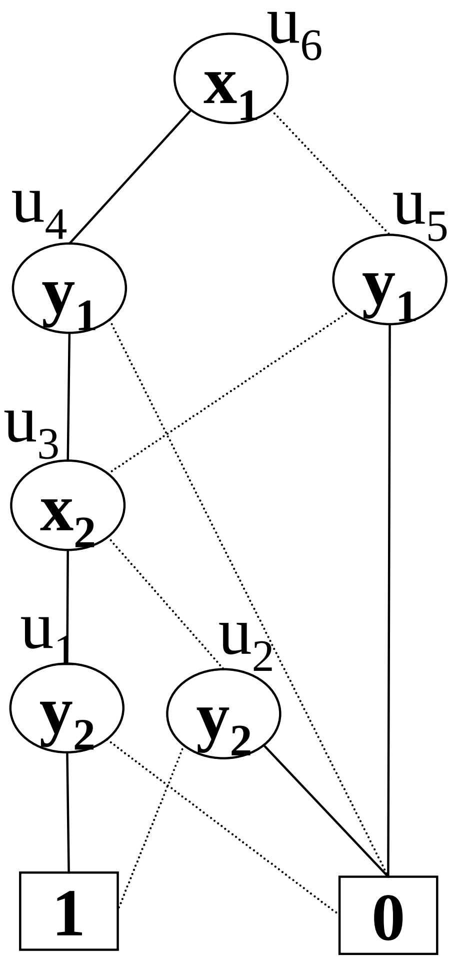
- If $x_1 < y_1 < x_2 < y_2 \ldots < x_n < y_n$, then the **OBDD** is of size **3n + 2**.
- If $x_1 < x_2 < \ldots < x_n < y_1 < \ldots < y_n$, then the **OBDD** is of size $3 \cdot 2^n - 1$!

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	**?**	**?**	**?**
$\mathbf{u_1}$	$\mathbf{y_2}$	**0**	**1**
$\mathbf{u_2}$	$\mathbf{y_2}$	**1**	**0**
$\mathbf{u_3}$	$\mathbf{x_2}$	$\mathbf{u_2}$	$\mathbf{u_1}$
$\mathbf{u_4}$	$\mathbf{y_2}$	**0**	$\mathbf{u_3}$
$\mathbf{u_5}$	$\mathbf{y_1}$	**0**	$\mathbf{u_3}$
$\mathbf{u_6}$	$\mathbf{x_1}$	$\mathbf{u_5}$	$\mathbf{u_4}$

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.
- $\mathbf{f} \vee \mathbf{g}$ **---------** $\mathbf{G_f} \;\; \mathbf{op}_{\vee}\, \mathbf{G_g} = \mathbf{G}_{\mathbf{f} \vee \mathbf{g}}$
- 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***.
- $\mathbf{X = X_G \cup X_{G'}}$
- 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):
- $\mathbf{f(x_1, x_2, \ldots, x_n)}$
  - $\mathbf{f|_{x_1 = 0} = f(0, x_2, \ldots, x_n)}$
    - $\mathbf{f = x_1 \vee (x_2 \wedge x_3)}$
    - $\mathbf{f|_{x_1 = 0} = x_2 \wedge x_3}$
  - 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 ROBDDs.

- Let **x** be the variable of the root of $\mathbf{G_f}$ and **y** the variable of the root 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 root of $\mathbf{G_f}$ and **y** the root 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!

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)
 Function App(u,v)
   if  G(u,v) ≠ empty then return G(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(u,v) = r
   return r
return App(u,v)
```

running time = $\mathbf{O(|G_u||G_v|)}$. Why?

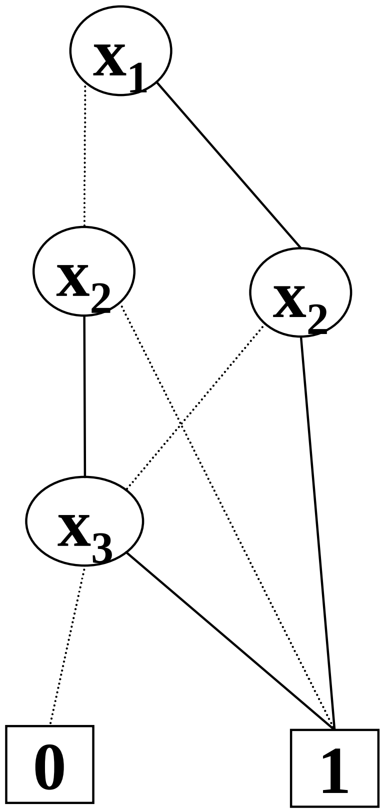
The Restrict operation

- ***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 $f(x_1,x_2,x_3) = ((x_1 \Leftrightarrow x_2) \vee x_3)$ we want to compute $f(x_1,x_2,x_3)[0/x_2] = f(x_1,0,x_3)$

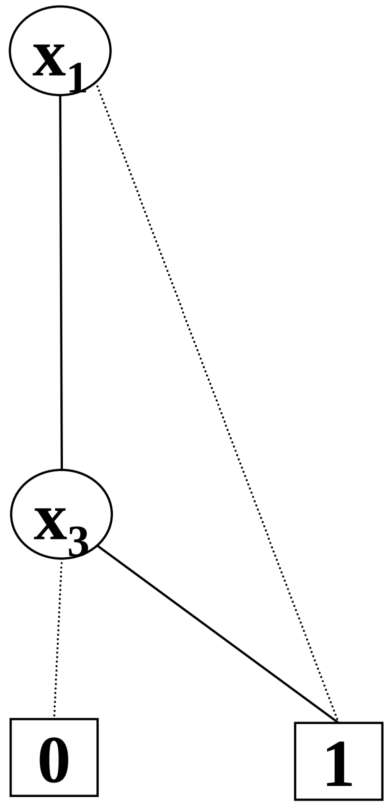
i.e.: $f(x_1,0,x_3)= \neg x_1 \vee x_3$

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 = $\mathbf{O(2^n)}$. Why?

Efficient algorithm for Restrict

```
Algorithm Restrict(u,i,b)
   init(G)
 Function Res(u)
   if  G(u) ≠ empty then return G(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(u) = r
   return r
return Res(u)
```

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

Quantification

- Extend the boolean language with

$$\exists \mathbf{x.t} \mid \forall \mathbf{x.t}$$

- They can be defined in terms of ROBDD operations:

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

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

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

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

Symbolic Model Checking

- $\mathbf{K = (S, S_0, R, AP, L)}$
- $\mathbf{AP = \{p, q, r\}}$
- Invent $\mathbf{\{p', q', r'\}}$
- Suppose $\mathbf{(s_1, s_2)}$ in $\mathbf{R}$ (i.e. $\mathbf{R(s_1, s_2)}$)
  with $\mathbf{L(s_1) = \{p, q\}}$ and $\mathbf{L(s_2) = \{r\}}$.
  Then $\mathbf{f_{R(s_1,s_2)} = f_{s_1} \wedge f'_{s_2}}$.
  - where $\mathbf{f'_{s_2} = \neg p' \wedge \neg q' \wedge r'}$
- $\mathbf{f_R} = \bigvee_{\mathbf{(s_1,\, s_2) \in R}} \mathbf{(f_{R(s_1,s_2)})}$
- **Choose the ordering** $\mathbf{p < p' < q < q' < r < r'}$ **!**

CTL symbolic Model Checking

- $\|[x_i]\| = f_{x_i}(x_i)$

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

  (apply negation of 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

- $\|[\mathbf{EX}\ \phi]\| =$

  $$\exists \mathbf{x'_1,\ldots,x'_n}(\mathbf{f}_\phi(\mathbf{x'_1,\ldots,x'_n}) \wedge \mathbf{f_R}(\mathbf{x_1,\ldots,x_n,x'_1,\ldots,x'_n}))$$

  (*relational product*, also known as *pre-image of R*)

- $\|[\mathbf{EU}(\phi,\psi)]\| =$

  $$\mu \mathbf{Z}.(\mathbf{f}_\psi(\mathbf{x_1,\ldots,x_n}) \vee (\mathbf{f}_\phi(\mathbf{x_1,\ldots,x_n}) \wedge \mathbf{EX\ Z}))$$

- $\|[\mathbf{EG}\ \phi]\| = \nu \mathbf{Z}.(\mathbf{f}_\phi(\mathbf{x_1,\ldots,x_n}) \wedge \mathbf{EX\ Z})$

Symbolic model checking: example

Given the boolean variable $V=\{x_1,\ldots, x_n\}$, **EG** ψ can be computed as follows:

- Assume the ROBDD $\mathbf{f_\psi(x_1,\ldots, x_n)}$ has been computed.
- $\mathbf{X_0 = f_\psi (x_1,\ldots, x_n)}$ [$\mathbf{f_\psi(x'_1,\ldots, x'_n)}$ by substitution]
- $\mathbf{X_{i+1} = X_i \cap Y_i}$ 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 be computed as $\mathbf{X_i \wedge Y_i}$
- Finally **whether** $\mathbf{X_{i+1} = X_i}$ can be checked by checking if the corresponding ROBDDs are **identical**.

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**.
- But we need **additional techniques** !