Formal Methods in software development

a.y.2017/2018

Prof. Anna Labella

OBDD [H-R chap.6]

Representing boolean functions

A formula can be represented by a boolean function, where its variables are boolean variables, as in circuits.

Definition 6.1 A boolean variable x is a variable ranging over the values 0 and 1. We write $x_1, x_2, \ldots$ and $x, y, z, \ldots$ to denote boolean variables. We define the following functions on the set $\{0, 1\}$:

- $\overline{0} \stackrel{\text{def}}{=} 1$ and $\overline{1} \stackrel{\text{def}}{=} 0$;
- $x \cdot y \stackrel{\text{def}}{=} 1$ if x and y have value 1; otherwise $x \cdot y \stackrel{\text{def}}{=} 0$;
- $x + y \stackrel{\text{def}}{=} 0$ if x and y have value 0; otherwise $x + y \stackrel{\text{def}}{=} 1$;
- $x \oplus y \stackrel{\text{def}}{=} 1$ if exactly one of x and y equals 1.

OBDD [H-R cap.6]

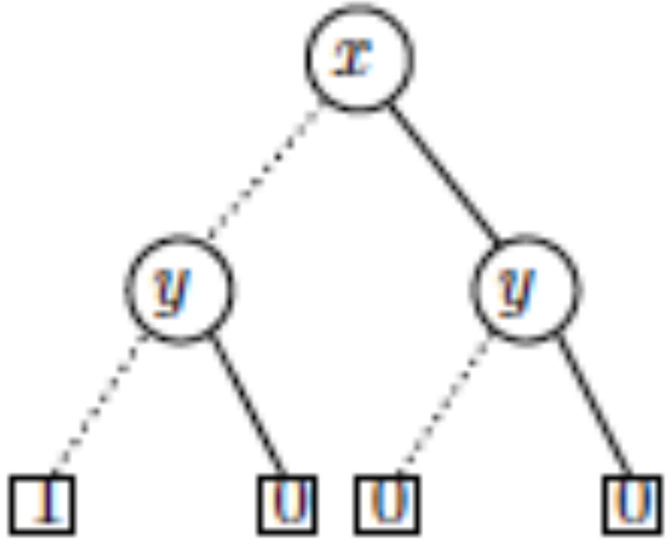
Representation of		test for		boolean operations		
boolean functions	compact?	satisf'ty	validity	$\cdot$	$+$	$\overline{\ }$
Prop. formulas	often	hard	hard	easy	easy	easy
Formulas in DNF	sometimes	easy	hard	hard	easy	hard
Formulas in CNF	sometimes	hard	easy	easy	hard	hard
Ordered truth tables	never	hard	hard	hard	hard	hard
Reduced OBDDs	often	easy	easy	medium	medium	easy

Exercises

Write down the truth tables for the boolean formulas

$$
\begin{array}{ll}
(1) & f(x,y) \stackrel{\text{def}}{=} x \cdot (y + \overline{x}) \\
(2) & g(x,y) \stackrel{\text{def}}{=} x \cdot y + (1 \oplus \overline{x}) \\
(3) & h(x,y,z) \stackrel{\text{def}}{=} x + y \cdot (x \oplus \overline{y}) \\
(4) & k() \stackrel{\text{def}}{=} 1 \oplus (0 \cdot \overline{1}).
\end{array}
$$

Binary decision trees



$$f(x,y) \stackrel{\text{def}}{=} \overline{x+y}.$$

Exercises

2. Consider the following truth table:

p	q	r	ϕ
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	F

Write down a binary decision tree which represents the boolean function specified in this truth table.

Exercises

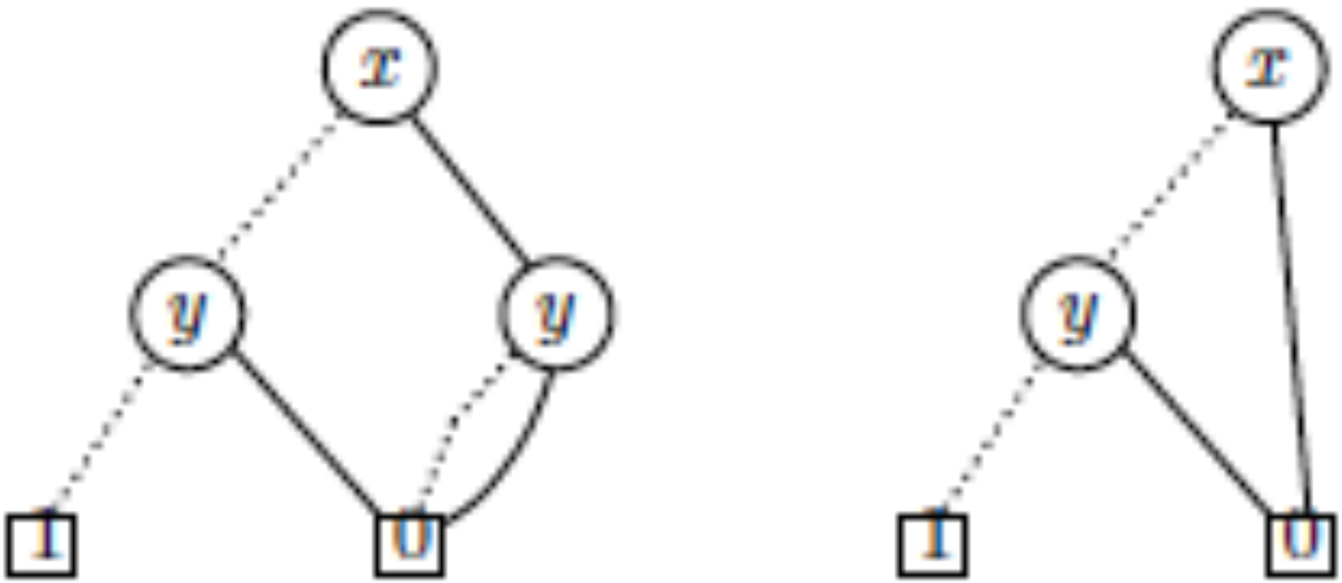
Consider the following boolean function given by its truth table:

x	y	z	$f(x,y,z)$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

(a) Construct a binary decision tree for $f(x,y,z)$ such that the root is an x-node followed by y- and then z-nodes.

(b) Construct another binary decision tree for $f(x,y,z)$, but now let its root be a z-node followed by y- and then x-nodes.

BDD

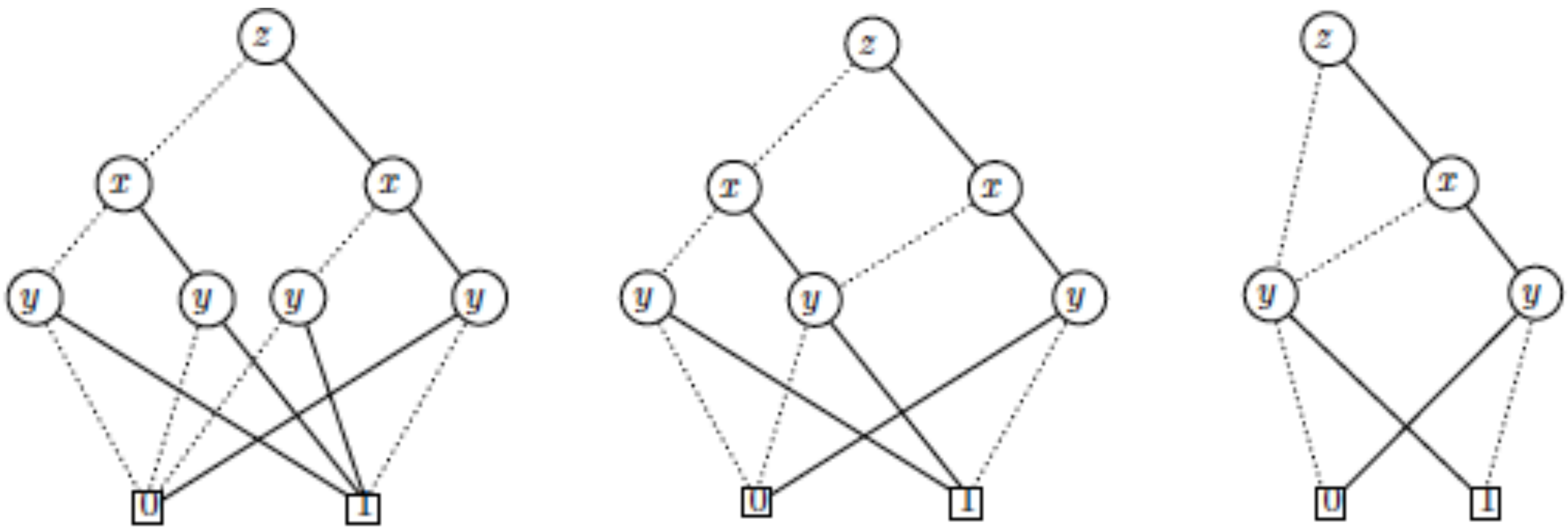


Removal of duplicated terminals

Removal of redundant tests

(they are nomore trees)

BDD



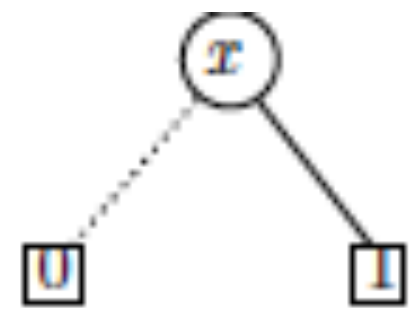
Removal of duplicated non terminals

BDD

- Eliminate duplicated terminals
- Eliminate redundant tests
- Eliminate duplicated non terminals

BDD: how do we introduce operations? Composing BDD

- constants
- variables



BDD: sums and products

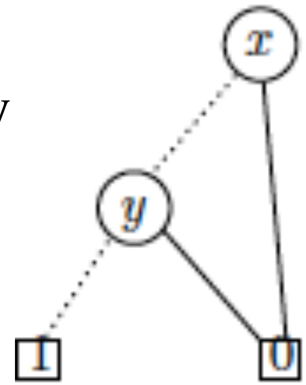
- We can substitute terminals by non terminals and compose functions

e.g.

- In the conjunction we substitute 1 by the BDD of the other function
- In the disjunction we substitute 0 by the BDD of the other function
- In the negation we swap 1 and 0

Exercises

Let f be represented by



describe

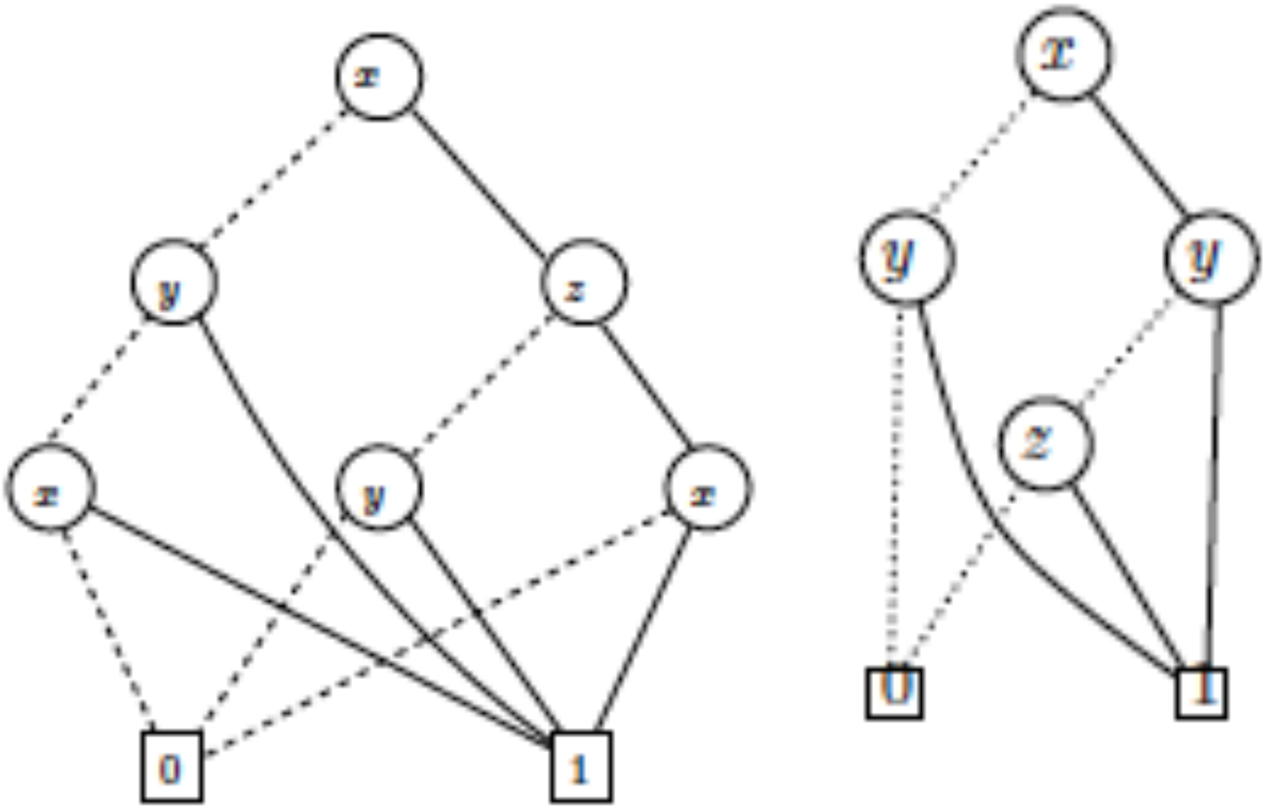
(a) $f \cdot x$
(b) $x + f$
(c) $\overline{f \cdot 0}$
(d) $f \cdot 1$.

BDD

- Satisfiability: to reach 1 via a coherent path
- Validity: it is not possible to reach 0 via a coherent path

OBDD: ordered binary decision diagrams

Equivalence?



OBDD

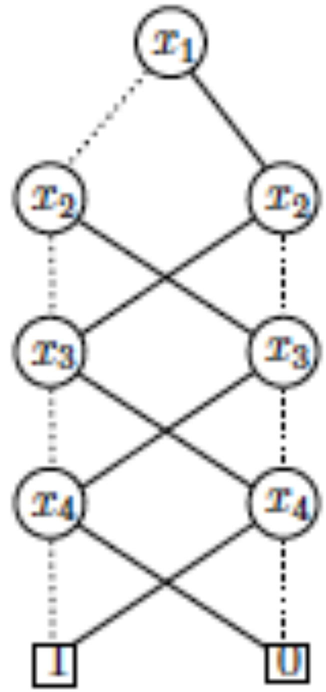
Normal form: order and then reduce

Theorem. We have a unique result

Hence there is a canonical form

OBDD

An example: the parity function on 4 variables



Repeated variables

OBDD

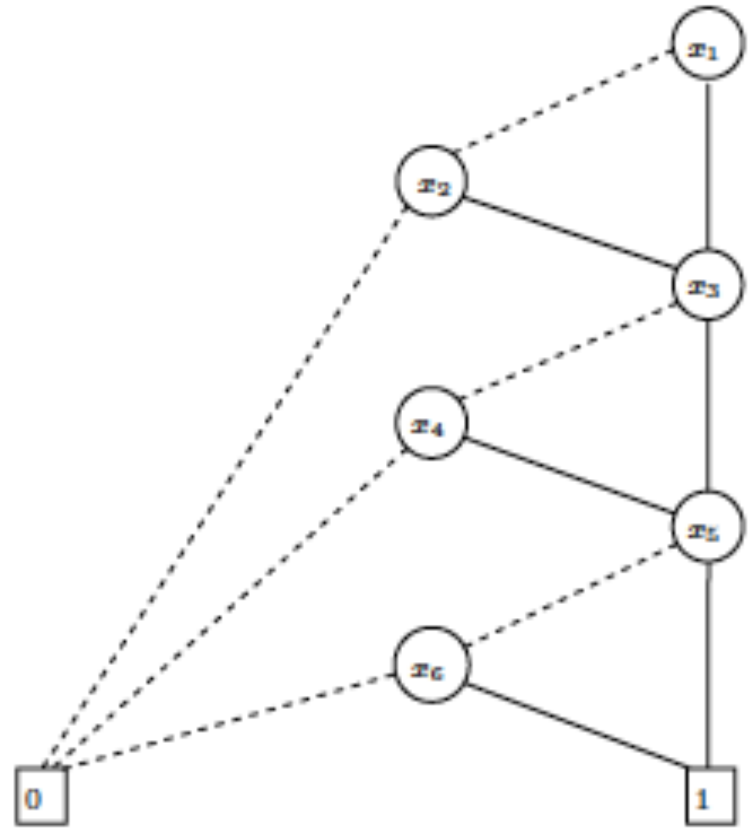


Figure 6.12. The OBDD for $(x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_5 + x_6)$ with variable ordering $[x_1, x_2, x_3, x_4, x_5, x_6]$.

order without repetitions

OBDD

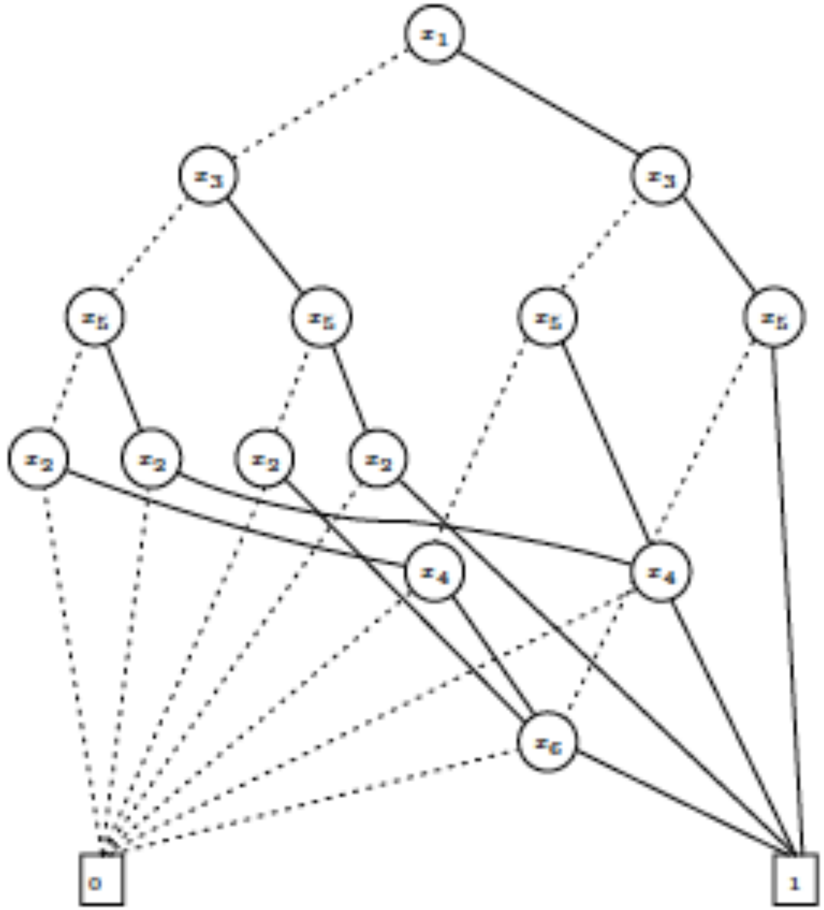


Figure 6.13. Changing the ordering may have dramatic effects on the size of an OBDD: the OBDD for $(x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_5 + x_6)$ with variable ordering $[x_1, x_3, x_5, x_2, x_4, x_6]$.

OBDD

- Composition is nomore directly allowed

But we have:

- Absence of redundant variables
- Test for semantic equivalence with a compatible ordering
- Test for validity (reducibility to B_1)
- Test for satisfiability (non reducibility to B_0)
- Test for implication (reducibility of $f \cdot \underline{g}$ to B_0)

Exercises

Given the truth table

x	y	z	$f(x,y,z)$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

compute the reduced OBDD with respect to the following ordering of variables:

(a) $[x, y, z]$
(b) $[z, y, x]$
(c) $[y, z, x]$
(d) $[x, z, y]$.

OBDD: the algorithm `reduce`

It identifies equal nodes going bottom up

- If the label id(lo(n)) is the same as id(hi(n)), then we set id(n) to be that label. That is because the boolean function represented at n is the same function as the one represented at lo(n) and hi(n). In other words, node n performs a redundant test and can be eliminated by reduction C2.
- If there is another node m such that n and m have the same variable x_i, and id(lo(n)) = id(lo(m)) and id(hi(n)) = id(hi(m)), then we set id(n) to be id(m). This is because the nodes n and m compute the same boolean function (compare with reduction C3).
- Otherwise, we set id(n) to the next unused integer label.

OBDD: the algorithm `reduce`

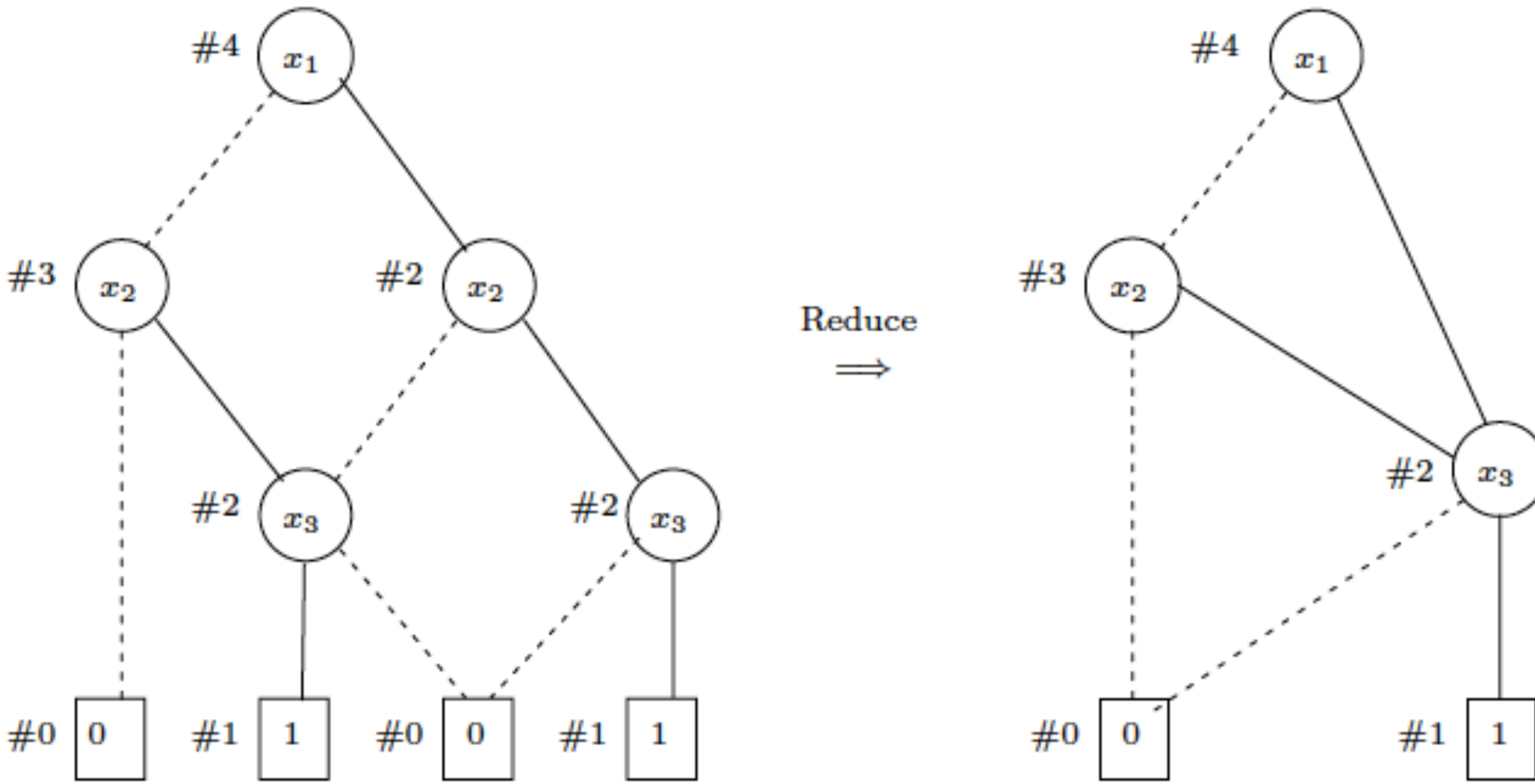


Figure 6.14. An example execution of the algorithm `reduce`.

OBDD: the algorithm `apply`

It exploits operations acting on (op, B_f, B_g)

The intuition behind the **`apply`** algorithm is fairly simple. The algorithm operates recursively on the structure of the two OBDDs:

1. let v be the variable highest in the ordering (=leftmost in the list) which occurs in B_f or B_g.
2. split the problem into two subproblems for v being 0 and v being 1 and solve recursively;
3. at the leaves, apply the boolean operation op directly.

OBDD: the algorithm `apply`

Shannon expansion theorem

$$f \equiv \underline{x}\, f[0/x] + x\, f[1/x]$$

In general form

$$f \; op \; g = \underline{x}_i \,(f[0/x_i] \; op \; g[0/x_i]) + x_i \,(f[1/x_i] \; op \; g[1/x_i])$$

This provides a recursive call structure

OBDD: the algorithm `apply`

1. If both r_f and r_g are terminal nodes with labels l_f and l_g, respectively (recall that terminal labels are either 0 or 1), then we compute the value $l_f \,\mathrm{op}\, l_g$ and let the resulting OBDD be B_0 if that value is 0 and B_1 otherwise.
2. In the remaining cases, at least one of the root nodes is a non-terminal. Suppose that both root nodes are x_i-nodes. Then we create an x_i-node n with a dashed line to **apply** $(\mathrm{op}, \mathrm{lo}(r_f), \mathrm{lo}(r_g))$ and a solid line to **apply** $(\mathrm{op}, \mathrm{hi}(r_f), \mathrm{hi}(r_g))$, i.e. we call **apply** recursively on the basis of (6.2).
3. If r_f is an x_i-node, but r_g is a terminal node or an x_j-node with $j > i$, then we know that there is no x_i-node in B_g because the two OBDDs have a compatible ordering of boolean variables. Thus, g is independent of x_i ($g \equiv g[0/x_i] \equiv g[1/x_i]$). Therefore, we create an x_i-node n with a dashed line to **apply** $(\mathrm{op}, \mathrm{lo}(r_f), r_g)$ and a solid line to **apply** $(\mathrm{op}, \mathrm{hi}(r_f), r_g)$.
4. The case in which r_g is a non-terminal, but r_f is a terminal or an x_j-node with $j > i$, is handled symmetrically to case 3.

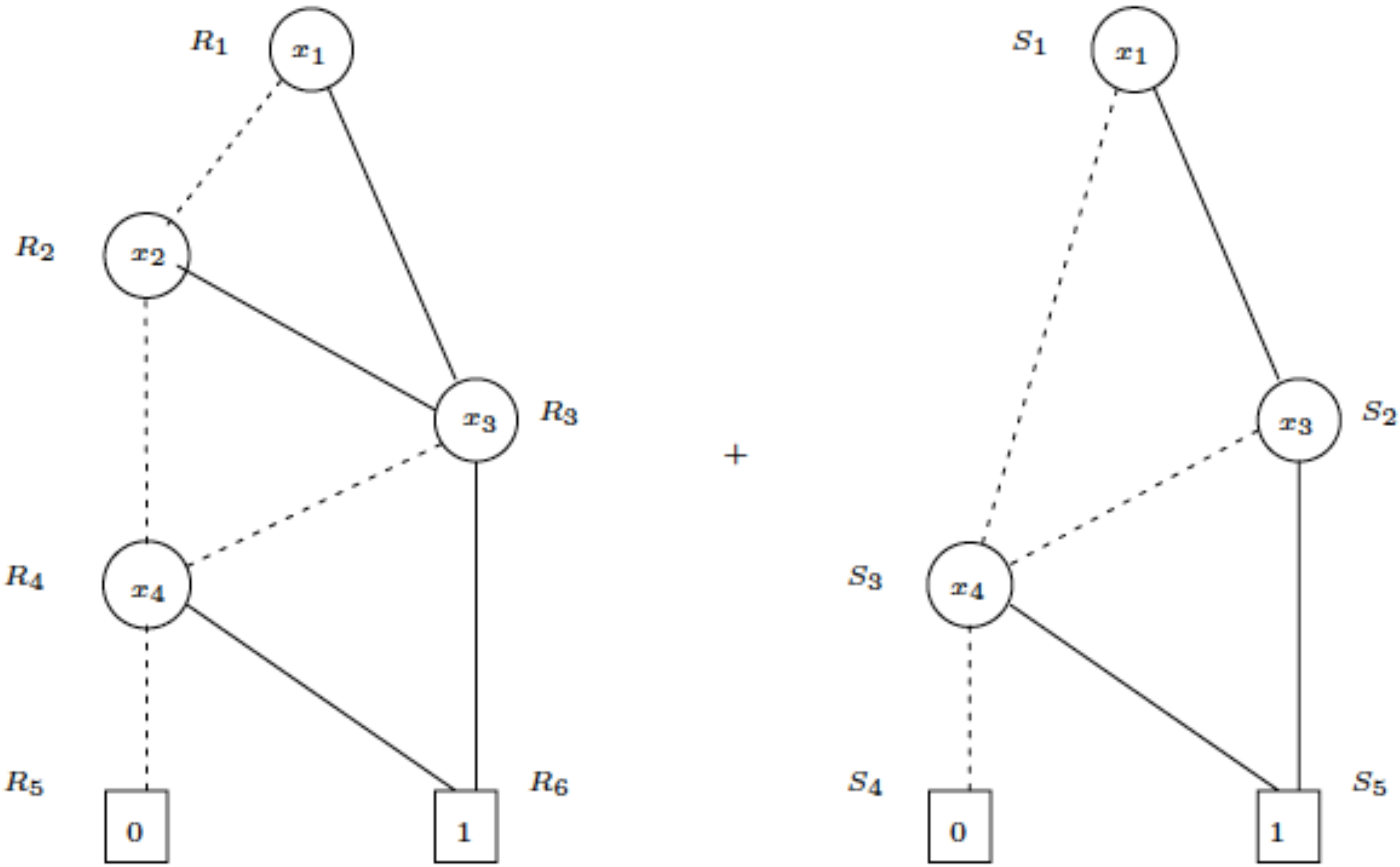


Figure 6.15. An example of two arguments for a call apply $(+, B_f, B_g)$.

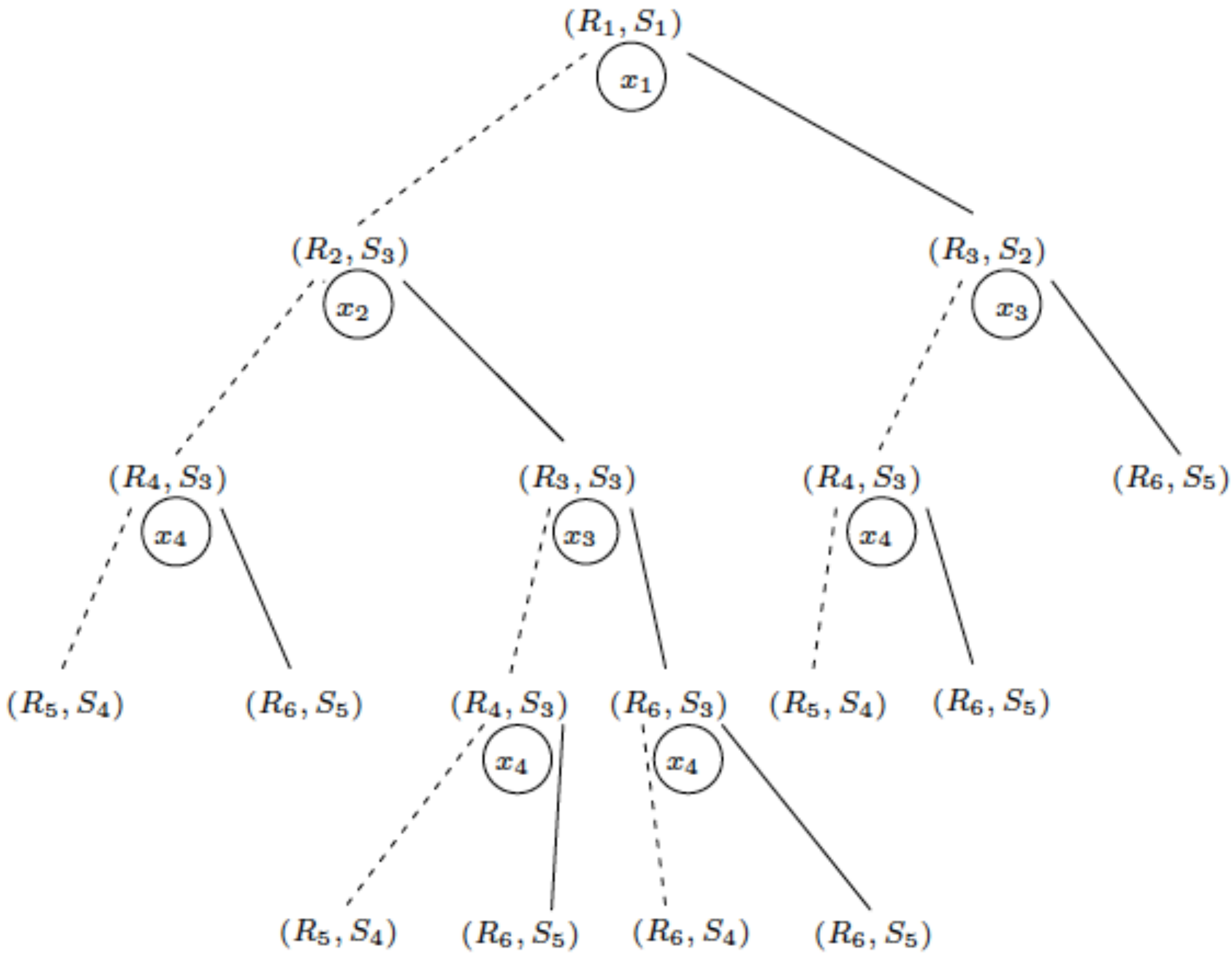


Figure 6.16. The recursive call structure of apply for the example

OBDD: the algorithm `apply`

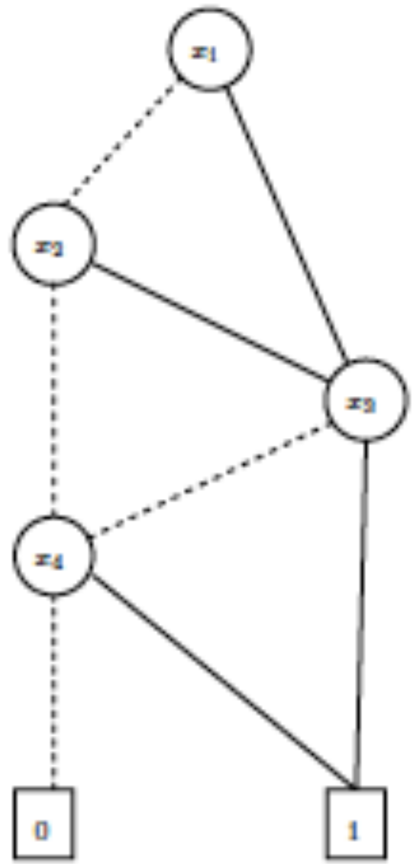


Figure 6.17. The result of apply $(+, B_f, B_g)$, where B_f and B_g are given in Figure 6.15.

OBDD: the algorithm `restrict`

Given f, `restrict`$(0, x, B_f)$ computes the reduced OBDD corresponding to *f[0/x]*

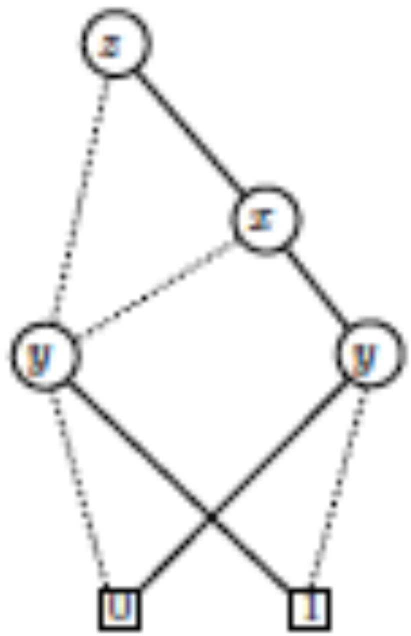
Analogously, `restrict`$(0, x, B_f)$ computes the reduced OBDD corresponding to to *f[1/x]*

`restrict`$(0, x, B_f)$ works as follows: For each node n labeled with x, incoming edges are redirected to $\mathrm{lo}(n)$ and n is removed. Then we call **reduce** on the resulting OBDD. The call **restrict** $(1, x, B_f)$ proceeds similarly, only we now redirect incoming edges to $\mathrm{hi}(n)$.

OBDD: `restrict` exercise

Let f be the reduced OBDD represented in Figure 6.5(b) (page 364). Compu
the reduced OBDD for the restrictions:

(a) $f[0/x]$
(b) $f[1/x]$
(c) $f[1/y]$
(d) $f[0/z]$.



OBDD: the algorithm `exists`

$$\exists f = f[0/x] + f[1/x]$$

$$\texttt{apply}\,(+, \texttt{restrict}\,(0, x, B_f), \texttt{restrict}\,(1, x, B_f))$$

$$\forall f = f[0/x] \,.\, f[1/x]$$

$$\texttt{apply}\,(\cdot, \texttt{restrict}\,(0, x, B_f), \texttt{restrict}\,(1, x, B_f))$$

OBDD: the algorithm `exists` exercise

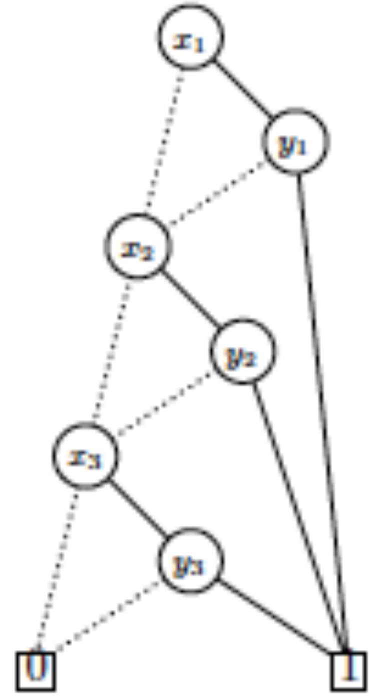


Figure 6.19. A BDD B_f to illustrate the exists algorithm.

OBDD: the algorithm `exists`

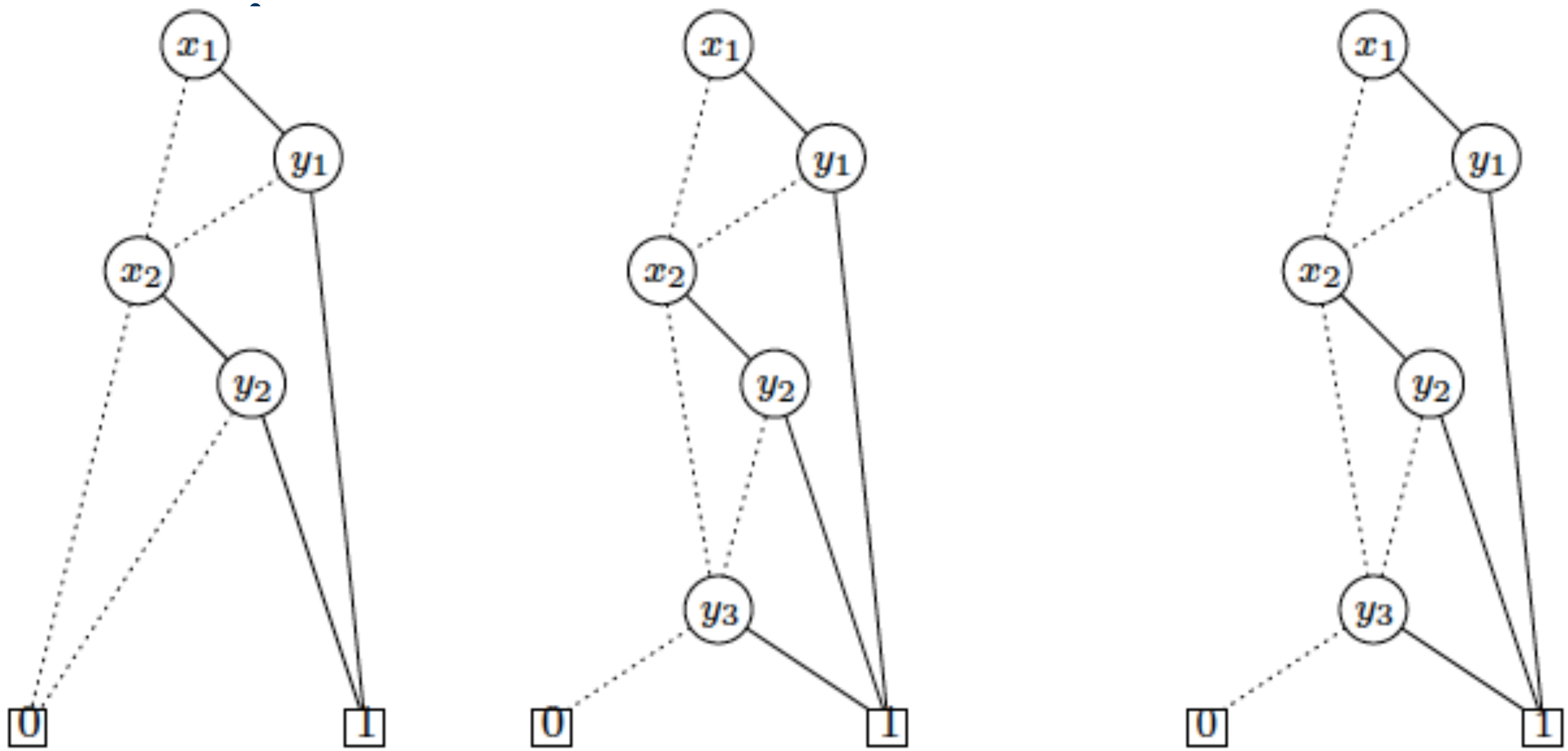


Figure 6.20. restrict$(0, x_3, B_f)$ and restrict$(1, x_3, B_f)$ and the result of applying $+$ to them.

Boolean formula f	Representing OBDD B_f
0	B_0 (Fig. 6.6)
1	B_1 (Fig. 6.6)
x	B_x (Fig. 6.6)
$\overline{f}$	swap the 0- and 1-nodes in B_f
$f + g$	$\mathtt{apply}\ (+, B_f, B_g)$
$f \cdot g$	$\mathtt{apply}\ (\cdot\ , B_f, B_g)$
$f \oplus g$	$\mathtt{apply}\ (\oplus, B_f, B_g)$
$f[1/x]$	$\mathtt{restrict}\ (1, x, B_f)$
$f[0/x]$	$\mathtt{restrict}\ (0, x, B_f)$
$\exists x.f$	$\mathtt{apply}\ (+, B_{f[0/x]}, B_{f[1/x]})$
$\forall x.f$	$\mathtt{apply}\ (\cdot\ , B_{f[0/x]}, B_{f[1/x]})$

Complexity

Algorithm	Input OBDD(s)	Output OBDD	Time-complexity
`reduce`	B	reduced B	$O(\|B\| \cdot \log \|B\|)$
`apply`	B_f, B_g (reduced)	$B_{f \text{ op } g}$ (reduced)	$O(\|B_f\| \cdot \|B_g\|)$
`restrict`	B_f (reduced)	$B_{f[0/x]}$ or $B_{f[1/x]}$ (reduced)	$O(\|B_f\| \cdot \log \|B_f\|)$
$\exists$	B_f (reduced)	$B_{\exists x_1.\exists x_2....\exists x_n.f}$ (reduced)	NP-complete

Exercises

9. Compute $\texttt{CNF}$ $(\texttt{NNF}$ $(\texttt{IMPL\_FREE}$ $\neg(p \to (\neg(q \land (\neg p \to q))))))$.
10. Use structural induction on the grammar of formulas in CNF to show that the 'otherwise' case in calls to $\texttt{DISTR}$ applies iff both η_1 and η_2 are of type D in (1.6) on page 55.
11. Use mathematical induction on the height of ϕ to show that the call $\texttt{CNF}$ $(\texttt{NNF}$ $(\texttt{IMPL\_FREE}$ $\phi))$ returns, up to associativity, ϕ if the latter is already in CNF.
12. Why do the functions $\texttt{CNF}$ and $\texttt{DISTR}$ preserve NNF and why is this important?
13. For the call $\texttt{CNF}\,(\texttt{NNF}\,(\texttt{IMPL\_FREE}\,(\phi)))$ on a formula ϕ of propositional logic, explain why
    (a) its output is always a formula in CNF
    (b) its output is semantically equivalent to ϕ
    (c) that call always terminates.

7. Construct a formula in CNF based on each of the following truth tables:

* (a)

p	q	ϕ_1
T	T	F
F	T	F
T	F	F
F	F	T

* (b)

p	q	r	ϕ_2
T	T	T	T
T	T	F	F
T	F	T	F
F	T	T	T
T	F	F	F
F	T	F	F
F	F	T	T
F	F	F	F