Unit-11: Binary Decision Diagrams (BDDs)

B. Srivathsan

Chennai Mathematical Institute

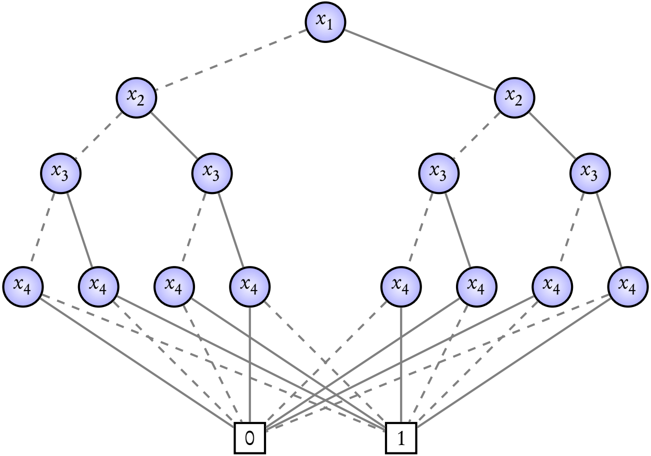
NPTEL-course

July - November 2015

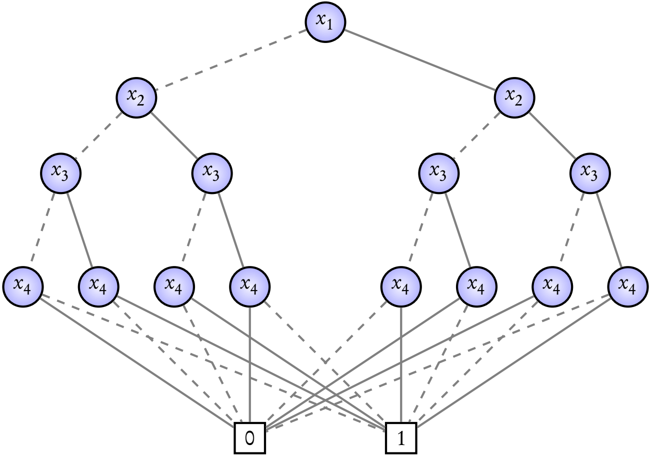
Module 2:

Ordered BDDs

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$

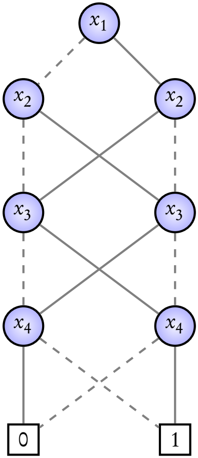


$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$



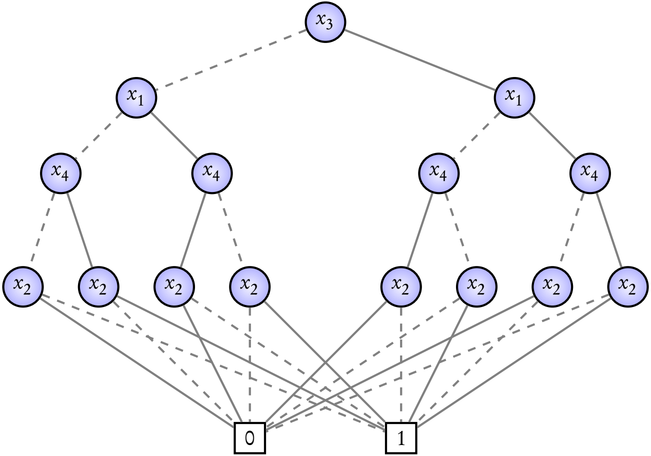
Ordered BDD for f with order $[x_1, x_2, x_3, x_4]$

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$



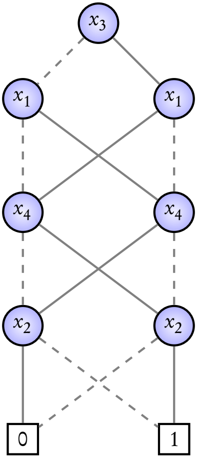
Reduced Ordered BDD for f with order $[x_1, x_2, x_3, x_4]$

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$



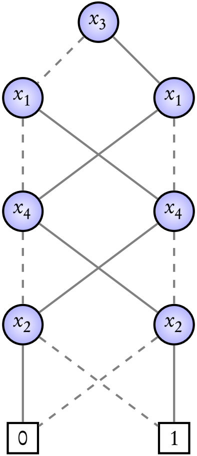
Ordered BDD for f with order $[x_3, x_1, x_4, x_2]$

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$



Reduced Ordered BDD for f with order $[x_3, x_1, x_4, x_2]$

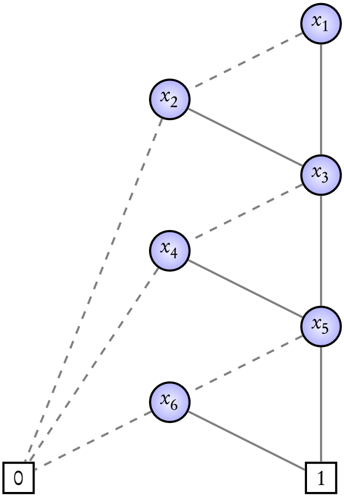
$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if an even number of variables is 1} \\ 0 & \text{otherwise} \end{cases}$$



f is **not sensitive** to ordering

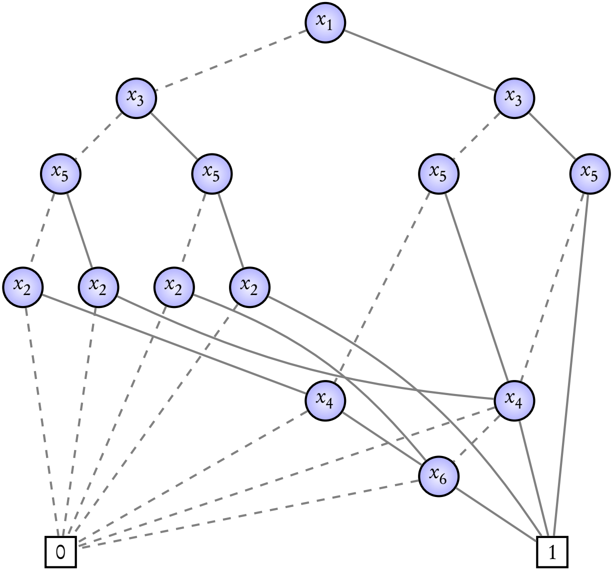
Reduced Ordered BDD for f with order $[x_3, x_1, x_4, x_2]$

$$g(x_1,x_2,x_3,x_4,x_5,x_6) = (x_1+x_2)\cdot(x_3+x_4)\cdot(x_5+x_6)$$



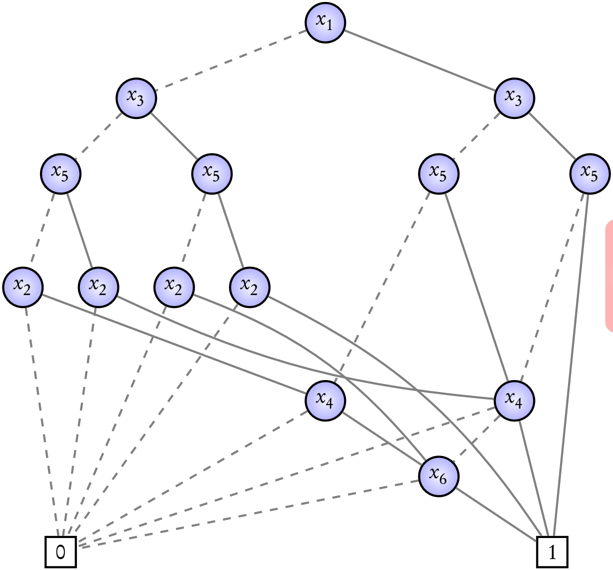
Reduced Ordered BDD (ROBDD) for g with order $[x_1,x_2,x_3,x_4,x_5,x_6]$

$$g(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_5 + x_6)$$



ROBDD for g with order $[x_1, x_3, x_5, x_2, x_4, x_6]$

$$g(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_5 + x_6)$$



g is **sensitive** to ordering

ROBDD for g with order $[x_1, x_3, x_5, x_2, x_4, x_6]$

Ordered BDDs

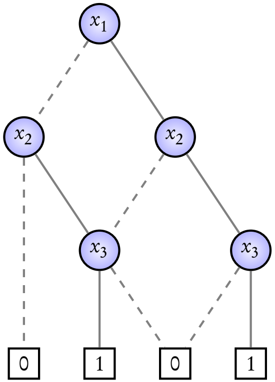
- BDDs with a **specified ordering** of variables
- For a given ordering, the reduced OBDD is **unique**
- Size of OBDD **depends** on the chosen **ordering**
- In practice, **heuristics** exist to find good orderings

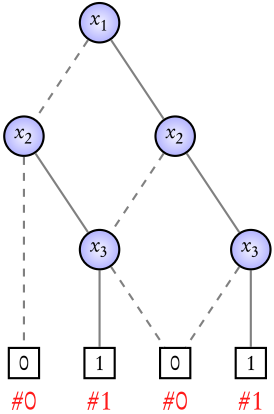
Ordered BDDs

- BDDs with a **specified ordering** of variables
- For a given ordering, the reduced OBDD is **unique**
- Size of OBDD **depends** on the chosen **ordering**
- In practice, **heuristics** exist to find good orderings

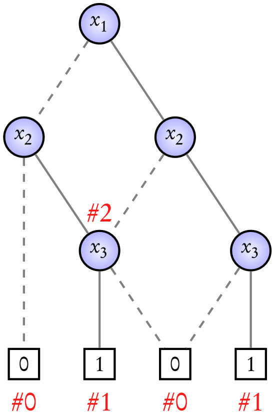
Coming next: Operations on OBDDs

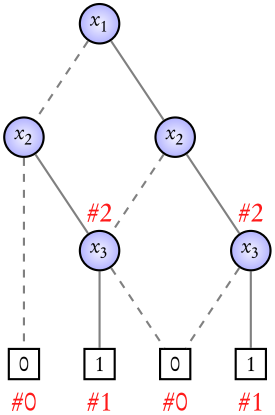
Algorithm to reduce an OBDD


x_1
x_2
x_2
x_3
x_3
0
1
0
1

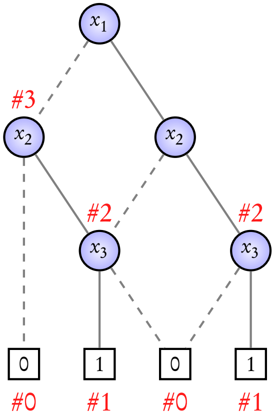


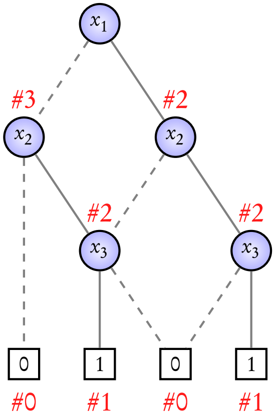
x_1
x_2
x_2
x_3
x_3
0
1
0
1
#0
#1
#0
#1

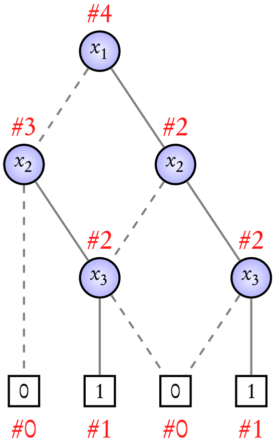

x_1
x_2
x_2
#2
x_3
x_3
0
1
0
1
#0
#1
#0
#1

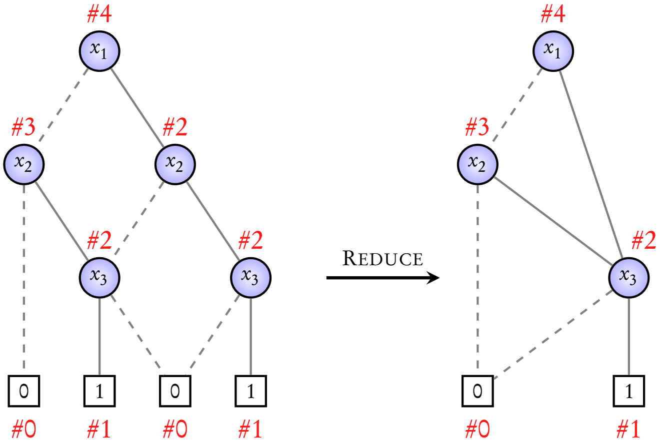


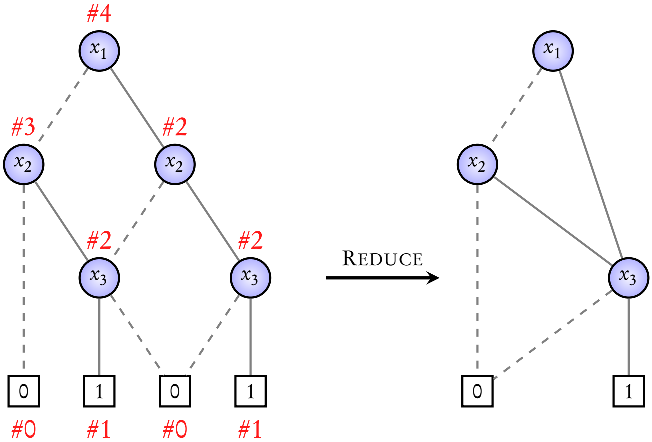
x_1
x_2
x_2
#2
#2
x_3
x_3
0
1
0
1
#0
#1
#0
#1

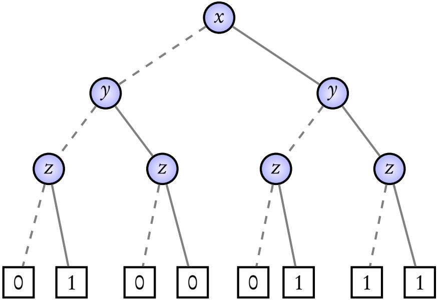

x_1
#3
x_2
x_2
#2
#2
x_3
x_3
0
1
0
1
#0
#1
#0
#1

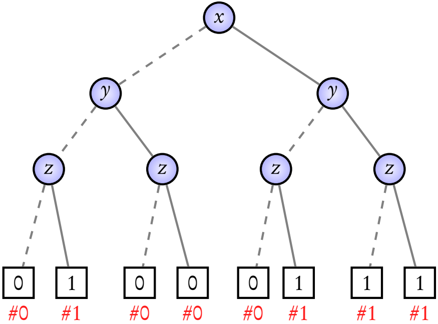

x_1
#3
x_2
#2
x_2
#2
x_3
#2
x_3
0
1
0
1
#0
#1
#0
#1

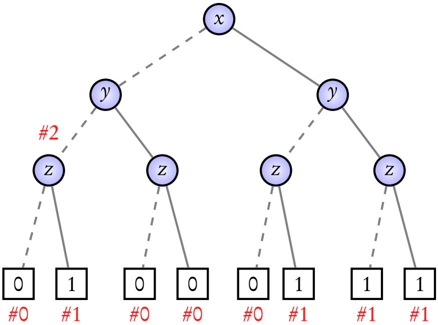

#4
x_1
#3
x_2
#2
x_2
#2
x_3
#2
x_3
0
1
0
1
#0
#1
#0
#1

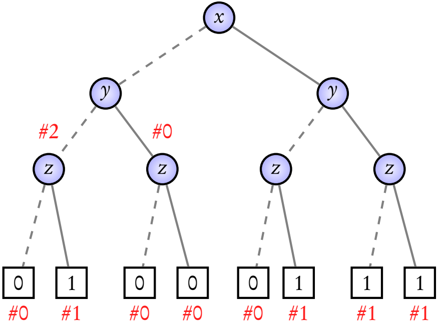

#4
x_1
#3
x_2
#2
x_2
#2
x_3
#2
x_3
0
1
0
1
#0
#1
#0
#1
REDUCE
#4
x_1
#3
x_2
#2
x_3
0
1
#0
#1

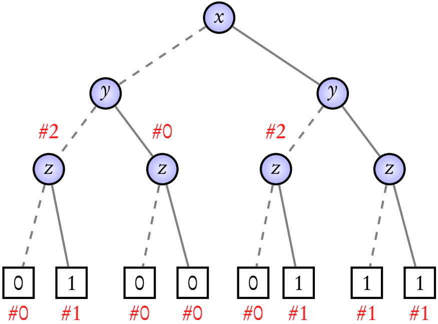

#4
x1
#3
x2
#2
x2
#2
x3
#2
x3
0
1
0
1
#0
#1
#0
#1
REDUCE
x1
x2
x3
0
1

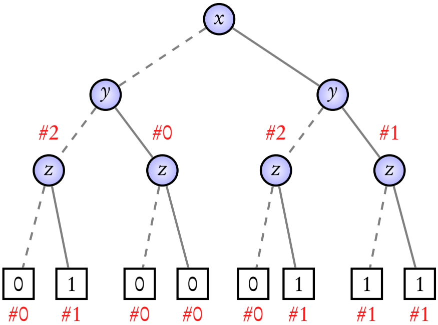

x
y
y
z
z
z
z
0
1
0
0
0
1
1
1

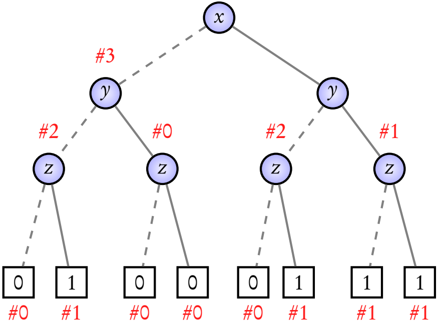

x
y
y
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1


x
y
y
#2
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1

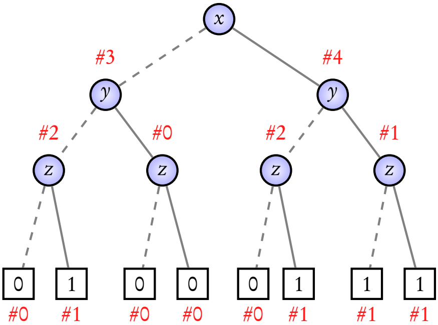

x
y
y
#2
#0
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1

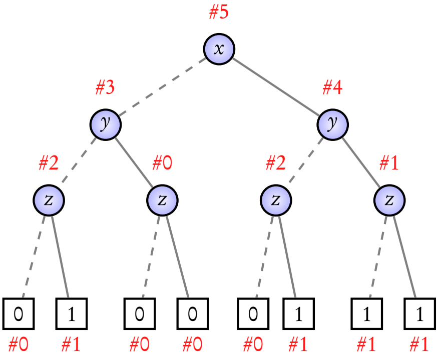

x
y
y
#2
#0
#2
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1

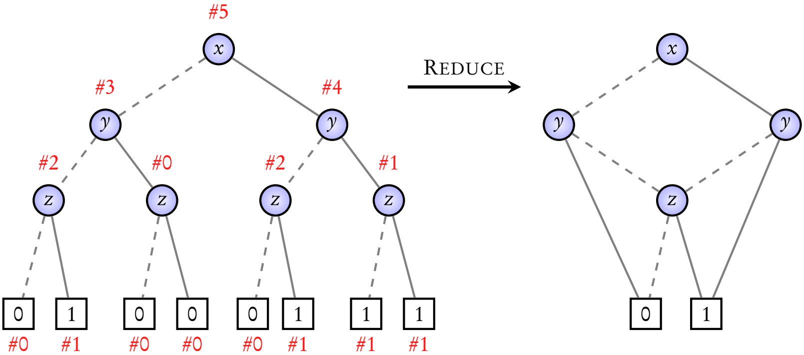

x
y
y
#2
#0
#2
#1
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1



x
#3
y
y
#2
#0
#2
#1
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1


x
#3
#4
y
y
#2
#0
#2
#1
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1


#5
x
#3
y
#4
y
#2
z
#0
z
#2
z
#1
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1


#5
x
Reduce
#3
#4
y
y
#2
#0
#2
#1
z
z
z
z
0
1
0
0
0
1
1
1
#0
#1
#0
#0
#0
#1
#1
#1
x
y
y
z
0
1

Algorithm to reduce OBDD

Algorithm to reduce OBDD

- **Leaves:** Label all `0` leaves with #`0` and all 1 leaves with #1

Algorithm to reduce OBDD

- **Leaves:** Label all 0 leaves with #0 and all 1 leaves with #1

- **Intermediate node** n

Algorithm to reduce OBDD

- **Leaves:** Label all `0` leaves with #`0` and all 1 leaves with #1
- **Intermediate node** n
  - If `0`**-child** and **1-child** of n have **same** label, set label of n to be that label

Algorithm to reduce OBDD

- **Leaves:** Label all `0` leaves with #`0` and all 1 leaves with #1
- **Intermediate node** n
  - If `0`**-child** and **1-child** of n have **same** label, set label of n to be that label
  - If there is **another** node m such that m has the **same variable** x_i and the **children** of n and m have **same** label, then set label of n to be the label of m

Algorithm to reduce OBDD

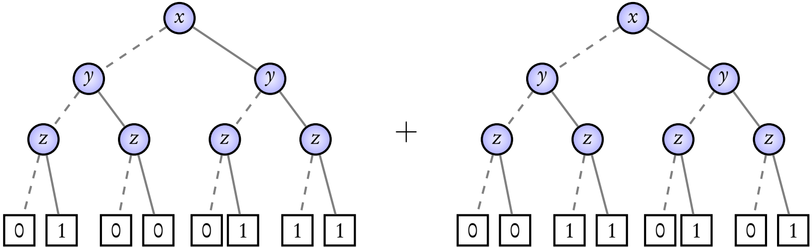
- **Leaves:** Label all `0` leaves with #`0` and all 1 leaves with #1
- **Intermediate node n**
  - If **`0`-child** and **1-child** of n have **same** label, set label of n to be that label
  - If there is **another** node m such that m has the **same variable** x_i and the **children** of n and m have **same** label, then set label of n to be the label of m
  - Otherwise set label of n to be **next unused integer**

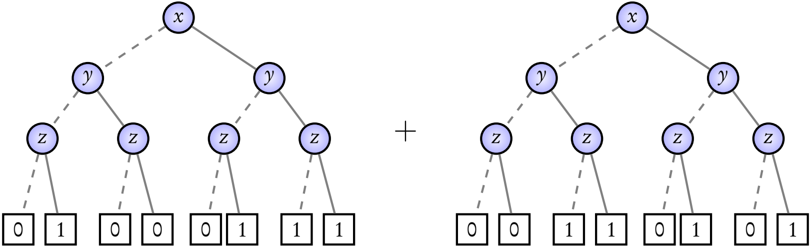
Algorithm to reduce OBDD

- **Leaves:** Label all `0` leaves with #`0` and all 1 leaves with #1
- **Intermediate node** n
  - If **`0`-child** and **1-child** of n have **same** label, set label of n to be that label
  - If there is **another** node m such that m has the **same variable** x_i and the **children** of n and m have **same** label, then set label of n to be the label of m
  - Otherwise set label of n to be **next unused integer**

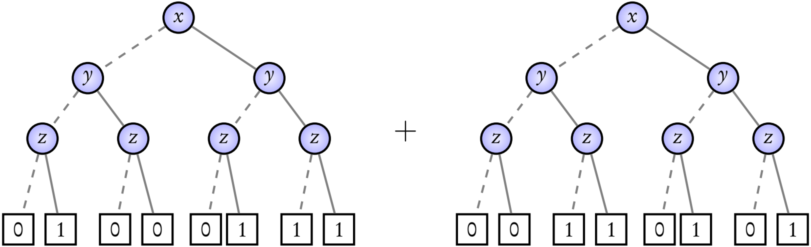
Reference: Logic in Computer Science, 2^{nd} edition, by *Huth* and *Ryan*
Section 6.2.1

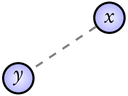
Coming next: Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

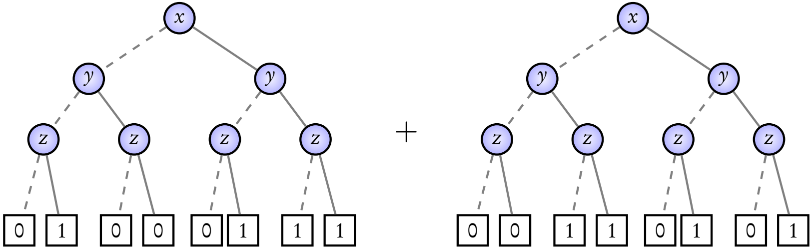

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

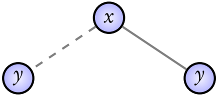

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

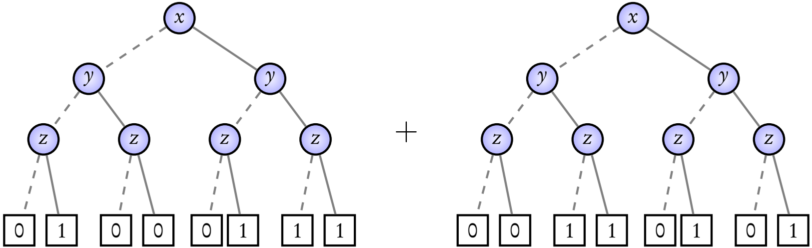

x

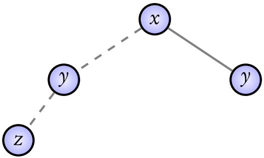

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

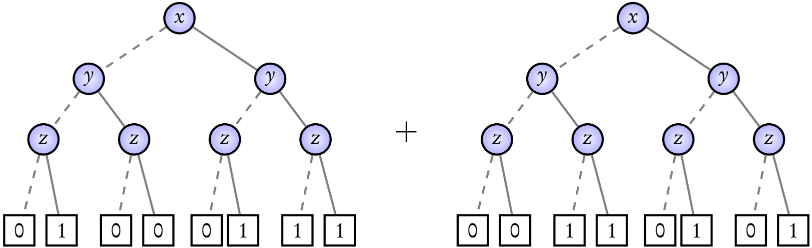

x
y

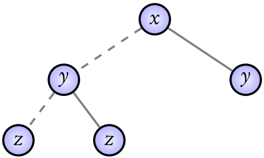

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

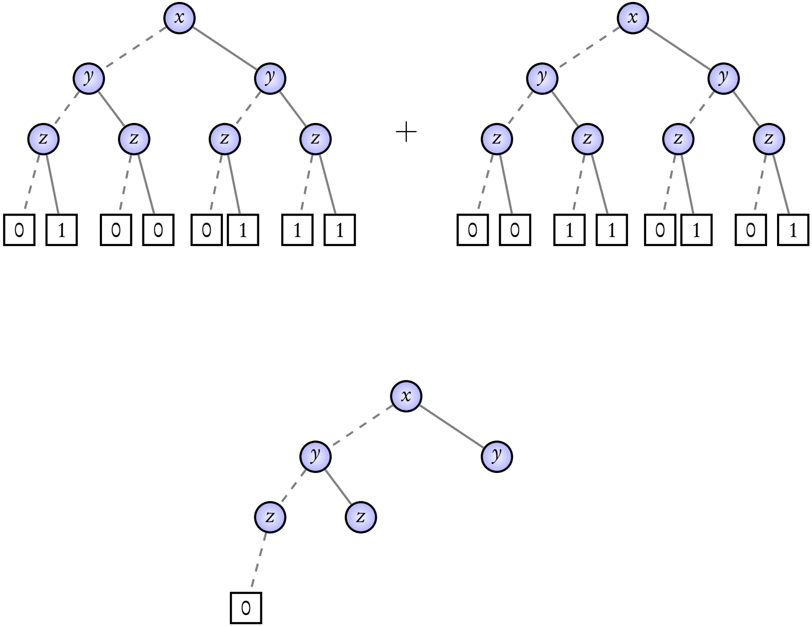

x
y y

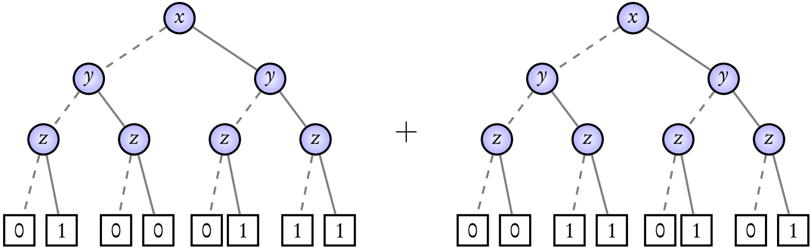

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

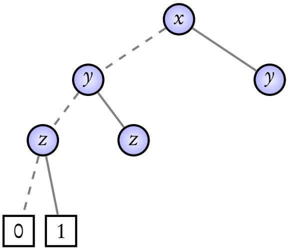

x
y y
z

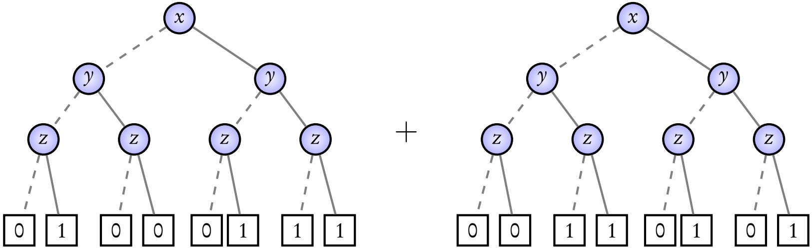

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

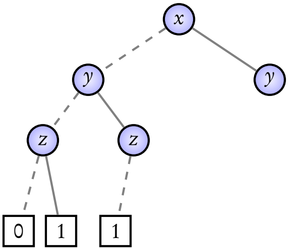

x
y y
z z

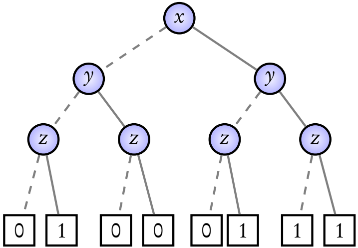

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1
x
y y
z z
0



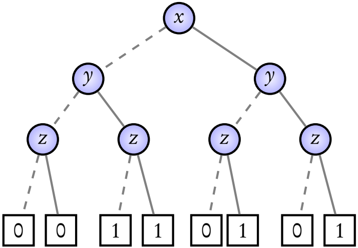


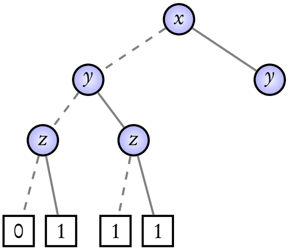

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

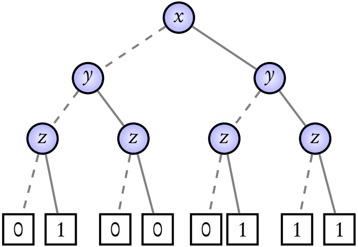

x
y y
z z
0 1 1


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

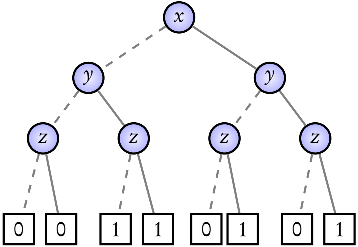
+

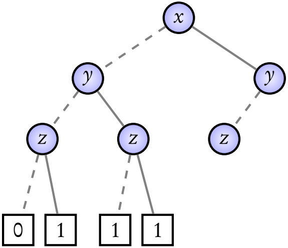

x
y y
z z z z
0 0 1 1 0 1 0 1

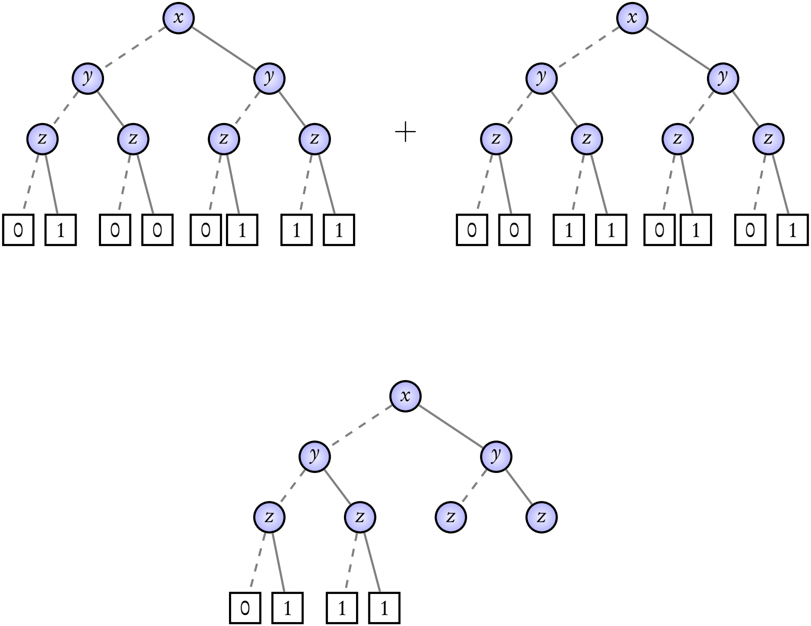

x
y y
z z
0 1 1 1

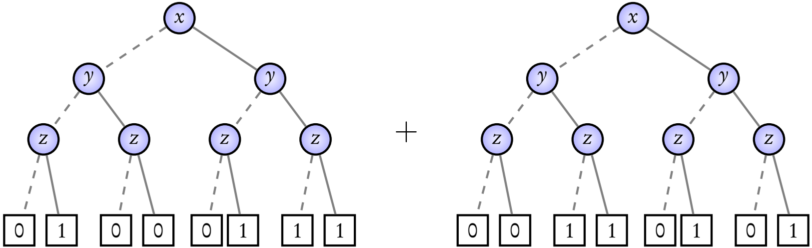

x
y y
z z z z
0 1 0 0 0 1 1 1

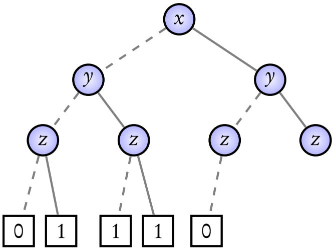
$+$

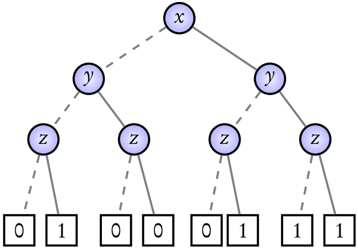

x
y y
z z z z
0 0 1 1 0 1 0 1


x
y y
z z z
0 1 1 1

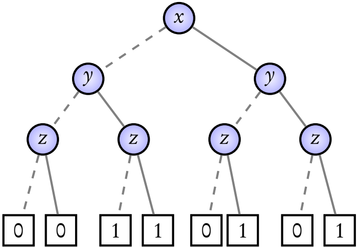

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1
x
y y
z z z z
0 1 1 1

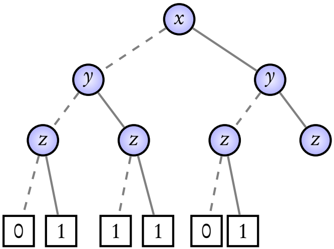

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1

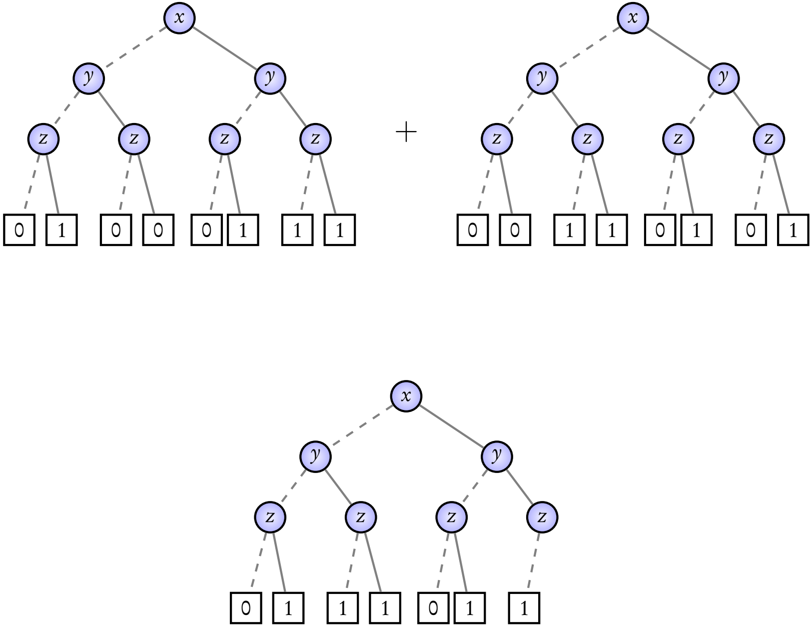

x
y y
z z z z
0 1 1 1 0

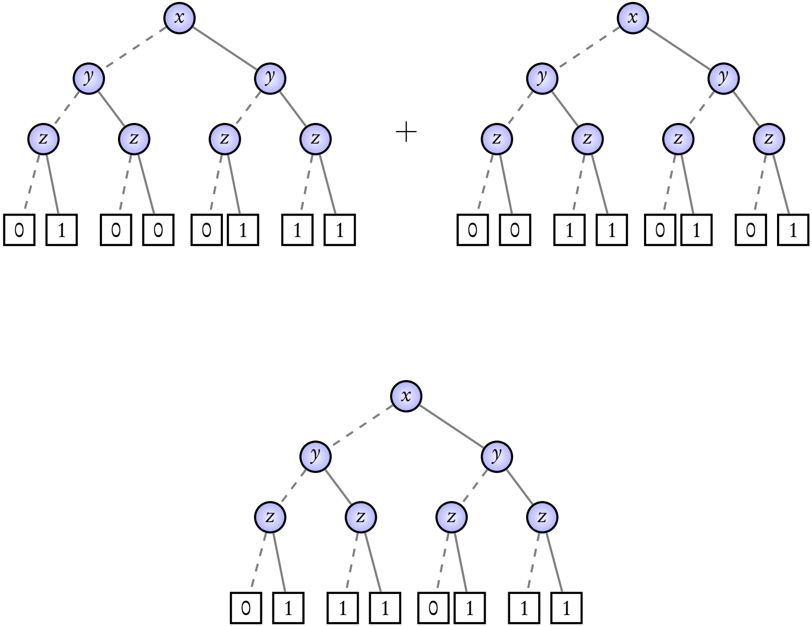

x
y y
z z z z
0 1 0 0 0 1 1 1

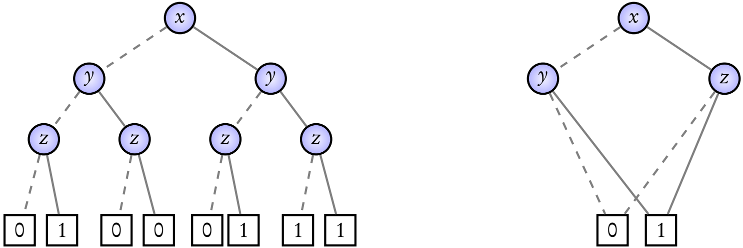
$+$

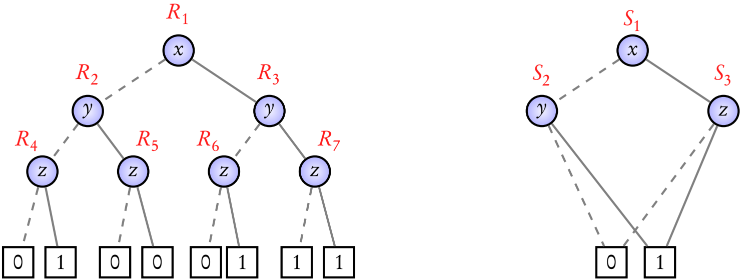

x
y y
z z z z
0 0 1 1 0 1 0 1

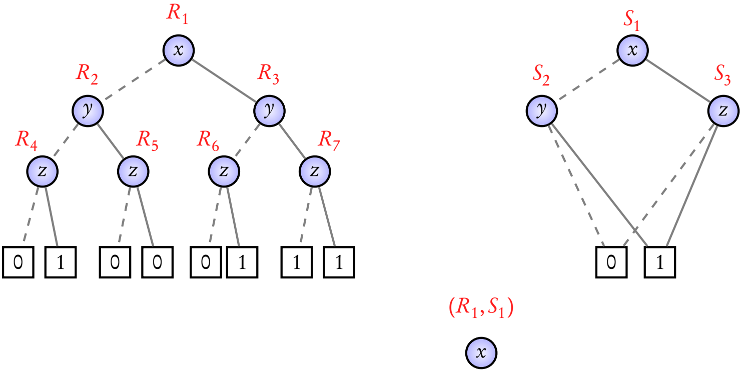

x
y y
z z z z
0 1 1 1 0 1

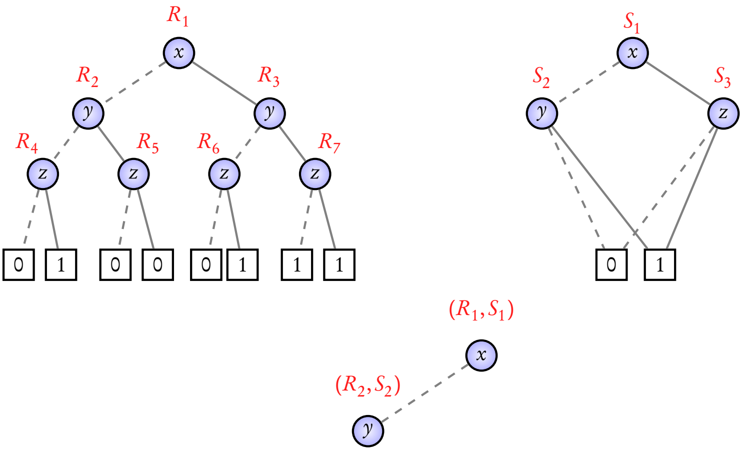

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1
x
y y
z z z z
0 1 1 1 0 1 1

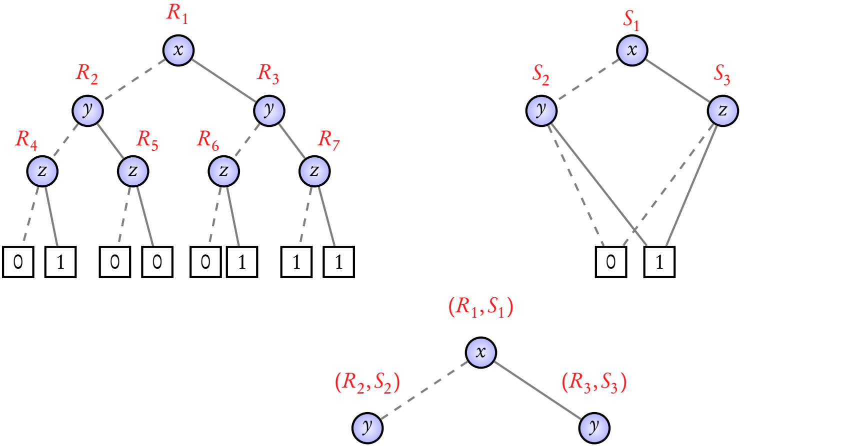

x
y y
z z z z
0 1 0 0 0 1 1 1
+
x
y y
z z z z
0 0 1 1 0 1 0 1
x
y y
z z z z
0 1 1 1 0 1 1 1

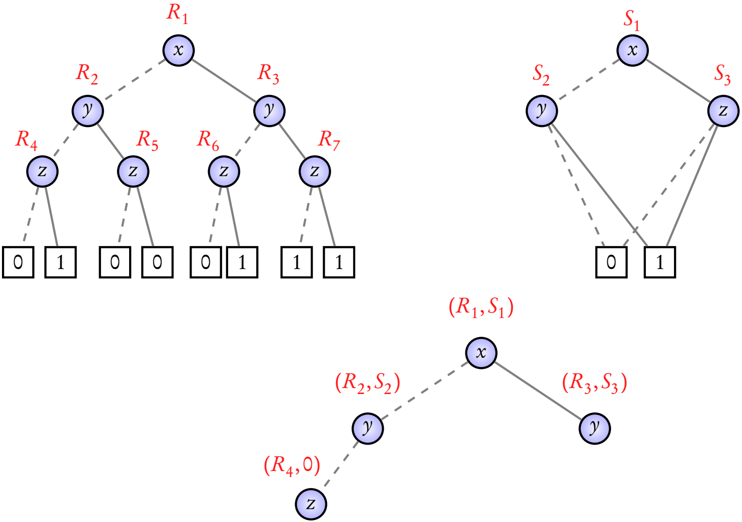

x
y
y
z
z
z
z
0
1
0
0
0
1
1
1
x
y
z
0
1

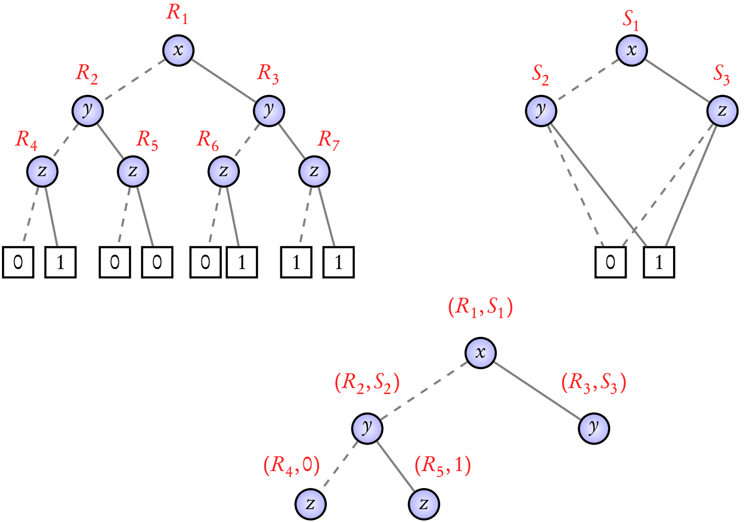

R_1
x
R_2
y
R_3
y
R_4
z
R_5
z
R_6
z
R_7
z
0 1 0 0 0 1 1 1
S_1
x
S_2
y
S_3
z
0 1


R_1
R_2
R_3
R_4
R_5
R_6
R_7
x
y
y
z
z
z
z
0
1
0
0
0
1
1
1
S_1
S_2
S_3
x
y
z
0
1
(R_1, S_1)
x

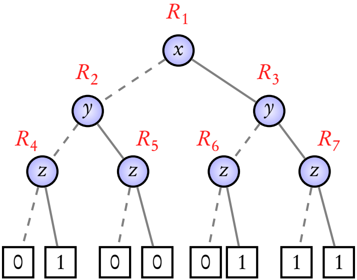

R1
R2
R3
R4
R5
R6
R7
S1
S2
S3
(R1, S1)
(R2, S2)
x
y
z
0
1


R_1
R_2
R_3
R_4
R_5
R_6
R_7
x
y
y
z
z
z
z
0
1
0
0
0
1
1
1
S_1
S_2
S_3
x
y
z
0
1
(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
x
y
y

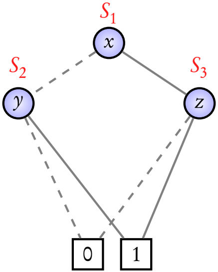

R_1
x
R_2
R_3
y
y
R_4
R_5
R_6
R_7
z
z
z
z
0
1
0
0
0
1
1
1
S_1
x
S_2
S_3
y
z
0
1
(R_1, S_1)
x
(R_2, S_2)
(R_3, S_3)
y
y
$(R_4, 0)$
z



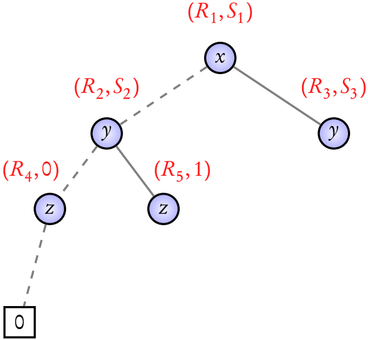
R_1
R_2
R_3
R_4
R_5
R_6
R_7
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
$(R_4, 0)$
$(R_5, 1)$



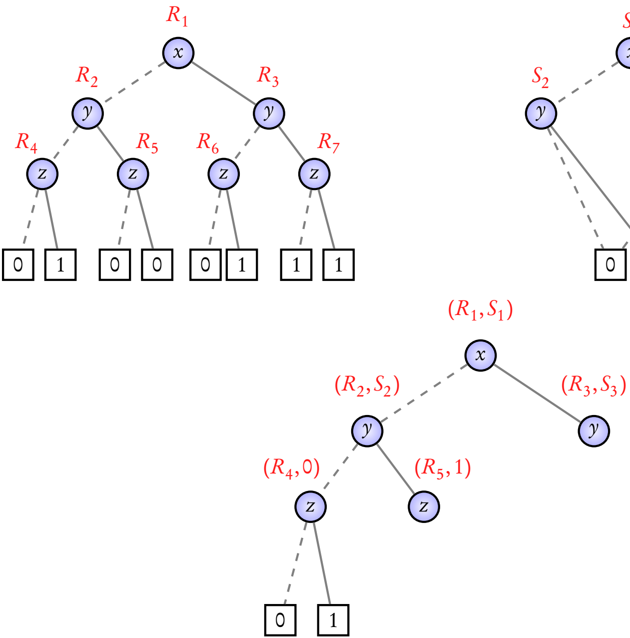
R_1
x
R_2
y
R_3
y
R_4
z
R_5
z
R_6
z
R_7
z
0
1
0
0
0
1
1
1

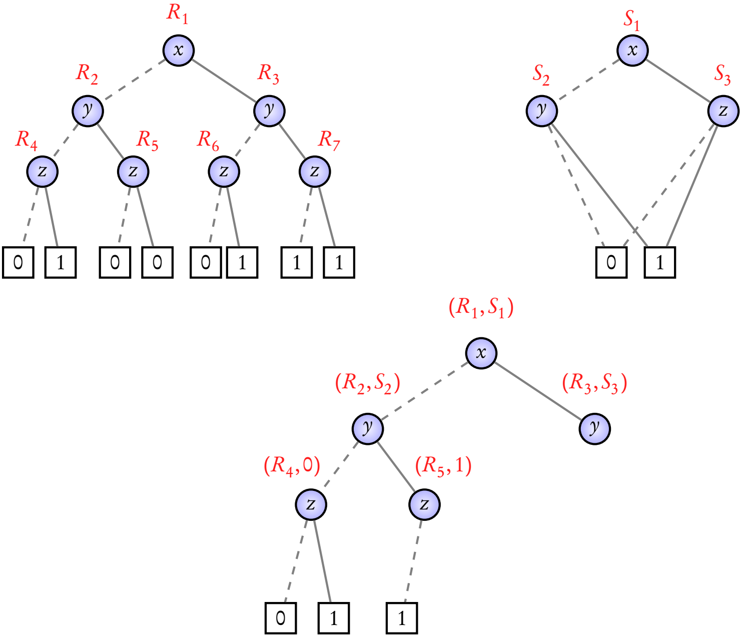


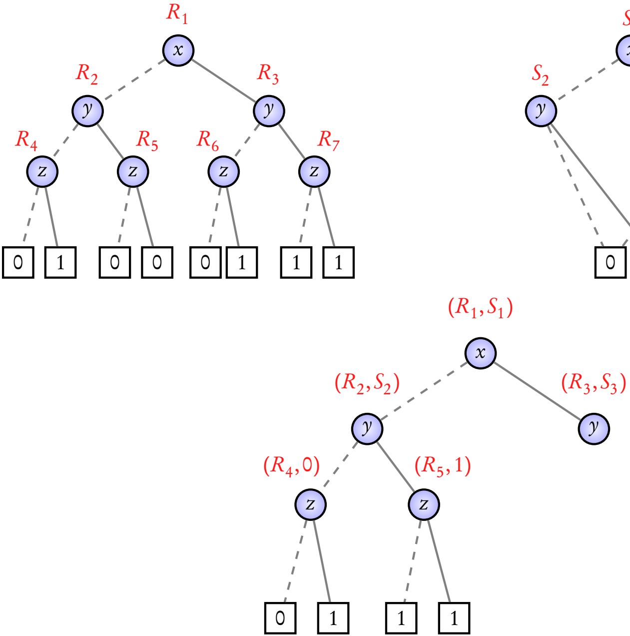
S_1
x
S_2
y
S_3
z
0
1



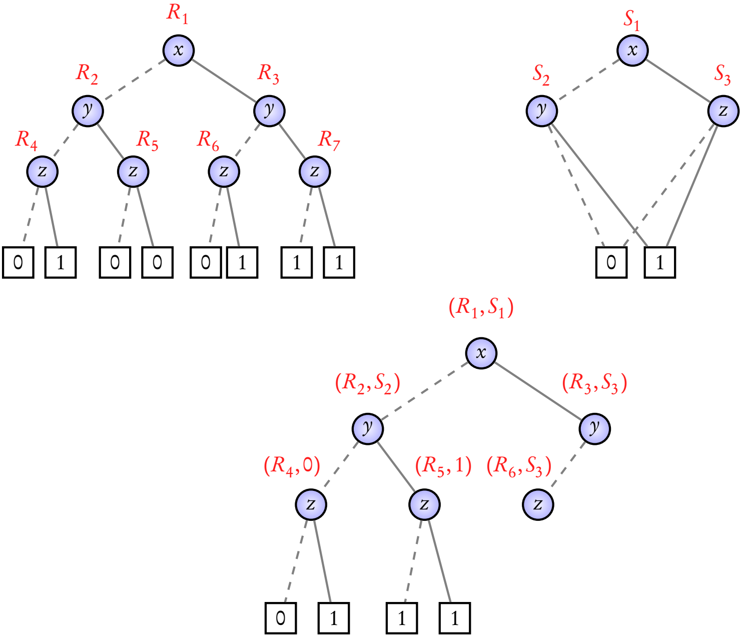
(R_1, S_1)
x
(R_2, S_2)
y
(R_3, S_3)
y
$(R_4, 0)$
z
$(R_5, 1)$
z
0

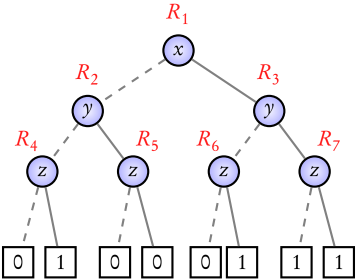

R1
R2
R3
R4
R5
R6
R7
x
y
y
z
z
z
z
0
1
0
0
0
1
1
1
S1
S2
S3
x
y
z
0
1
(R1, S1)
(R2, S2)
(R3, S3)
(R4, 0)
(R5, 1)
x
y
y
z
z
0
1

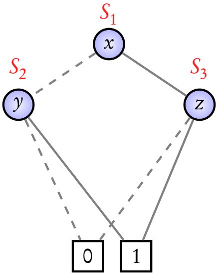

R1
x
R2
R3
y
y
R4
R5
R6
R7
z
z
z
z
0 1 0 0 0 1 1 1
S1
x
S2
S3
y
z
0 1
(R1, S1)
x
(R2, S2)
(R3, S3)
y
y
(R4, 0)
(R5, 1)
z
z
0 1 1

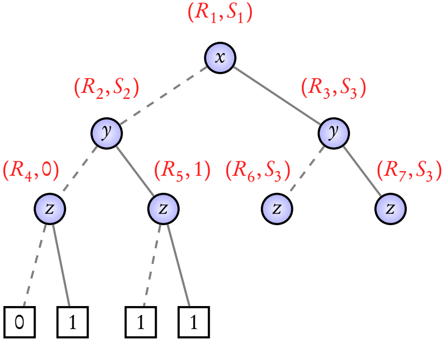


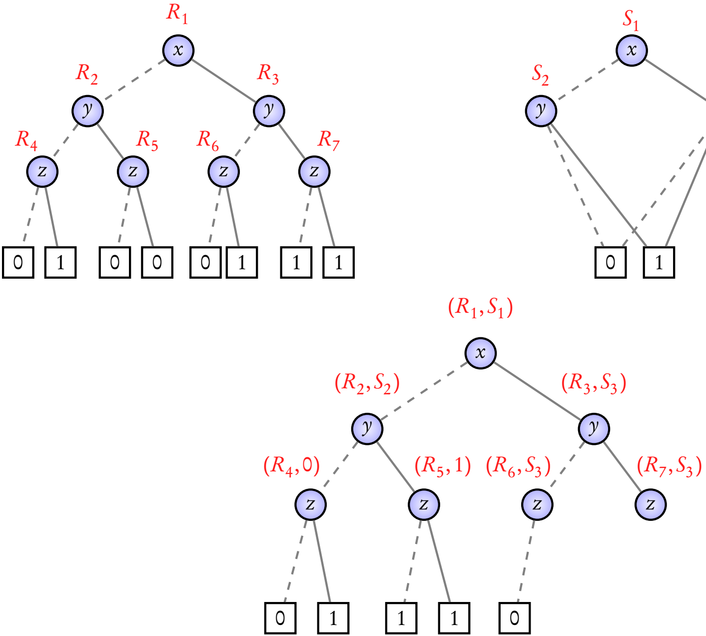
R_1
R_2
R_3
R_4
R_5
R_6
R_7
x
y
y
z
z
z
z
0 1 0 0 0 1 1 1
S_1
S_2
S_3
x
y
z
0 1
(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
$(R_4, 0)$
$(R_5, 1)$
x
y
y
z
z
0 1 1 1

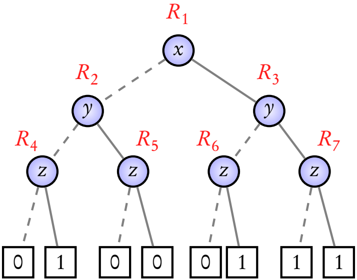

R1
x
R2
R3
y
y
R4
R5
R6
R7
z
z
z
z
0
1
0
0
0
1
1
1
S1
x
S2
S3
y
z
0
1
(R1, S1)
x
(R2, S2)
(R3, S3)
y
y
(R4, 0)
(R5, 1)
(R6, S3)
z
z
z
0
1
1
1

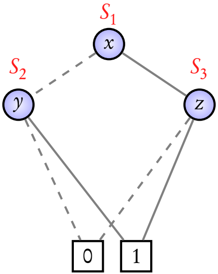

R_1
x
R_2
y
R_3
y
R_4
R_5
R_6
R_7
z
z
z
z
0
1
0
0
0
1
1
1

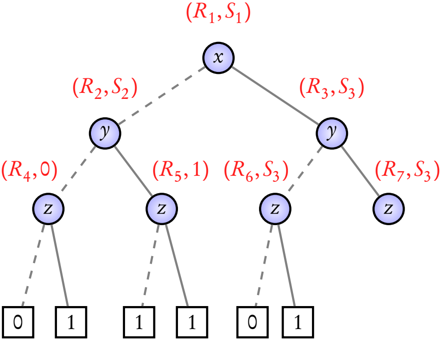

S_1
x
S_2
y
S_3
z
0
1

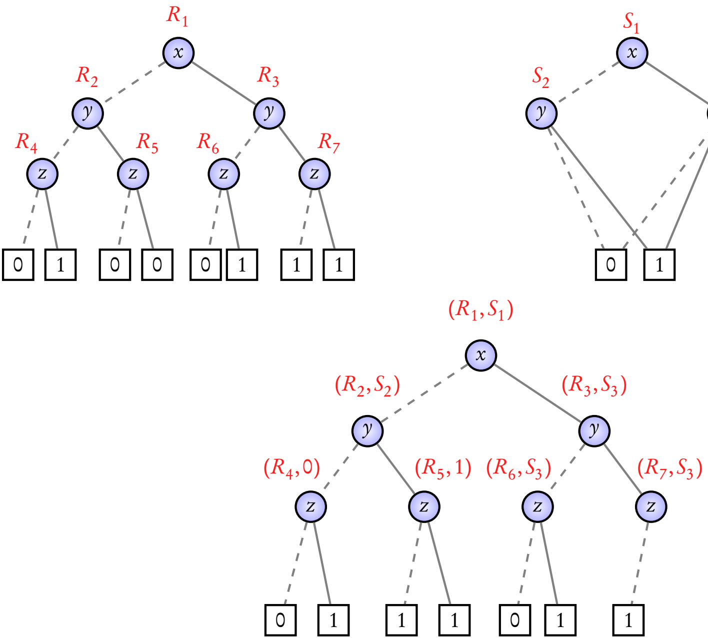

(R_1, S_1)
x
(R_2, S_2)
y
(R_3, S_3)
y
$(R_4, 0)$
$(R_5, 1)$
(R_6, S_3)
(R_7, S_3)
z
z
z
z
0
1
1
1


R_1
R_2
R_3
R_4
R_5
R_6
R_7
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
$(R_4, 0)$
$(R_5, 1)$
(R_6, S_3)
(R_7, S_3)
x
y
z
0
1

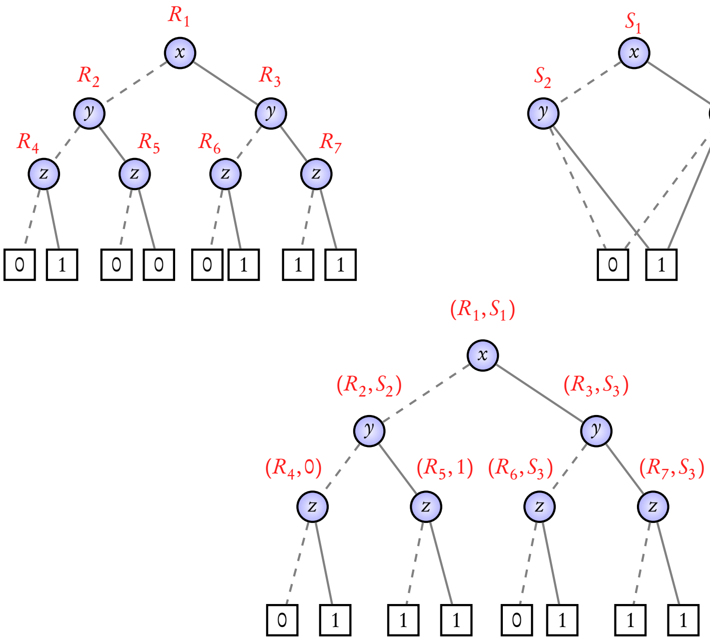

R_1
R_2
R_3
R_4
R_5
R_6
R_7
x
y
y
z
z
z
z
0
1
0
0
0
1
1
1

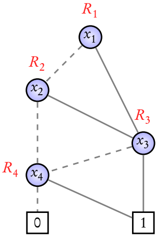

S_1
S_2
S_3
x
y
z
0
1

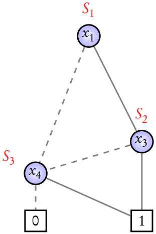

(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
$(R_4, 0)$
$(R_5, 1)$
(R_6, S_3)
(R_7, S_3)
x
y
y
z
z
z
z
0
1
1
1
0
1

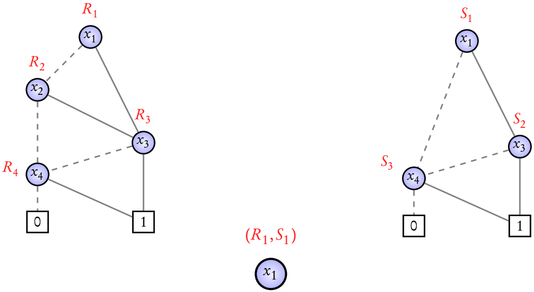


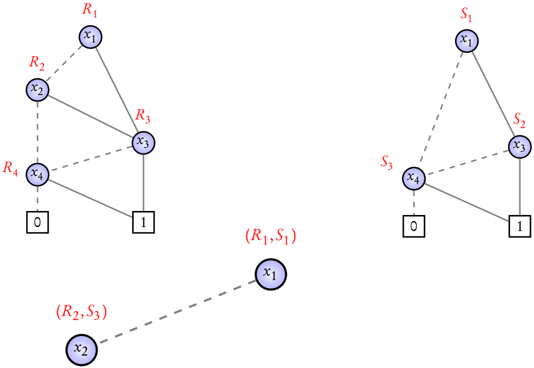
R_1
R_2
R_3
R_4
R_5
R_6
R_7
x
y
z
0
1
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_2)
(R_3, S_3)
$(R_4, 0)$
$(R_5, 1)$
(R_6, S_3)
(R_7, S_3)

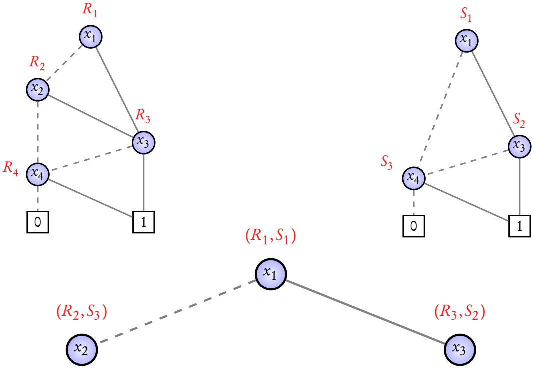

R1
x
R2
R3
y
y
R4
R5
R6
R7
z
z
z
z
0 1 0 0 0 1 1 1
S1
x
S2
S3
y
z
0 1
(R1, S1)
x
(R2, S2)
(R3, S3)
y
y
(R4, 0)
(R5, 1)
(R6, S3)
(R7, S3)
z
z
z
z
0 1 1 1 0 1 1 1

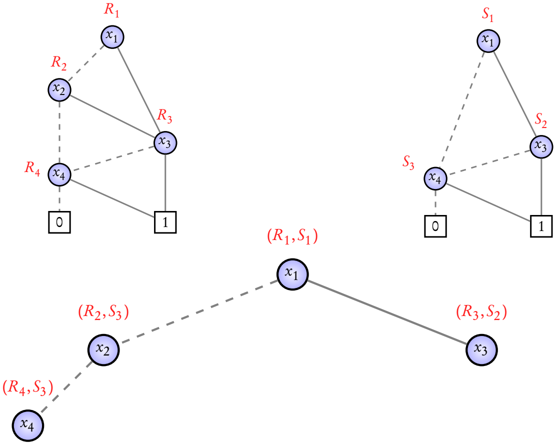

R1
x1
R2
x2
R3
x3
R4
x4
0
1


S1
x1
S2
x3
S3
x4
0
1

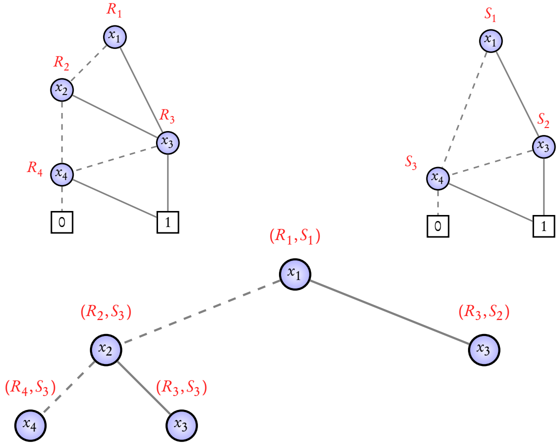

R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
x_1
x_3
x_4
0
1
(R_1, S_1)
x_1

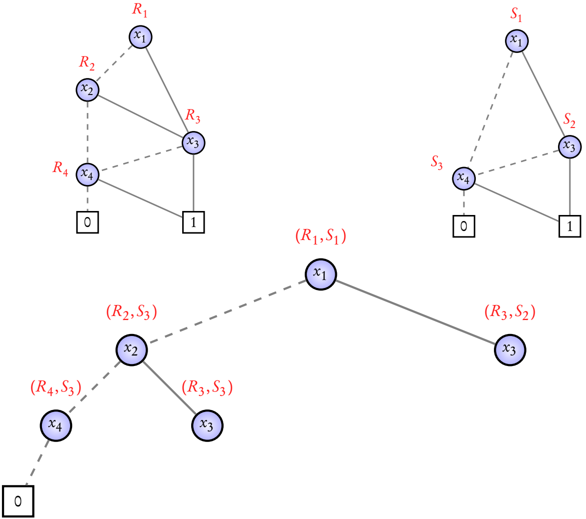

R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2

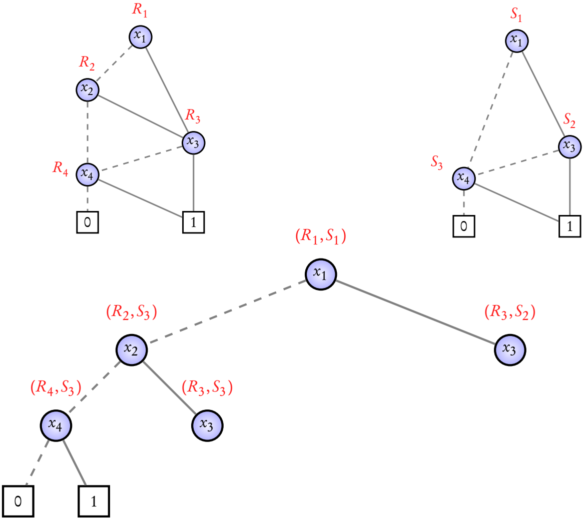

R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3



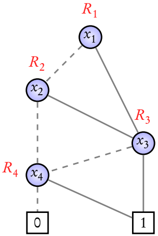
R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4

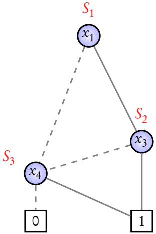

R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
x_1
x_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3

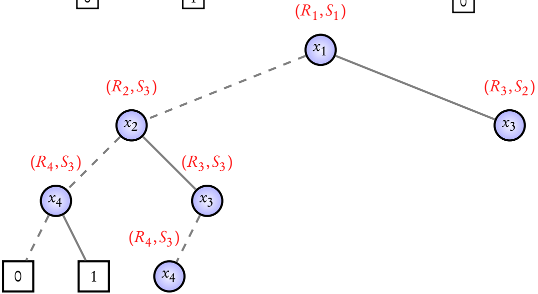

R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_3)
(R_3, S_2)
(R_4, S_3)
(R_3, S_3)

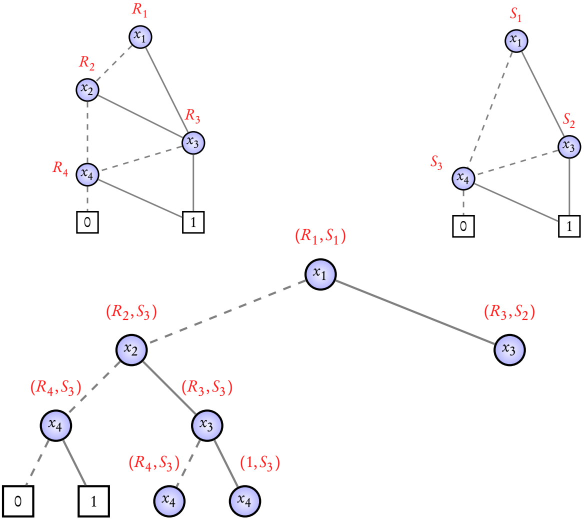


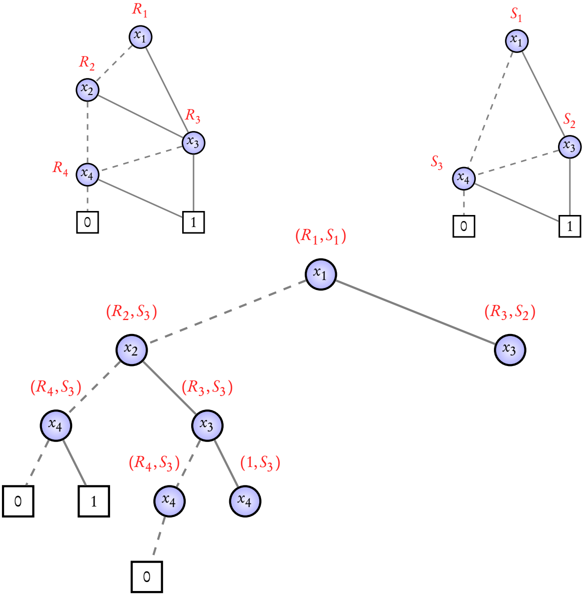
R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3
0
1

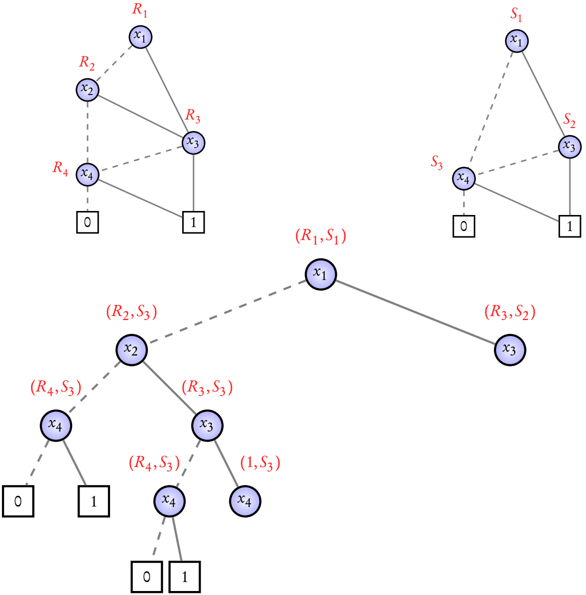

R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1

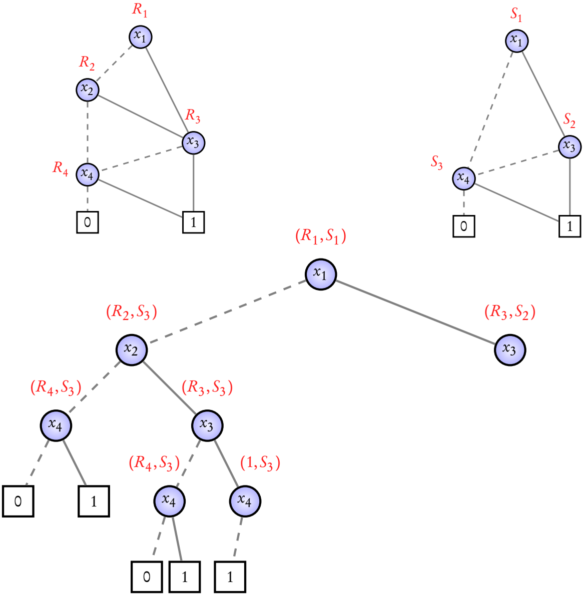

S_1
x_1
S_2
x_3
S_3
x_4
0
1


(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3
(R_4, S_3)
0
1
x_4

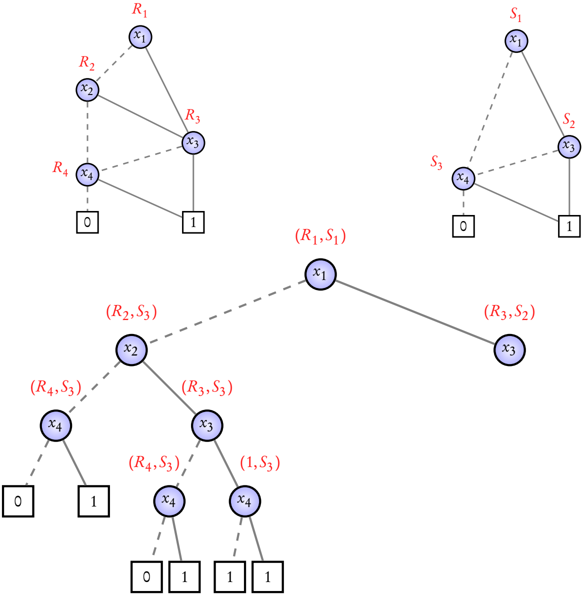

R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_3)
(R_3, S_2)
(R_4, S_3)
(R_3, S_3)
(R_4, S_3)
$(1, S_3)$

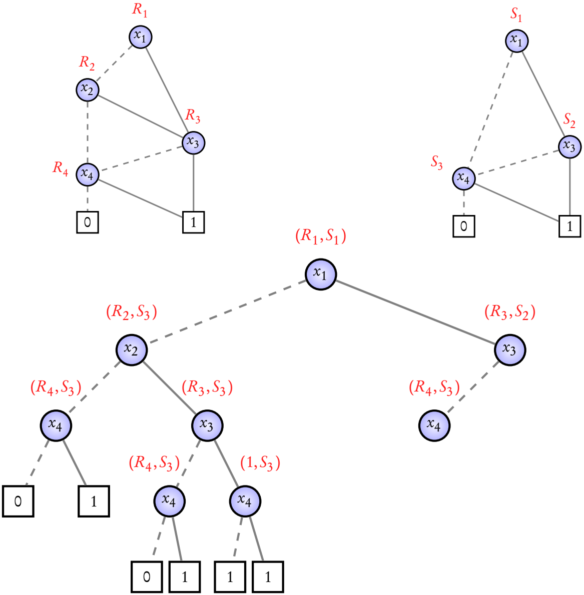

R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_3)
(R_3, S_2)
(R_4, S_3)
(R_3, S_3)
(R_4, S_3)
$(1, S_3)$

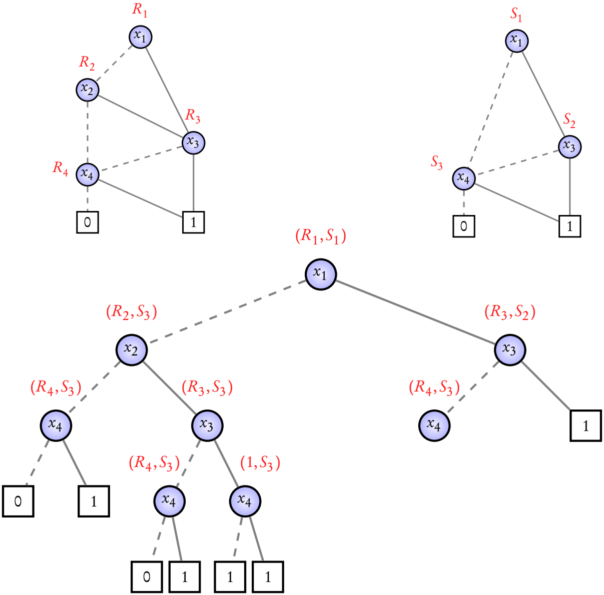

R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3
0
1
(R_4, S_3)
x_4
$(1, S_3)$
x_4
0
1

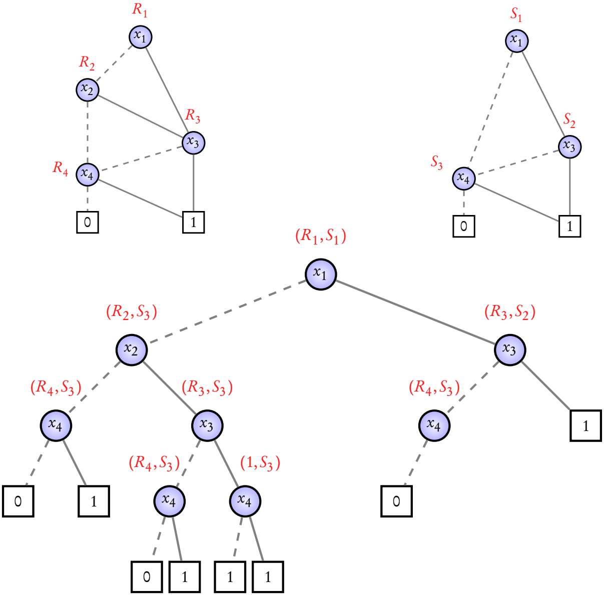


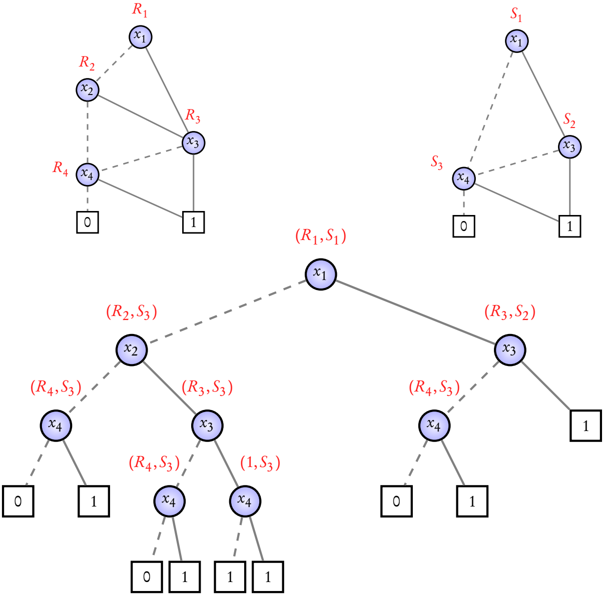
R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3
0
1
(R_4, S_3)
x_4
$(1, S_3)$
x_4
0
1
1

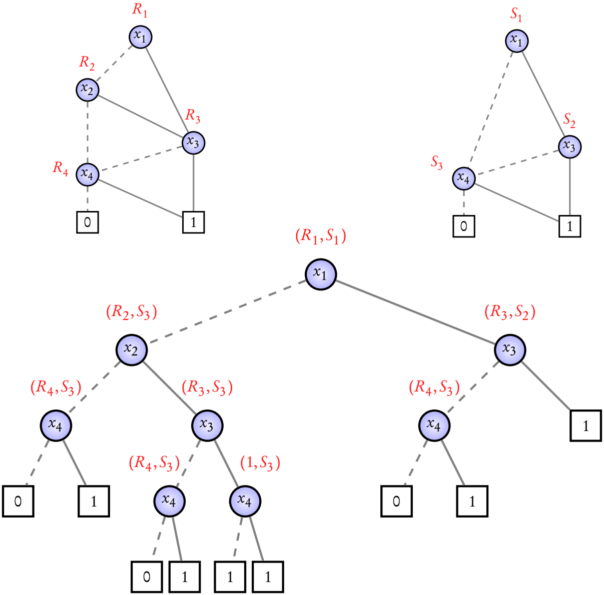

R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1,S_1)
x_1
(R_2,S_3)
x_2
(R_3,S_2)
x_3
(R_4,S_3)
x_4
(R_3,S_3)
x_3
0
1
(R_4,S_3)
x_4
$(1,S_3)$
x_4
0
1
1
1


R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
x_1
S_2
x_3
S_3
x_4
0
1
(R_1, S_1)
x_1
(R_2, S_3)
x_2
(R_3, S_2)
x_3
(R_4, S_3)
x_4
(R_3, S_3)
x_3
(R_4, S_3)
x_4
0
1
(R_4, S_3)
x_4
$(1, S_3)$
x_4
0
1
1
1


R_1
R_2
R_3
R_4
x_1
x_2
x_3
x_4
0
1
S_1
S_2
S_3
x_1
x_3
x_4
0
1
(R_1, S_1)
(R_2, S_3)
(R_3, S_2)
(R_4, S_3)
(R_3, S_3)
(R_4, S_3)
(R_4, S_3)
$(1, S_3)$


R1
x1
R2
x2
R3
x3
R4
x4
0
1
S1
x1
S2
x3
S3
x4
0
1
(R1,S1)
x1
(R2,S3)
x2
(R3,S2)
x3
(R4,S3)
x4
(R3,S3)
x3
(R4,S3)
x4
1
0
1
(R4,S3)
x4
(1,S3)
x4
0
0
1
1
1


R_1
x_1
R_2
x_2
R_3
x_3
R_4
x_4
0
1
S_1
S_2
S_3
(R_1, S_1)
(R_2, S_3)
(R_3, S_2)
(R_4, S_3)
(R_3, S_3)
$(1, S_3)$



Reduce the resulting OBDD

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$\texttt{apply}(+, r, s)$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$\texttt{apply}(+, r, s)$

- If both r and s are terminals, create a terminal node $r + s$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$\texttt{apply}(+, r, s)$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$
- If r is x_i node and s is a terminal or an x_j node with $j > i$, create an x_i node with:

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$
- If r is x_i node and s is a terminal or an x_j node with $j > i$, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), s)$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$
- If r is x_i node and s is a terminal or an x_j node with $j > i$, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), s)$
  - right child: $\texttt{apply}(+, \texttt{right}(r), s)$

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$$\texttt{apply}(+, r, s)$$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$
- If r is x_i node and s is a terminal or an x_j node with $j > i$, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), s)$
  - right child: $\texttt{apply}(+, \texttt{right}(r), s)$
- If s is x_i node and r is a terminal: similar to Case 3

Algorithm for $\text{OBDD}_1 + \text{OBDD}_2$

$\texttt{apply}(+, r, s)$

- If both r and s are terminals, create a terminal node $r + s$
- If both r and s are x_i nodes, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), \texttt{left}(s))$
  - right child: $\texttt{apply}(+, \texttt{right}(r), \texttt{right}(s))$
- If r is x_i node and s is a terminal or an x_j node with $j > i$, create an x_i node with:
  - left child: $\texttt{apply}(+, \texttt{left}(r), s)$
  - right child: $\texttt{apply}(+, \texttt{right}(r), s)$
- If s is x_i node and r is a terminal: similar to Case 3

$\text{OBDD}_1 + \text{OBDD}_2$: $\texttt{apply}(+, \text{root}_1, \text{root}_2)$ and then reduce

Operations on BDDs

- OR: $\texttt{apply}(+\,,\mathrm{root}_1,\mathrm{root}_2)$
- AND: $\texttt{apply}(\cdot\,,\mathrm{root}_1,\mathrm{root}_2)$
- XOR: $\texttt{apply}(\mathrm{XOR}\,,\mathrm{root}_1,\mathrm{root}_2)$
- NOT: Use the fact that $\overline{f} = f \text{ XOR } 1$

OBDDs

Reduction algorithm

Operations on OBDDs