I/O-efficient Manipulation of Binary Decision Diagrams

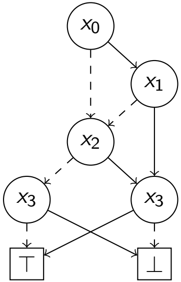
Steffan Christ Sølvsten

S. C. Sølvsten, J. van de Pol, A. B. Jakobsen, and M. W. B. Thomasen.
Adiar: Binary Decision Diagrams in External Memory. 2022

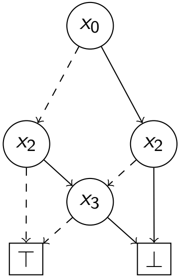
AARHUS
UNIVERSITY

Contents

Contents



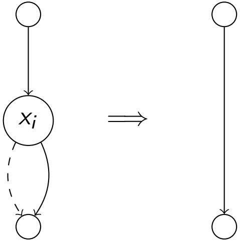
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



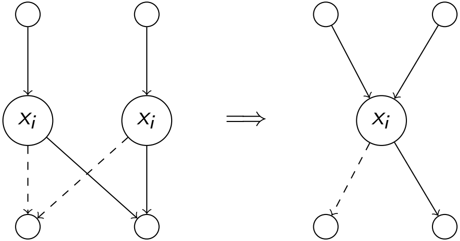
(b) $\neg(x_0 \ ? \ x_2 \wedge x_3 : x_2 \wedge x_3)$

Examples of (Reduced Ordered) Binary Decision Diagrams.

Theorem (Bryant '86)
For a fixed variable order, if one exhaustively applies the two rules below, then one obtains the Reduced OBDD, which is a unique canonical form of the function.



(1) Remove redundant nodes

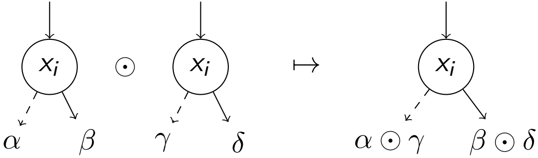


(2) Merge duplicate nodes

`bdd_apply(f,g,⊙)`

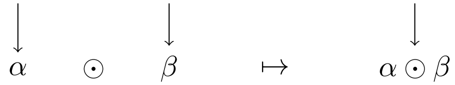
Base Case ($f, g \in \mathbb{B}$):

Inductive Case:

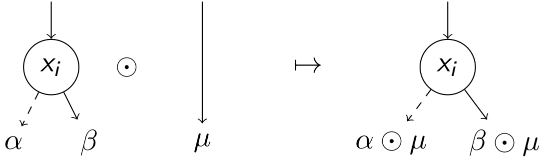


`bdd_apply(f,g,⊙)`

Base Case ($f, g \in \mathbb{B}$):



Inductive Case:



`bdd_apply(f,g,`$\odot$`)`

Let N_f, N_g be the size of the BDDs for f and g.

Let T be the $O(N_f \cdot N_g)$ size of the BDD for $f \odot g$.

Theorem
`bdd_apply(f,g,`$\odot$`)` *runs in* $O(N_f + N_g + T)$ *time*

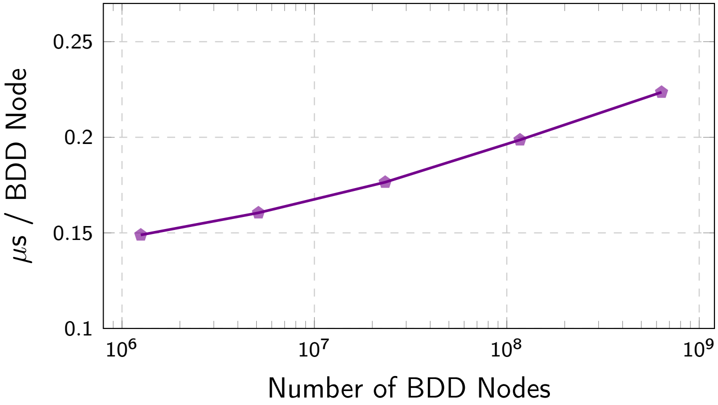
- Memoisation (*Computation Cache*) ensures each (t_f, t_g) is only computed once.
- Reduction Rules can be maintained with a `make_node(`i`,`t`,`e`)` in $O(1)$ time.
  1. Redundancy is resolved with an if-statement.
  2. Duplication is avoided with a hash table (*Unique Node Table*).

Corollary
`bdd_apply(f,g,`$\odot$`)` *runs in* $O(1)$ *time per BDD node.*

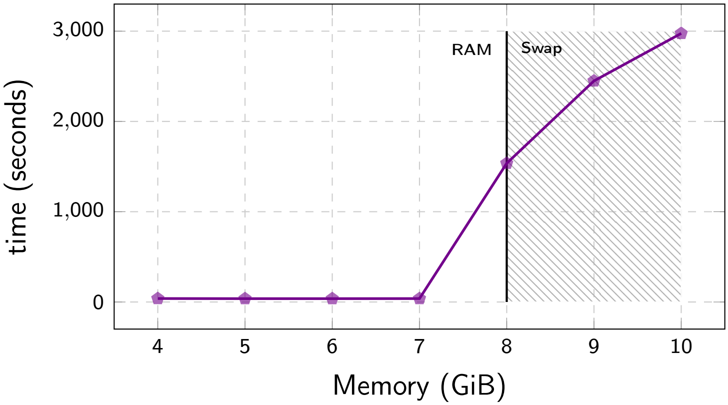
Adiar

I/O-efficient Decision Diagrams

github.com/ssoelvsten/adiar

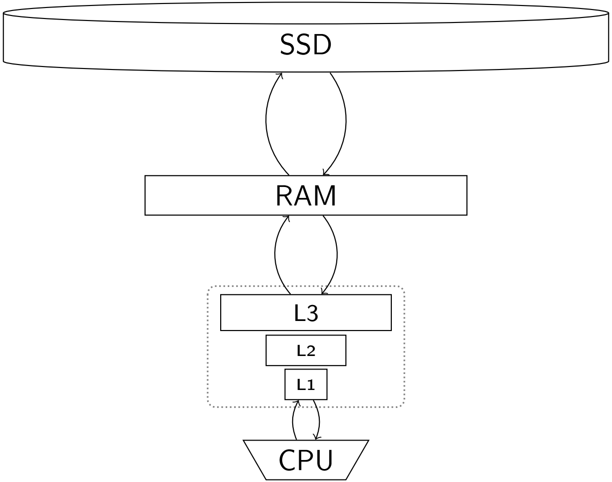


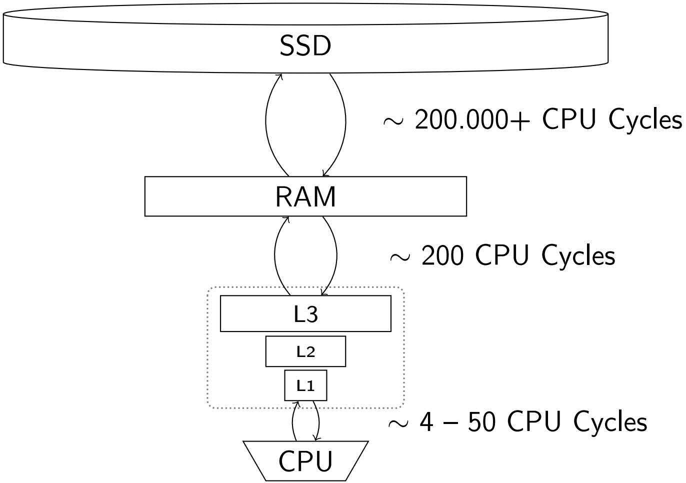
Running time of *BuDDy* for the *N*-Queens problem.

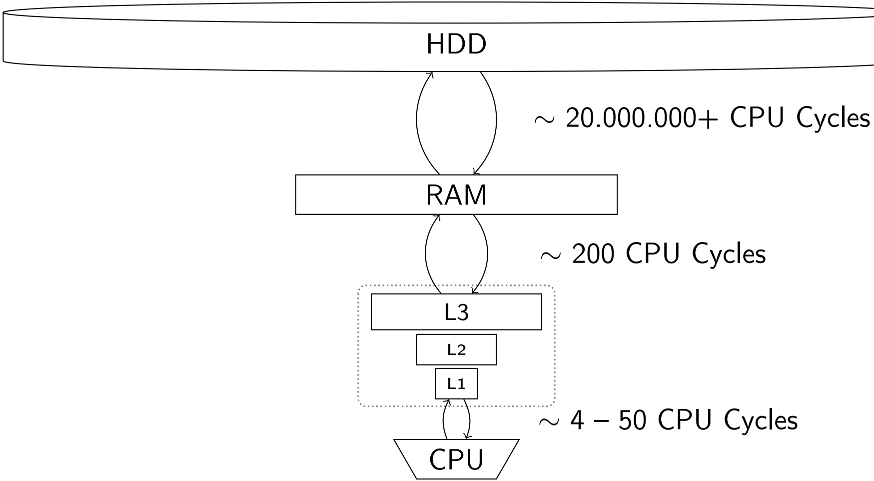


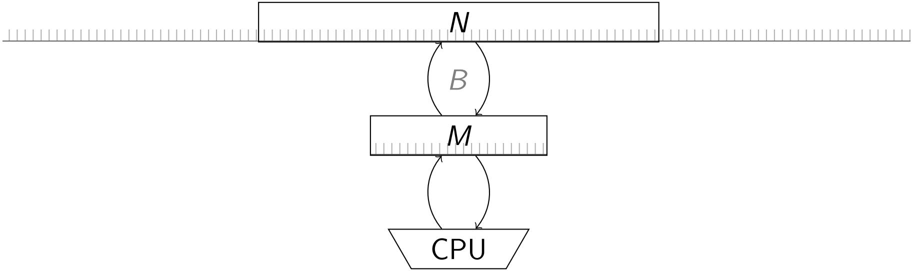
Running time of *BuDDy* for *3D Tic-Tac-Toe* with $N = 21$.

Contents


SSD
RAM
L3
L2
L1
CPU


SSD
∼ 200.000+ CPU Cycles
RAM
∼ 200 CPU Cycles
L3
L2
L1
∼ 4 – 50 CPU Cycles
CPU


HDD
~ 20.000.000+ CPU Cycles
RAM
~ 200 CPU Cycles
L3
L2
L1
~ 4 – 50 CPU Cycles
CPU



The I/O model by Aggarwal and Vitter '87

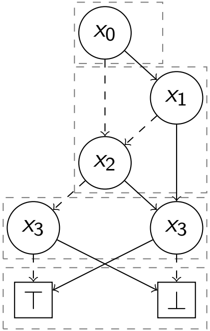
For any realistic values of N, M, and B we have that

$$N/B \quad < \quad \mathrm{sort}(N) \triangleq N/B \cdot \log_{M/B} N/B \quad \ll \quad N \;,$$

Theorem (Aggarwal and Vitter '87)
N elements can be sorted in $\Theta(sort(N))$ I/Os.

Theorem (Arge '95)
A Priority Queue can do N insertions and extractions in $\Theta(sort(N))$ I/Os.

CountPaths : *Example*

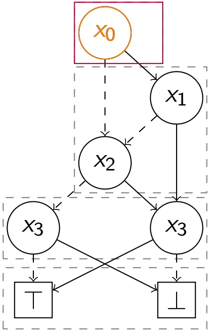


(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
0	0

CountPaths : *Example*

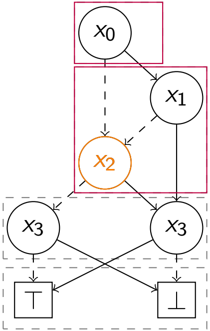


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
1	1

CountPaths : *Example*

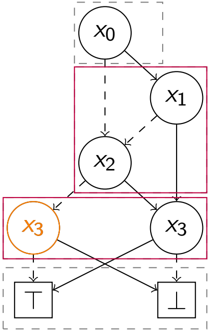


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
2	2

CountPaths : *Example*

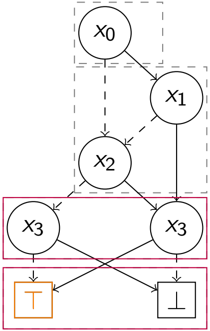


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
3	3

CountPaths : *Example*

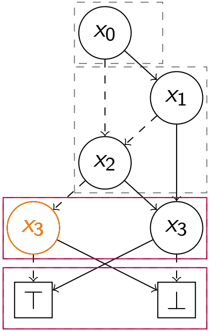


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
4	3

CountPaths : *Example*

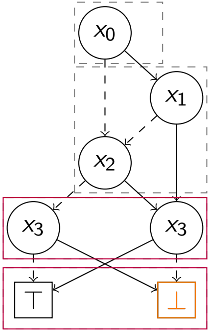


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
4	3

CountPaths : *Example*

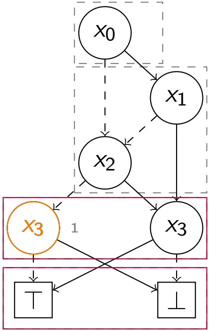


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
4	3

CountPaths : *Example*

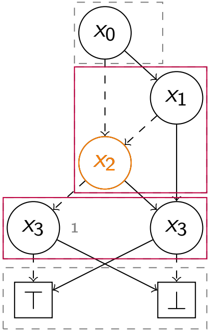


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
4	3

CountPaths : *Example*

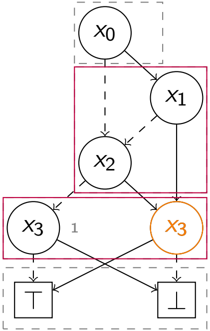


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
5	3

CountPaths : *Example*

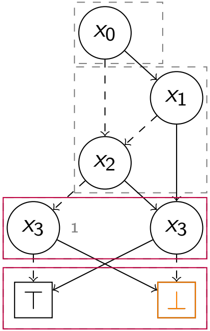


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
5	4

CountPaths : *Example*

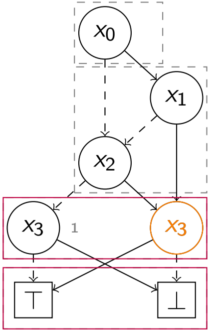


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
6	4

CountPaths : *Example*

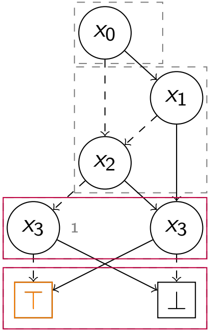


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
6	4

CountPaths : *Example*

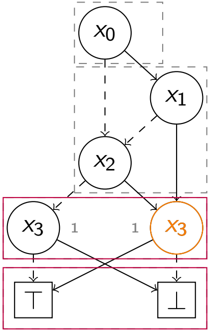


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\; B = 2$$

node I/Os	cache lookups
6	4

CountPaths : *Example*

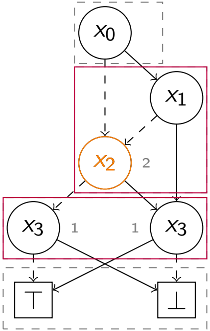


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
6	4

CountPaths : *Example*

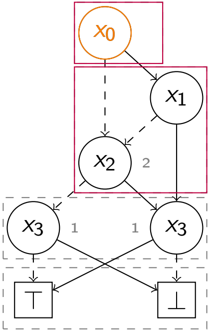


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
7	4

CountPaths : *Example*

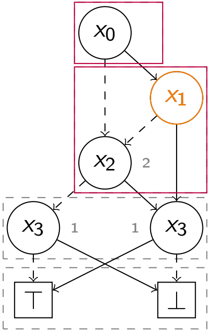


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
8	4

CountPaths : *Example*

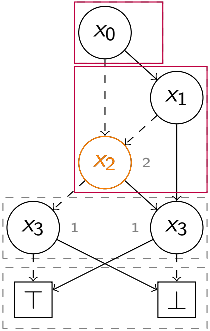


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
8	5

CountPaths : *Example*

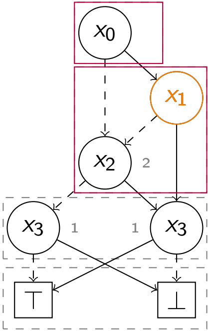


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
8	6

CountPaths : *Example*

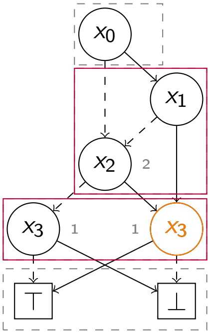


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
8	6

CountPaths : *Example*

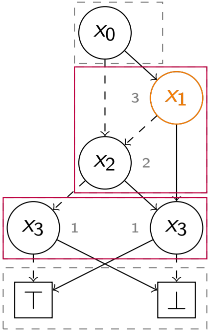


(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
9	7

CountPaths : *Example*

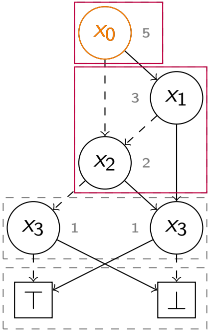


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, \; B = 2$$

node I/Os	cache lookups
9	7

CountPaths : *Example*

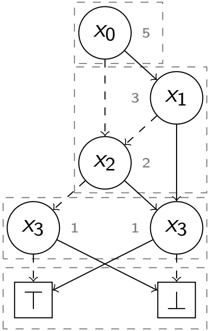


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4,\ B = 2$$

node I/Os	cache lookups
10	7

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$M = 4,\ B = 2$

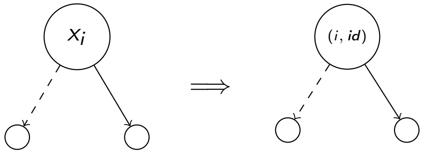
node I/Os	cache lookups
10	7

Algorithm	Time Complexity
`bdd_pathcount`	$O(N_f)$
`bdd_not`	$O(N_f)$
`bdd_restrict`	$O(N_f)$
`bdd_apply`	$O(N_f \cdot N_g)$
`bdd_equal`	$O(1)$

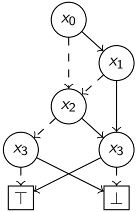
Algorithm	I/O-Complexity
`bdd_pathcount`	$O(N_f)$
`bdd_not`	$O(N_f)$
`bdd_restrict`	$O(N_f)$
`bdd_apply`	$O(N_f \cdot N_g)$
`bdd_equal`	$O(1)$

Contents

Contents

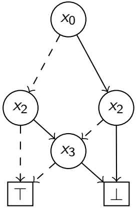


$$(i_1, id_1) < (i_2, id_2) \equiv i_1 < i_2 \vee (i_1 = i_2 \wedge id_i < id_j)$$



$$
\begin{array}{lcl}
[& ((0,0),(2,0),(1,0)) & , \\
 & ((1,0),(2,0),(3,1)) & , \\
 & ((2,0),(3,0),(3,1)) & , \\
 & ((3,0),\top,\bot) & , \\
 & ((3,1),\bot,\top) &]
\end{array}
$$

(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

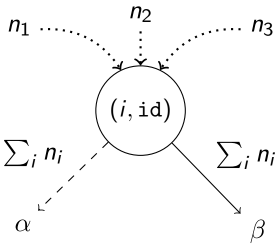


$$
\begin{array}{lcl}
[& ((0,0),(2,0),(2,1)) & , \\
 & ((2,0),\top,(3,0)) & , \\
 & ((2,1),(3,0),\bot) & , \\
 & ((3,0),\top,\bot) &]
\end{array}
$$

(b) $\neg(x_0 \ ? \ x_2 \wedge x_3 : x_2 \wedge x_3)$

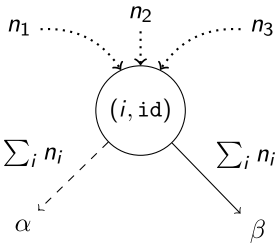
Node-based representation of prior shown BDDs

CountPaths



Idea

Count the number of in-going paths to each node.

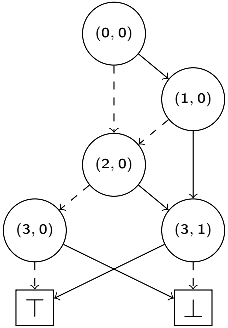


Time-Forward Processing

Defer work with Q_{count} : $\texttt{PriorityQueue}\langle(s \to t, \quad \mathbb{N})\rangle$ sorted on t in ascending order.

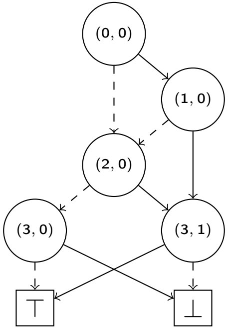
$$((i, \texttt{id}) \xrightarrow{\perp} \alpha, \quad \sum_i n_i), \qquad ((i, \texttt{id}) \xrightarrow{\top} \beta, \quad \sum_i n_i)$$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

CountPaths : *Example*



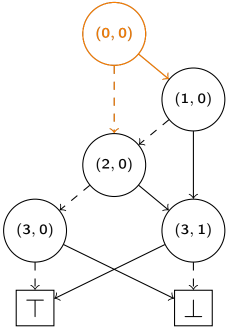
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

[

]

CountPaths : *Example*



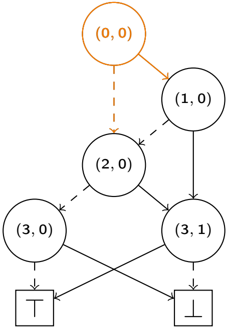
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

[

]

CountPaths : *Example*

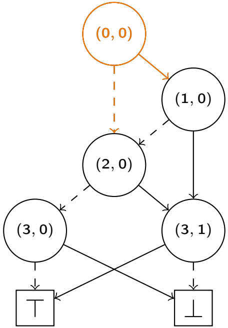


(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$

Priority Queue: Q_{count}:

$$[\quad ((0,0) \xrightarrow{\top} (1,0),\quad 1)\quad ,$$
$$((0,0) \xrightarrow{\perp} (2,0),\quad 1)\quad ,$$
$$]$$

CountPaths : *Example*



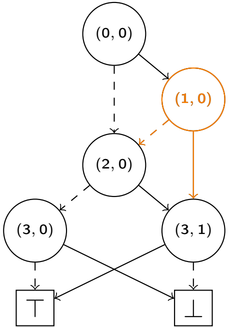
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	0	0

Priority Queue: Q_{count}:

$$[\quad ((0,0) \xrightarrow{\top} (1,0),\ 1),$$
$$((0,0) \xrightarrow{\bot} (2,0),\ 1), \quad]$$

CountPaths : *Example*



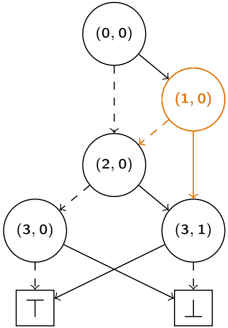
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	0	0

Priority Queue: Q_{count}:

$$[\ ((0,0) \xrightarrow{\top} (1,0),\ 1),\quad ((0,0) \xrightarrow{\perp} (2,0),\ 1),\ ]$$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	1	0

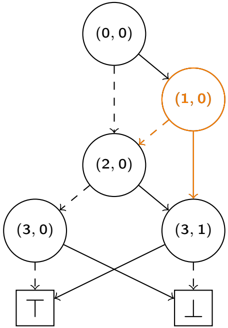
Priority Queue: Q_{count}:

[

$((0,0) \xrightarrow{\perp} (2,0),\quad 1)$,

]

CountPaths : *Example*



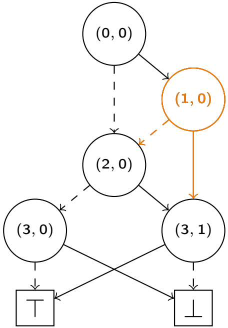
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(1, 0)$	1	0

Priority Queue: Q_{count}:

$$[\; ((0,0) \xrightarrow{\perp} (2,0),\ 1),\ ((1,0) \xrightarrow{\perp} (2,0),\ 1),\ ((1,0) \xrightarrow{\top} (3,1),\ 1),\ ]$$

CountPaths : *Example*



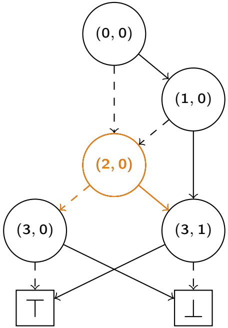
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(2, 0)$	0	0

Priority Queue: Q_{count}:

$$\begin{array}{l} [\\ \quad ((0,0) \xrightarrow{\perp} (2,0), \quad 1) \quad , \\ \quad ((1,0) \xrightarrow{\perp} (2,0), \quad 1) \quad , \\ \\ \quad ((1,0) \xrightarrow{\top} (3,1), \quad 1) \quad , \\ ] \end{array}$$

CountPaths : *Example*



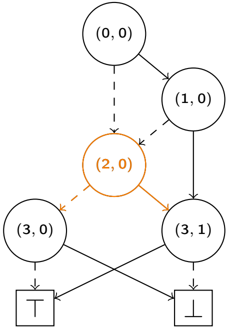
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	0	0

Priority Queue: Q_{count}:

$$
\begin{array}{l}
[\\
\quad ((0,0) \xrightarrow{\perp} (2,0), \quad 1) \quad , \\
\quad ((1,0) \xrightarrow{\perp} (2,0), \quad 1) \quad , \\
\\
\quad ((1,0) \xrightarrow{\top} (3,1), \quad 1) \quad , \\
]
\end{array}
$$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	1	0

Priority Queue: Q_{count}:

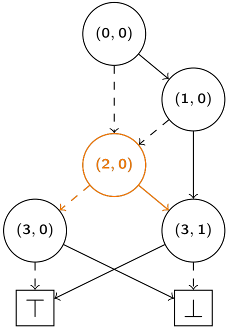
$[$

$((1,0) \xrightarrow{\perp} (2,0), \quad 1) \quad ,$

$((1,0) \xrightarrow{\top} (3,1), \quad 1) \quad ,$

$]$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	2	0

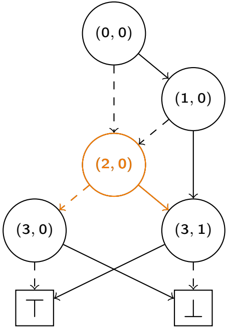
Priority Queue: Q_{count}:

[

$((1,0) \xrightarrow{\top} (3,1), \quad 1)$,

]

CountPaths : *Example*



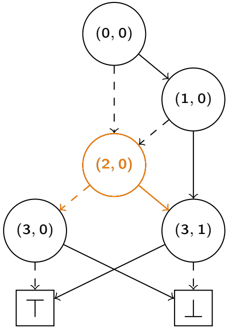
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	2	0

Priority Queue: Q_{count}:

$$[\; ((2,0) \xrightarrow{\perp} (3,0),\ 2),\; ((1,0) \xrightarrow{\top} (3,1),\ 1),\; ((2,0) \xrightarrow{\top} (3,1),\ 2) \;]$$

CountPaths : *Example*



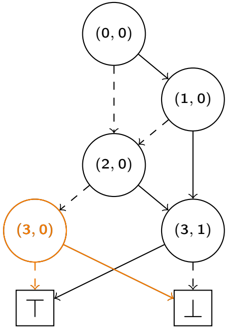
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3,0)$	0	0

Priority Queue: Q_{count}:

$$[\ ((2,0) \xrightarrow{\perp} (3,0),\ 2),\ ((1,0) \xrightarrow{\top} (3,1),\ 1),\ ((2,0) \xrightarrow{\top} (3,1),\ 2)\ ]$$

CountPaths : *Example*



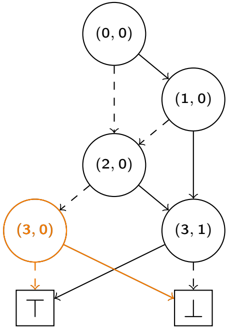
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3,0)$	0	0

Priority Queue: Q_{count}:

$$[\ ((2,0) \xrightarrow{\perp} (3,0),\ 2),\ ((1,0) \xrightarrow{\top} (3,1),\ 1),\ ((2,0) \xrightarrow{\top} (3,1),\ 2)\ ]$$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
$(3, 0)$	2	0

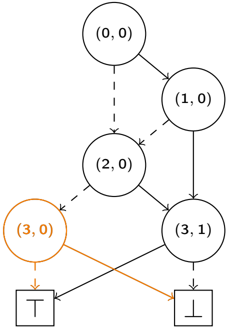
Priority Queue: Q_{count}:

[

$((1, 0) \xrightarrow{\top} (3, 1), \quad 1)$,

$((2, 0) \xrightarrow{\top} (3, 1), \quad 2)$]

CountPaths : *Example*



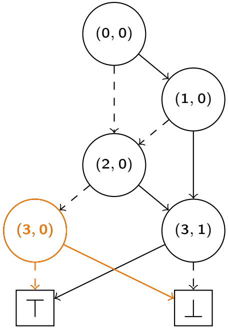
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 0)	2	2

Priority Queue: Q_{count}:

$$[\ ((1,0) \xrightarrow{\top} (3,1),\quad 1)\ ,\quad ((2,0) \xrightarrow{\top} (3,1),\quad 2)\ ]$$

CountPaths : *Example*



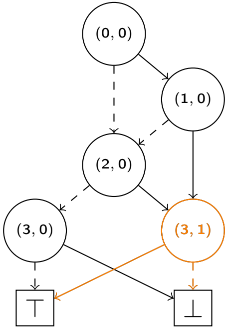
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 1)	0	2

Priority Queue: Q_{count}:

$$[\ ((1,0) \xrightarrow{\top} (3,1),\quad 1),\quad ((2,0) \xrightarrow{\top} (3,1),\quad 2)\ ]$$

CountPaths : *Example*



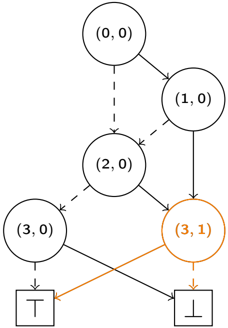
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(3, 1)	0	2

Priority Queue: Q_{count}:

$$[\ ((1,0) \xrightarrow{\top} (3,1),\ 1),\ ((2,0) \xrightarrow{\top} (3,1),\ 2)\ ]$$

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

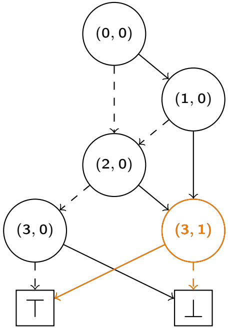
Seek	Sum	Result
(3, 1)	1	2

Priority Queue: Q_{count}:

[

$((2, 0) \xrightarrow{\top} (3, 1), \quad 2)$]

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

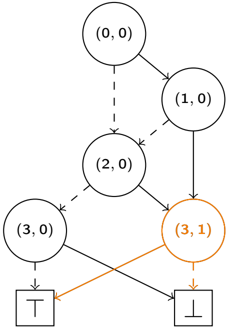
Seek	Sum	Result
(3, 1)	3	2

Priority Queue: Q_{count}:

[

]

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

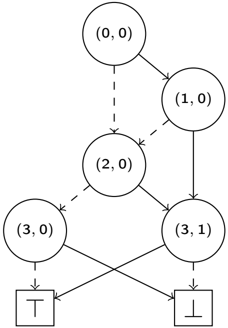
Seek	Sum	Result
$(3, 1)$	3	5

Priority Queue: Q_{count}:

[

]

CountPaths : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Result

5

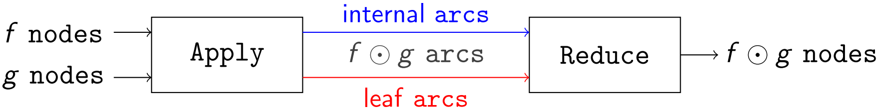
Priority Queue: Q_{count}:

[

]

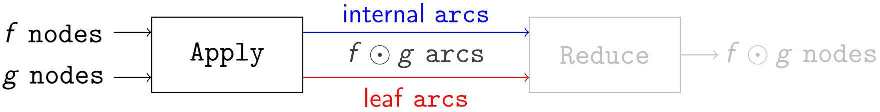
Contents

Apply



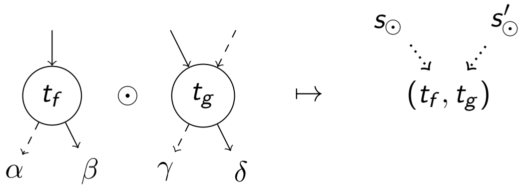
f nodes
g nodes
Apply
internal arcs
$f \odot g$ arcs
leaf arcs
Reduce
$f \odot g$ nodes

Apply



f nodes
g nodes
Apply
internal arcs
$f \odot g$ arcs
leaf arcs
Reduce
$f \odot g$ nodes

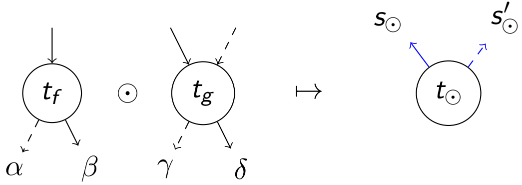
Apply



Time-Forward Processing

Defer resolving products with $Q_{app:1}$, $Q_{app:2}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g), ...)\rangle$.

Apply



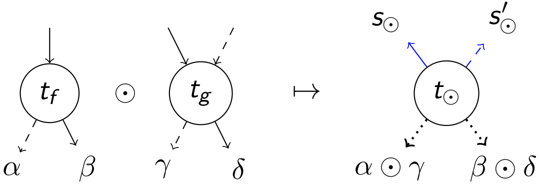
Observation (semi-tranposition)

$\leftarrow$: $s \to t$ (Internal Arcs) are output at time t and hence sorted by t.

Time-Forward Processing

Defer resolving products with $Q_{app:1}$, $Q_{app:2}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g), \ldots)\rangle$.

Apply



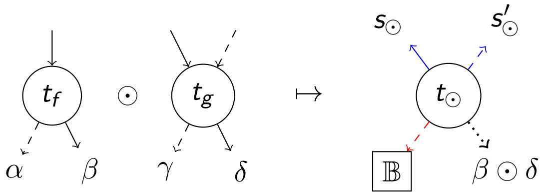
Observation (semi-tranposition)

$\leftarrow$: $s \rightarrow t$ (Internal Arcs) are output at time t and hence sorted by t.

Time-Forward Processing

Defer resolving products with $Q_{app:1}$, $Q_{app:2}$: $\texttt{PriorityQueue}\langle(s \rightarrow (t_f, t_g), \ldots)\rangle$.

Apply



Observation (semi-tranposition)

$\leftarrow$: $s \rightarrow t$ (Internal Arcs) are output at time t and hence sorted by t.

$\rightarrow$: $s \rightarrow \mathbb{B}$ (Terminal Arcs) are output at time s.

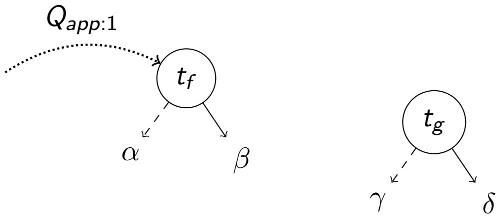
Time-Forward Processing

Defer resolving products with $Q_{app:1}$, $Q_{app:2}$: $\texttt{PriorityQueue}\langle(s \rightarrow (t_f, t_g), \ldots)\rangle$.

Apply

$Q_{app:1}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g))\rangle$
sorted on $\min(t_f, t_g)$ in ascending order.

Case 1 :
$t_f.var() \neq t_g.var()$



Apply

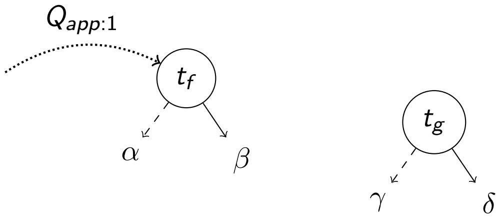
$Q_{app:1}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g))\rangle$
sorted on $\min(t_f, t_g)$ in ascending order.

Case 1 :

$t_f.var() \neq t_g.var()$

Case 2(a):

$t_f.var() = t_g.var() \wedge t_f.id() = t_g.id()$

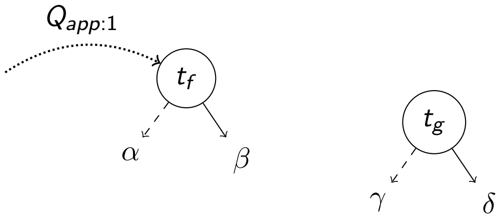


Apply

$Q_{app:1}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g))\rangle$ sorted on $\min(t_f, t_g)$ in ascending order.

Case 1 :

$t_f.var() \neq t_g.var()$



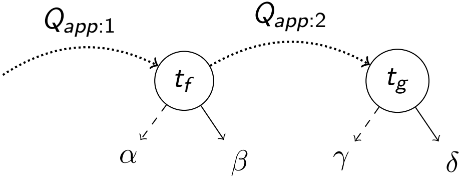
$Q_{app:2}$: $\texttt{PriorityQueue}\langle(s \to (t_f, t_g), (\alpha, \beta))\rangle$ sorted on $\max(t_f, t_g)$ in ascending order.

Case 2(a):

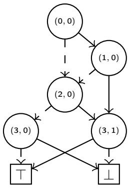
$t_f.var() = t_g.var() \wedge t_f.id() = t_g.id()$

Case 2(b):

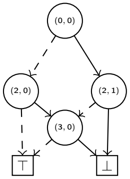
$t_f.var() = t_g.var() \wedge t_f.id() \neq t_g.id()$



Apply : *Example*



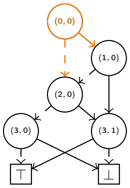
(a) $(x_{\mathbf{0}} \wedge x_{\mathbf{1}} \wedge x_{\mathbf{3}}) \vee (x_{\mathbf{2}} \oplus x_{\mathbf{3}})$



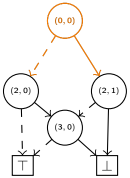
(b) $\neg(x_{\mathbf{0}} \ ? \ x_{\mathbf{2}} \vee x_{\mathbf{3}} \ : \ x_{\mathbf{2}} \wedge x_{\mathbf{3}})$

(c) $(a) \wedge (b)$

Apply : *Example*



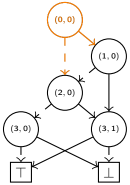
(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



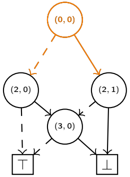
(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



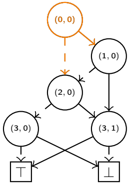
(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

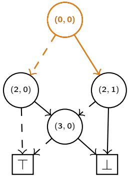
$$[\quad (0,0) \xrightarrow{\top} ((1,0),(2,1)) \quad ,$$
$$(0,0) \xrightarrow{\bot} ((2,0),(2,0)) \quad ,$$
$$]$$

(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

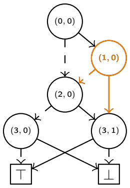
Seek:
$\min((1,0),(2,1))$

Priority Queue: $Q_{app:1}$:

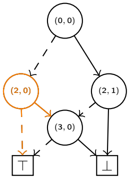
$$[\quad (0,0) \xrightarrow{\top} ((1,0),(2,1)) \ , \quad (0,0) \xrightarrow{\perp} ((2,0),(2,0)) \ , \quad]$$

(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

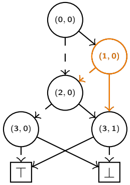
Seek:
$\min((1, 0), (2, 1))$

Priority Queue: $Q_{app:1}$:

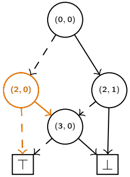
$$[\quad (0,0) \xrightarrow{\top} ((1,0),(2,1)) \quad , \quad (0,0) \xrightarrow{\bot} ((2,0),(2,0)) \quad , \quad]$$

(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_{\mathbf{0}} \wedge x_{\mathbf{1}} \wedge x_{\mathbf{3}}) \vee (x_{\mathbf{2}} \oplus x_{\mathbf{3}})$

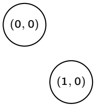


(b) $\neg(x_{\mathbf{0}} \ ? \ x_{\mathbf{2}} \vee x_{\mathbf{3}} \ : \ x_{\mathbf{2}} \wedge x_{\mathbf{3}})$

Seek:
$\min((1,0),(2,1))$

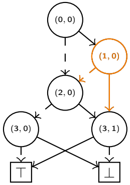
Priority Queue: $Q_{\mathbf{app}:\mathbf{1}}$:

$$
\begin{array}{llll}
[& (0,0) \xrightarrow{\top} ((1,0),(2,1)) & , \\
 & (0,0) \xrightarrow{\perp} ((2,0),(2,0)) & , \\
 & (1,0) \xrightarrow{\perp} ((2,0),(2,1)) & , \\
 & (1,0) \xrightarrow{\top} ((3,1),(2,1)) & , \\
 & &]
\end{array}
$$

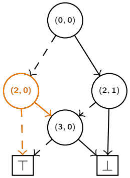


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((1,0),(2,1))$

Priority Queue: $Q_{app:1}$:

[

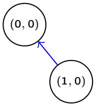
$(0,0) \xrightarrow{\perp} ((2,0),(2,0))$,

$(1,0) \xrightarrow{\perp} ((2,0),(2,1))$,

$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,

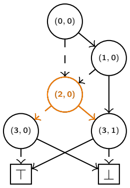
]

Output:
$(0,0) \xrightarrow{\top} (1,0)$

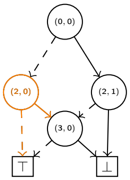


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((2,0),(2,0))$

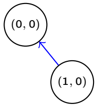
Priority Queue: $Q_{app:1}$:

[

$(0,0) \xrightarrow{\bot} ((2,0),(2,0))$,
$(1,0) \xrightarrow{\bot} ((2,0),(2,1))$,
$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,

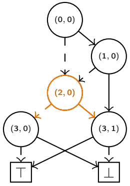
]

Output:

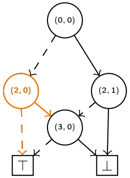


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

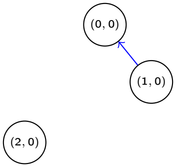
Seek:

$\min((2,0),(2,0))$

Priority Queue: $Q_{app:1}$:

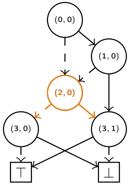
$$
\begin{aligned}
[\ & \\
& (0,0) \xrightarrow{\bot} ((2,0),(2,0)) \quad , \\
& (1,0) \xrightarrow{\bot} ((2,0),(2,1)) \quad , \\
& (1,0) \xrightarrow{\top} ((3,1),(2,1)) \quad , \\
& (2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad , \\
& (2,0) \xrightarrow{\bot} ((3,0),\top) \quad]
\end{aligned}
$$

Output:

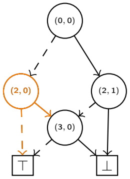


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_{\mathbf{0}} \wedge x_{\mathbf{1}} \wedge x_{\mathbf{3}}) \vee (x_{\mathbf{2}} \oplus x_{\mathbf{3}})$



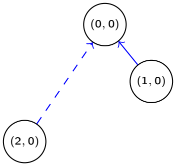
(b) $\neg(x_{\mathbf{0}} \; ? \; x_{\mathbf{2}} \vee x_{\mathbf{3}} \; : \; x_{\mathbf{2}} \wedge x_{\mathbf{3}})$

Seek:
$\min((2,0),(2,0))$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

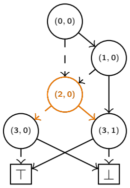
$$
\begin{array}{ll}
[& \\
(1,0) \xrightarrow{\perp} ((2,0),(2,1)) & , \\
(1,0) \xrightarrow{\top} ((3,1),(2,1)) & , \\
(2,0) \xrightarrow{\top} ((3,1),(3,0)) & , \\
(2,0) \xrightarrow{\perp} ((3,0),\top) &]
\end{array}
$$

Output:
$(0,0) \xrightarrow{\perp} (2,0)$

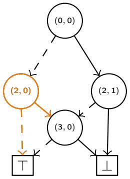


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((2,0),(2,1))$

Priority Queue: $Q_{app:1}$:

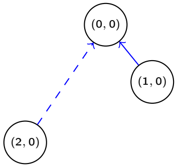
[

$(1,0) \xrightarrow{\perp} ((2,0),(2,1))$,
$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,

$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

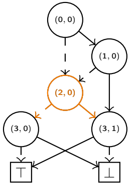
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Output:

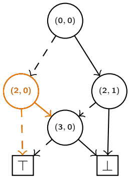


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \land x_1 \land x_3) \lor (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \lor x_3 \ : \ x_2 \land x_3)$

Seek:
min((2, 0), (2, 1))

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(1, 0) \xrightarrow{\top} ((3, 1), (2, 1))$,

$(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$,

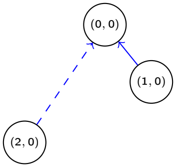
$(2, 0) \xrightarrow{\bot} ((3, 0), \top)$]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[$(1, 0) \xrightarrow{\bot} ((2, 0), (2, 1)) \quad ((3, 0), (3, 1))$,

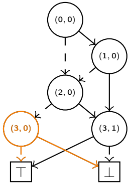
]

Output:

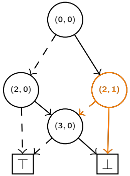


(c) $(a) \land (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:

$\max((2,0),(2,1))$

Priority Queue: $Q_{app:1}$:

[

$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,

$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

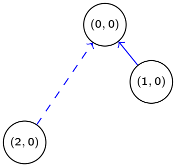
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[$(1,0) \xrightarrow{\bot} ((2,0),(2,1))$ $((3,0),(3,1))$,

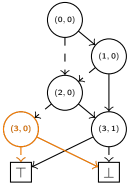
]

Output:

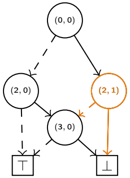


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_{\mathbf{0}} \wedge x_{\mathbf{1}} \wedge x_{\mathbf{3}}) \vee (x_{\mathbf{2}} \oplus x_{\mathbf{3}})$



(b) $\neg(x_{\mathbf{0}} \; ? \; x_{\mathbf{2}} \vee x_{\mathbf{3}} \; : \; x_{\mathbf{2}} \wedge x_{\mathbf{3}})$

Seek:
$\max((2,0),(2,1))$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,
$(2,1) \xrightarrow{\bot} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

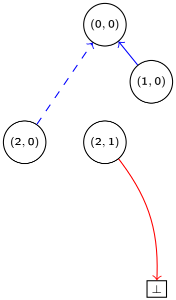
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[$(1,0) \xrightarrow{\bot} ((2,0),(2,1))$ $((3,0),(3,1))$,

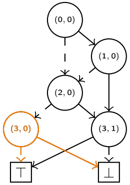
]

Output:
$(2,1) \xrightarrow{\top} \bot$

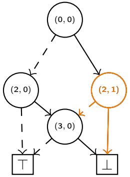


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

Seek:
$\max((2,0),(2,1))$

Priority Queue: $Q_{app:1}$:

[

$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,
$(2,1) \xrightarrow{\perp} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

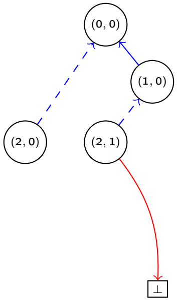
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[

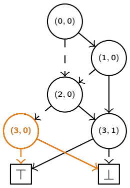
]

Output:
$(1,0) \xrightarrow{\perp} (2,1)$

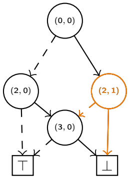


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((3,1),(2,1))$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,
$(2,1) \xrightarrow{\perp} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

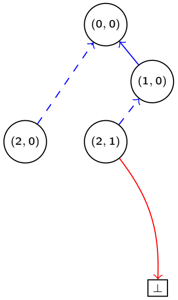
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

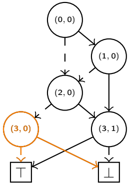
]

Output:

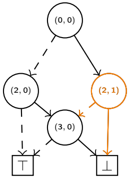


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \;?\; x_2 \vee x_3 \;:\; x_2 \wedge x_3)$

Seek:
$\min((3,1),(2,1))$

Priority Queue: $Q_{app:1}$:

[

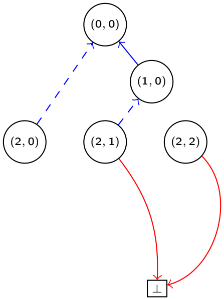
$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,
$(2,1) \xrightarrow{\bot} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\bot} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[

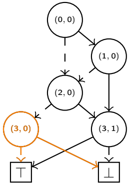
]

Output:
$(2,2) \xrightarrow{\top} \bot$

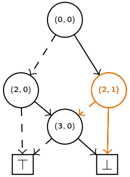


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((3,1),(2,1))$

Priority Queue: $Q_{app:1}$:

[

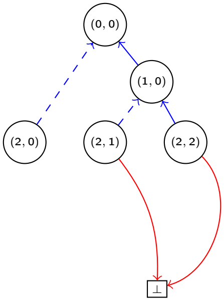
$(2,1) \xrightarrow{\perp} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\perp} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[

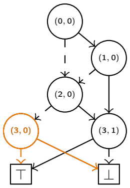
]

Output:
$(1,0) \xrightarrow{\top} (2,2)$

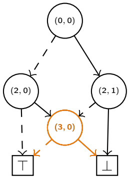


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
min((3, 0), (3, 0))

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

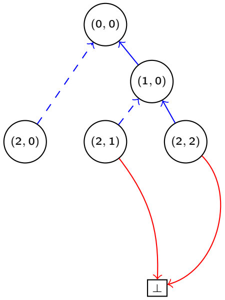
$(2,1) \xrightarrow{\perp} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\perp} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

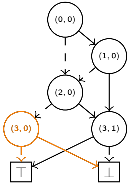
]

Output:

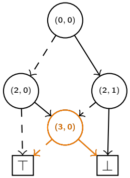


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((3,0),(3,0))$

Priority Queue: $Q_{app:1}$:

[

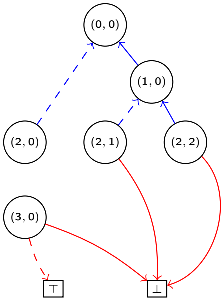
$(2,1) \xrightarrow{\perp} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\perp} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[

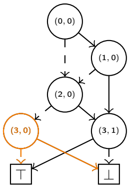
]

Output:
$(3,0) \xrightarrow{\perp} \top, (3,0) \xrightarrow{\top} \perp$

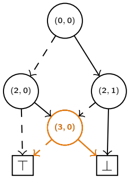


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \;?\; x_2 \vee x_3 \;:\; x_2 \wedge x_3)$

Seek:
$\min((3,0),(3,0))$

Priority Queue: $Q_{app:1}$:

[

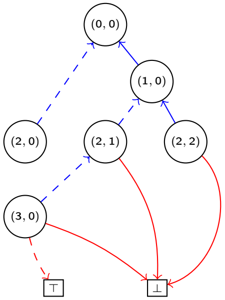
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\bot} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

Priority Queue: $Q_{app:2}$:

[

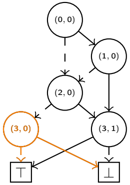
]

Output:
$(2,1) \xrightarrow{\bot} (3,0)$

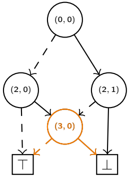


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:

min((3, 1), (3, 0))

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,

$(2,2) \xrightarrow{\perp} ((3,1),(3,0))$,

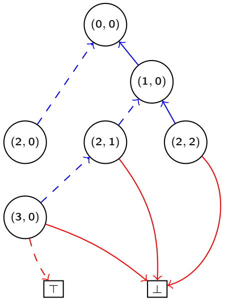
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

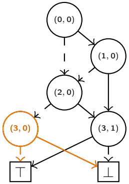
]

Output:

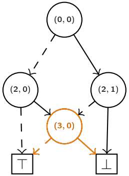


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
min((3, 1), (3, 0))

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(2,0) \xrightarrow{\bot} ((3,0), \top)$]

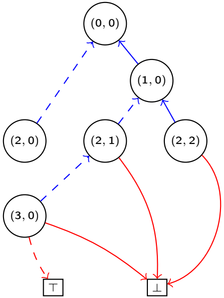
Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \bot)$,

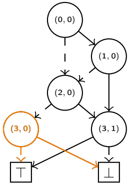
$(2,2) \xrightarrow{\bot} ((3,1),(3,0)) \quad (\top, \bot)$]

Output:

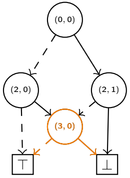


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

Seek:
$\min((3,0), \top)$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

$(2,0) \xrightarrow{\perp} ((3,0), \top)$]

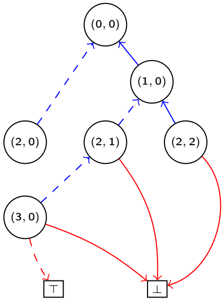
Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \perp)$,

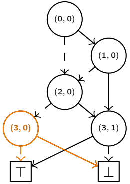
$(2,2) \xrightarrow{\perp} ((3,1),(3,0)) \quad (\top, \perp)$]

Output:

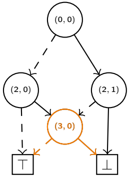


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0\ ?\ x_2 \vee x_3\ :\ x_2 \wedge x_3)$

Seek:

$\min((3,0), \top)$

Priority Queue: $Q_{app:1}$:

[

$(2,0) \xrightarrow{\perp} ((3,0), \top)$]

Priority Queue: $Q_{app:2}$:

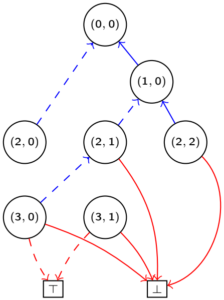
[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \perp)$,

$(2,2) \xrightarrow{\perp} ((3,1),(3,0)) \quad (\top, \perp)$]

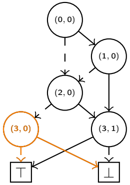
Output:

$(3,1) \xrightarrow{\perp} \top, (3,1) \xrightarrow{\top} \perp$

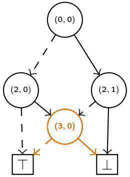


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:
$\min((3,0), \top)$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

]

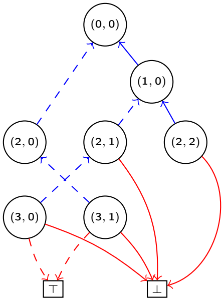
Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \bot)$,

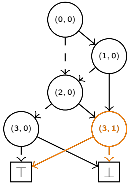
$(2,2) \xrightarrow{\bot} ((3,1),(3,0)) \quad (\top, \bot)$]

Output:
$(2,0) \xrightarrow{\bot} (3,1)$

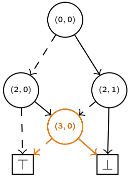


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:

$\max((3,1),(3,0))$

Priority Queue: $Q_{app:1}$:

[

]

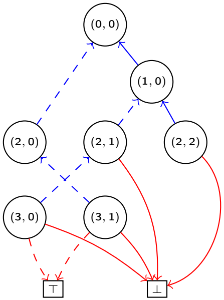
Priority Queue: $Q_{app:2}$:

[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \bot)$,

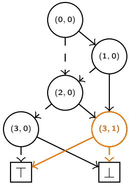
$(2,2) \xrightarrow{\bot} ((3,1),(3,0)) \quad (\top, \bot)$]

Output:

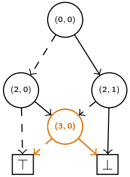


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:

$\max((3,1),(3,0))$

Priority Queue: $Q_{app:1}$:

[

]

Priority Queue: $Q_{app:2}$:

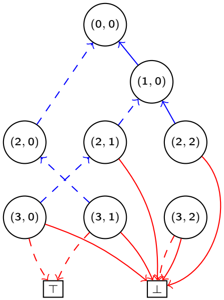
[

$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top,\bot)$,

$(2,2) \xrightarrow{\bot} ((3,1),(3,0)) \quad (\top,\bot)$]

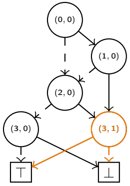
Output:

$(3,2) \xrightarrow{\bot} \bot, (3,2) \xrightarrow{\top} \bot$

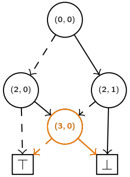


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Seek:

max((3, 1), (3, 0))

Priority Queue: $Q_{app:1}$:

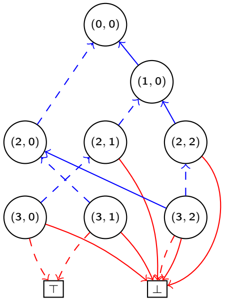
[ ]

Priority Queue: $Q_{app:2}$:

[ ]

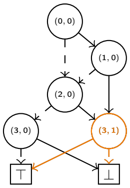
Output:

$(2,0) \xrightarrow{\top} (3,2), (2,2) \xrightarrow{\perp} (3,2)$

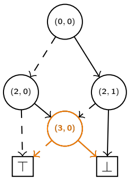


(c) $(a) \wedge (b)$

Apply : *Example*



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{1}}$:

[

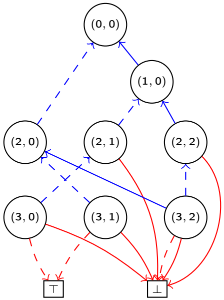
]

Priority Queue: $Q_{\boldsymbol{app}:\mathbf{2}}$:

[

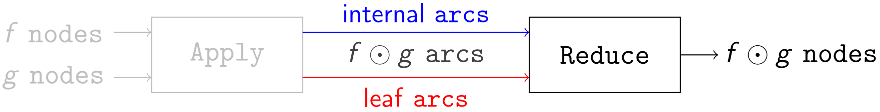
]

Output:



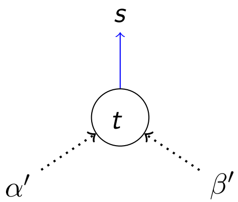
(c) $(a) \wedge (b)$

Apply



f nodes
g nodes
Apply
internal arcs
$f \odot g$ arcs
leaf arcs
Reduce
$f \odot g$ nodes

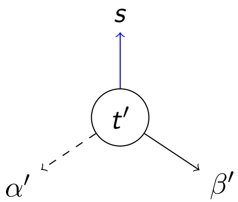
Apply (Reduce)



Time-Forward Processing

Send reduction t' with Q_{red} : $\texttt{PriorityQueue}\langle(s \to t')\rangle$ descending on parent s.

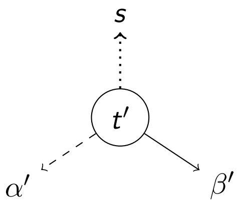
Apply (Reduce)



Time-Forward Processing

Send reduction t' with Q_{red} : $\texttt{PriorityQueue}\langle(s \to t')\rangle$ descending on parent s.

Apply (Reduce)



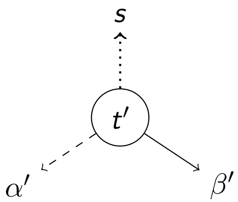
Time-Forward Processing

Send reduction t' with Q_{red} : `PriorityQueue`$\langle(s \rightarrow t')\rangle$ descending on parent s.

Observation (semi-tranposition)

$\leftarrow$: $s \rightarrow t$ (Internal Arcs) provide parents of unreduced node t.

Apply (Reduce)



Time-Forward Processing

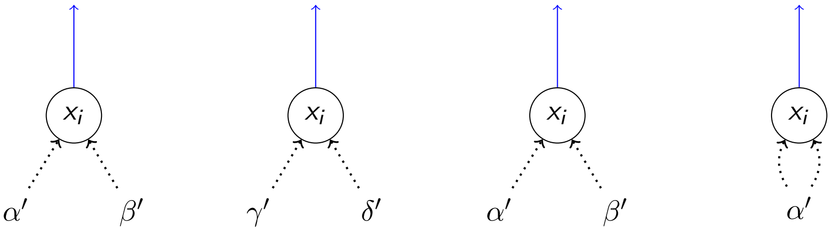
Send reduction t' with Q_{red} : $\texttt{PriorityQueue}\langle(s \rightarrow t')\rangle$ descending on parent s.

Observation (semi-tranposition)

$\leftarrow$: $s \rightarrow t$ (Internal Arcs) provide parents of unreduced node t.

$\rightarrow$: $s \rightarrow \mathbb{B}$ (Terminal Arcs) are reduced and already sorted as per Q_{red}.

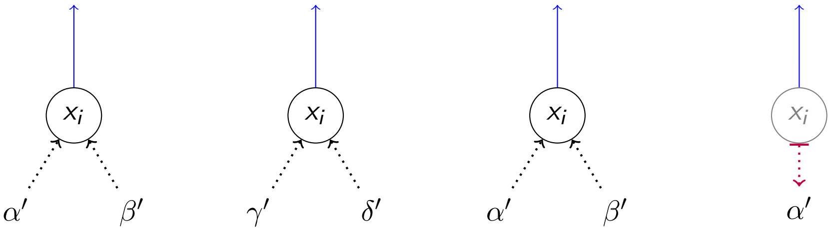
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.

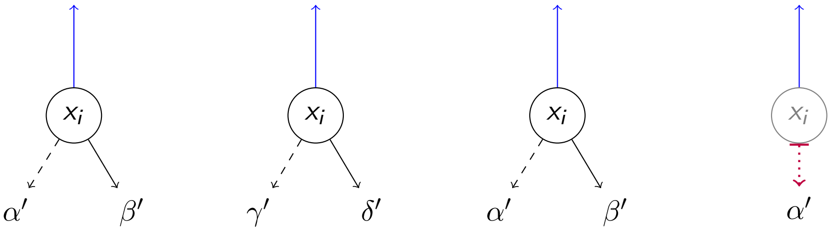
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.

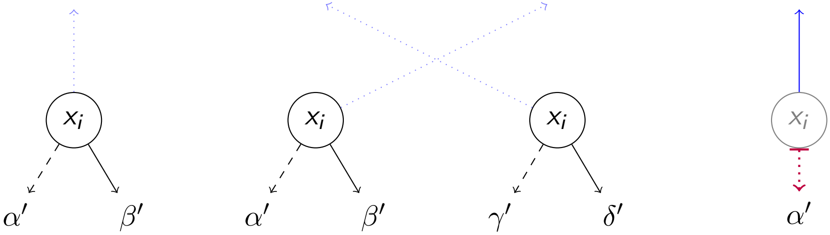
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.

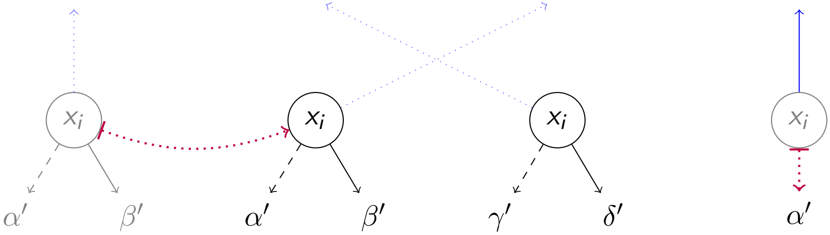
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.
2. Sort remaining nodes by children, output unique nodes, and remember duplications.

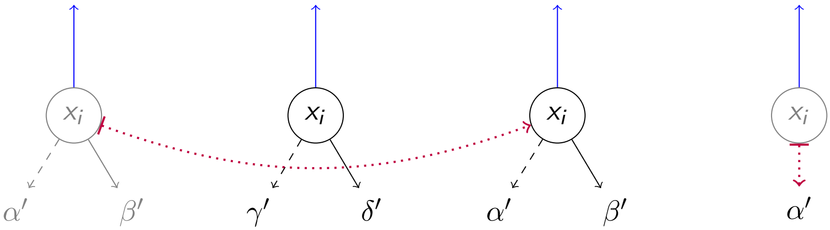
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.
2. Sort remaining nodes by children, output unique nodes, and remember duplications.

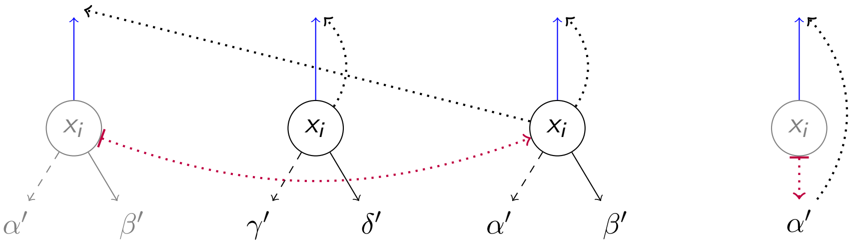
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.
2. Sort remaining nodes by children, output unique nodes, and remember duplications.
3. Sort back to match internal arcs and forward to parents with Q_{red}.

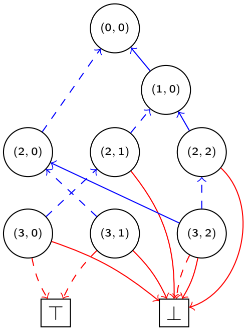
Apply (Reduce)



Reduce Level i:

1. Obtain nodes from Q_{red} and terminal arcs. Filter and remember redundant nodes.
2. Sort remaining nodes by children, output unique nodes, and remember duplications.
3. Sort back to match internal arcs and forward to parents with Q_{red}.

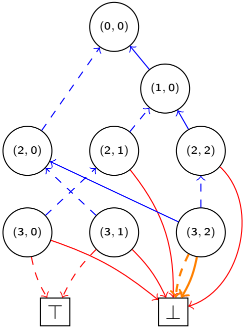
Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

(d) $(a) \wedge (b)$ **reduced**

Apply (Reduce) : *Example*



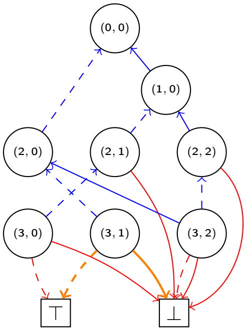
(c) $(a) \wedge (b)$

Level: 3

[$[(3,2) \mapsto \perp]$]

(d) $(a) \wedge (b)$ **reduced**

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

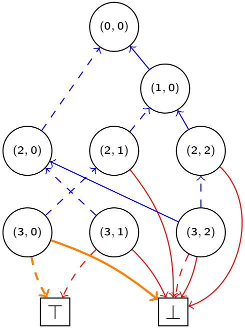
Level: 3

[$[(3,2) \mapsto \perp]$]

[$((3,1), \top, \perp)$,
]

(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

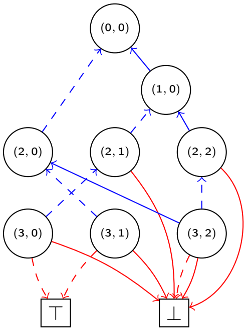
Level: 3

[$[(3,2) \mapsto \bot]$]

[$((3,1), \top, \bot)$,
$((3,0), \top, \bot)$]

(d) $(a) \wedge (b)$ **reduced**

Apply (Reduce) : *Example*



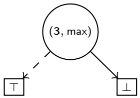
(c) $(a) \wedge (b)$

Level: 3

[$[(3,2) \mapsto \perp]$]

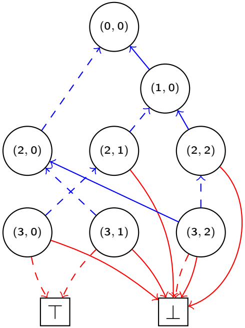
[$[(3,1) \mapsto (3, \text{max})]$,
$((3,0), \top, \perp)$]

Output:
$((3, \text{max}), \top, \perp)$



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



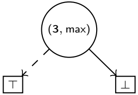
(c) $(a) \wedge (b)$

Level: 3

[$[(3, 2) \mapsto \bot]$]

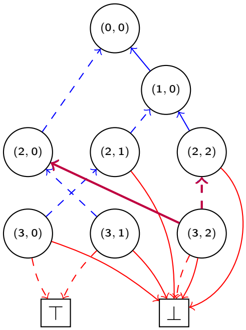
[$[(3, 1) \mapsto (3, \text{max})]$,
$[(3, 0) \mapsto (3, \text{max})]$]

Output:



(d) $(a) \wedge (b)$ **reduced**

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: Q_{red}:

$[\quad (2,2) \xrightarrow{\bot} \bot \quad ,$

$(2,0) \xrightarrow{\top} \bot \quad ,$

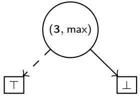
$]$

Level: 3

$[\quad]$

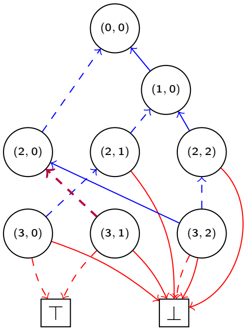
$[\quad [(3,1) \mapsto (3, \max)] \quad ,$
$[(3,0) \mapsto (3, \max)] \quad]$

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: Q_{red}:

$[\quad (2,2) \xrightarrow{\perp} \perp \quad ,$

$(2,0) \xrightarrow{\top} \perp \quad ,$

$(2,0) \xrightarrow{\perp} (3, \text{max}) \quad ,$

$]$

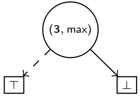
Level: 3

$[\quad]$

$[\quad ,$

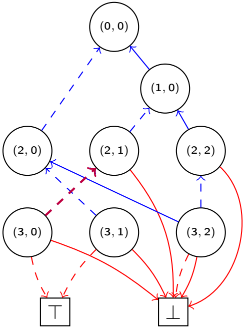
$[(3,0) \mapsto (3, \text{max})] \quad]$

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

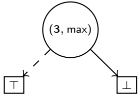
[$(2,2) \xrightarrow{\perp} \perp$,
$(2,1) \xrightarrow{\perp} (3, \text{max})$,
$(2,0) \xrightarrow{\top} \perp$,
$(2,0) \xrightarrow{\perp} (3, \text{max})$,
]

Level: 3

[ ]

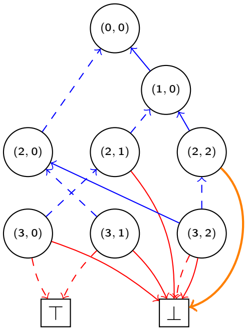
[,
]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

$(2,1) \xrightarrow{\perp} (3, \max)$,

$(2,0) \xrightarrow{\top} \perp$,

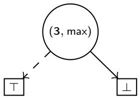
$(2,0) \xrightarrow{\perp} (3, \max)$,

]

Level: 2

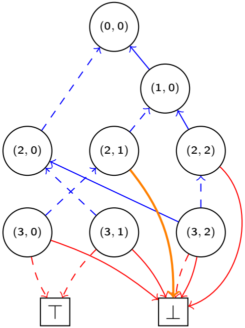
[$[(2,2) \mapsto \perp]$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

$(2, 0) \xrightarrow{\top} \bot$,

$(2, 0) \xrightarrow{\bot} (3, \text{max})$,

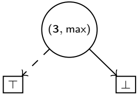
]

Level: 2

[$[(2, 2) \mapsto \bot]$]

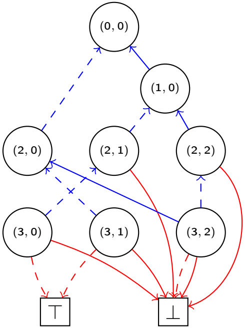
[$((2, 1), (3, \text{max}), \bot)$,
]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

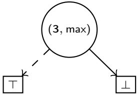
]

Level: 2

[$[(2, 2) \mapsto \perp]$]

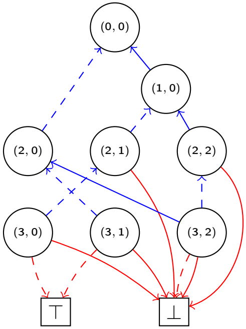
[$((2, 1), (3, \text{max}), \perp)$,
$((2, 0), (3, \text{max}), \perp)$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

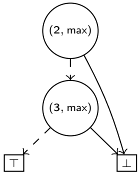
]

Level: 2

[$[(2, 2) \mapsto \bot]$]

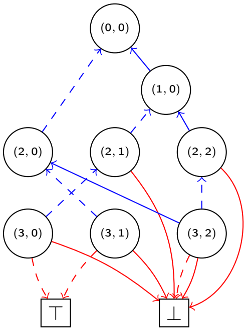
[$[(2, 1) \mapsto (2, \max)]$,
$((2, 0), (3, \max), \bot)$]

Output:
$((2, \max), (3, \max), \bot)$



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

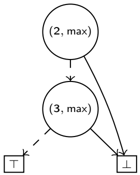
]

Level: 2

[$[(2,2) \mapsto \bot]$]

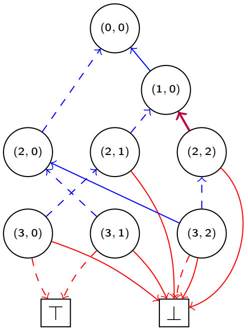
[$[(2,1) \mapsto (2, \mathsf{max})]$,
$[(2,0) \mapsto (2, \mathsf{max})]$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

$(1, 0) \xrightarrow{\top} \bot$,

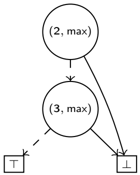
]

Level: 2

[ ]

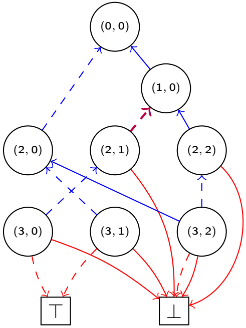
[$[(2, 1) \mapsto (2, \mathrm{max})]$,
$[(2, 0) \mapsto (2, \mathrm{max})]$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

$(1,0) \xrightarrow{\top} \bot$,

$(1,0) \xrightarrow{\bot} (2, \mathrm{max})$,

]

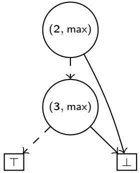
Level: 2

[ ]

[,

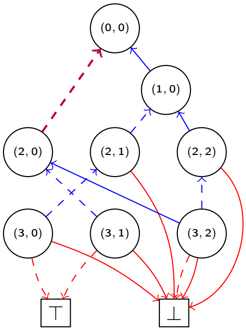
$[(2,0) \mapsto (2, \mathrm{max})]$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

$(1,0) \xrightarrow{\top} \bot$,

$(1,0) \xrightarrow{\bot} (2, \mathrm{max})$,

$(0,0) \xrightarrow{\bot} (2, \mathrm{max})$]

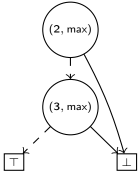
Level: 2

[ ]

[,

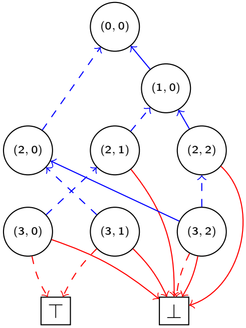
]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

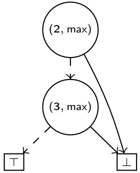
$(0, 0) \xrightarrow{\perp} (2, \text{max})$]

Level: 1

[ ]

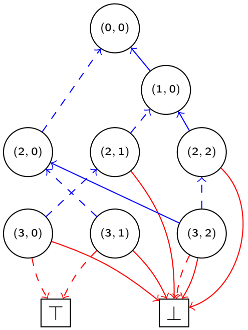
[$((1, 0), (2, \text{max}), \perp)$]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

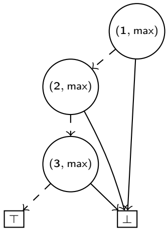
$(0, 0) \xrightarrow{\perp} (2, \max)$]

Level: 1

[ ]

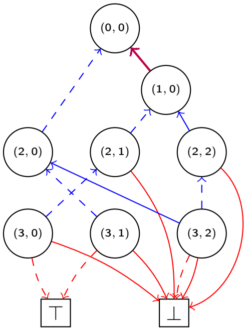
[$[(1, 0) \mapsto (1, \max)]$]

Output:
$((1, \max), (2, \max), \perp)$



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: Q_{red}:

[

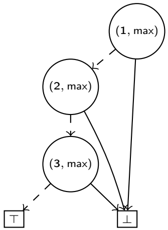
$(0,0) \xrightarrow{\top} (1, \max)$,
$(0,0) \xrightarrow{\perp} (2, \max)$]

Level: 1

[ ]

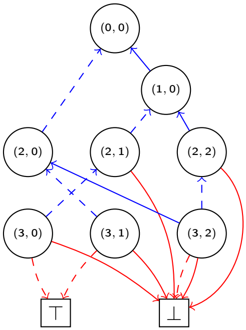
[ ]

Output:



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \land (b)$

Priority Queue: Q_{red}:

[

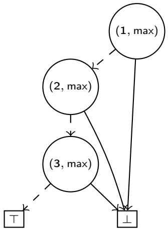
]

Level: 0

[ ]

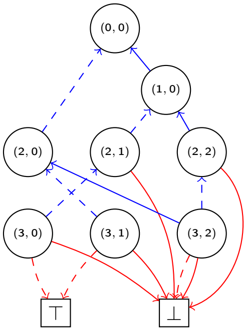
[((0, 0), (2, max), (1, max))]

Output:



(d) $(a) \land (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

]

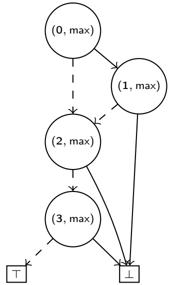
Level: 0

[ ]

[[(0, 0) $\mapsto$ (0, max)]]

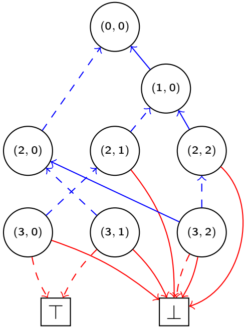
Output:

((0, max), (2, max), (1, max))



(d) $(a) \wedge (b)$ reduced

Apply (Reduce) : *Example*



(c) $(a) \wedge (b)$

Priority Queue: $Q_{\boldsymbol{red}}$:

[

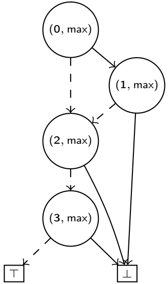
]

Level: 0

[ ]

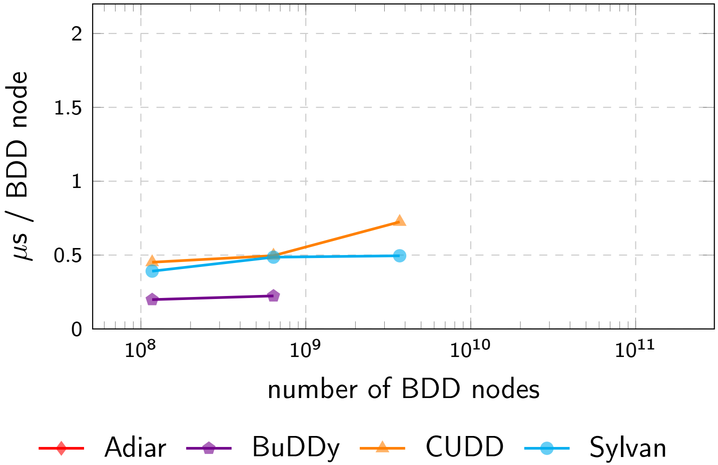
[ ]

Output:

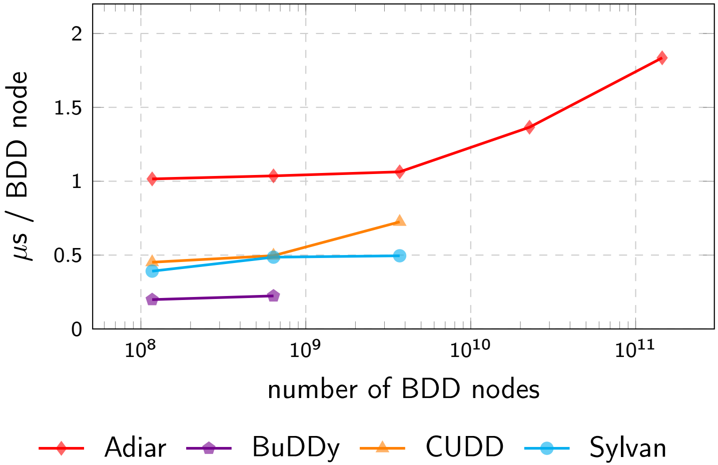


(d) $(a) \wedge (b)$ reduced

Algorithm	I/O-Complexity
`bdd_pathcount`	$O(\text{sort}(N_f))$
`bdd_not`	$2N_f/B$
`bdd_restrict`	$O(\text{sort}(N_f))$
`bdd_apply`	$O(\text{sort}(N_f \cdot N_g))$



Running time for the *N-Queens* problems.



Running time for the *N-Queens* problems.

Contents

Algorithm	I/O-Complexity
`bdd_pathcount`	$O(\text{sort}(N_f))$
`bdd_not`	$2N_f/B$
`bdd_restrict`	$O(\text{sort}(N_f))$
`bdd_apply`	$O(\text{sort}(N_f \cdot N_g))$

Algorithm	I/O-Complexity
`bdd_pathcount`	$O(\mathrm{sort}(N_f))$
`bdd_not`	$2N_f/B$
`bdd_restrict`	$O(\mathrm{sort}(N_f))$
`bdd_apply`	$O(\mathrm{sort}(N_f \cdot N_g))$
`bdd_equal`	?

Equality Checking

$$f \leftrightarrow g \equiv \top$$

Equality Checking

$$f \leftrightarrow g \equiv \top$$

$$\underbrace{O(\mathrm{sort}(N^2))}_{\texttt{Apply}} + \underbrace{O(\mathrm{sort}(N^2))}_{\texttt{Reduce}} + \underbrace{O(1))}_{\text{check is } \top} = O(\mathrm{sort}(N^2))$$

Equality Checking

Theorem (Bryant '86)
Let π *be a variable order and* $f : \mathbb{B}^n \to \mathbb{B}$ *then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing* f *with ordering* π*.*

Equality Checking

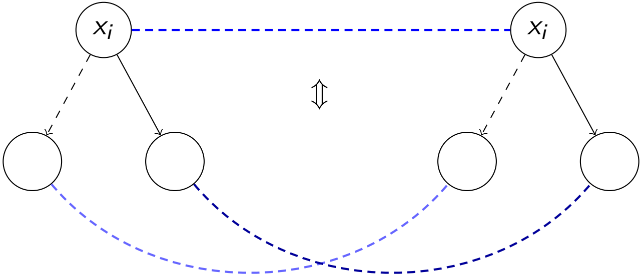
Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \rightarrow \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.

Trivial cases: $f \not\equiv g$ if there is a mismatch in

$N_f \neq N_g$	Number of nodes	$O(1)$ I/Os
$L_f \neq L_g$	Number of levels	$O(1)$ I/Os
$N_{f,i} \neq N_{g,i}$	Number of nodes on a level	$O(L/B)$ I/Os
$L_{f,i} \neq L_{g,i}$	Label of an ith level	$O(L/B)$ I/Os

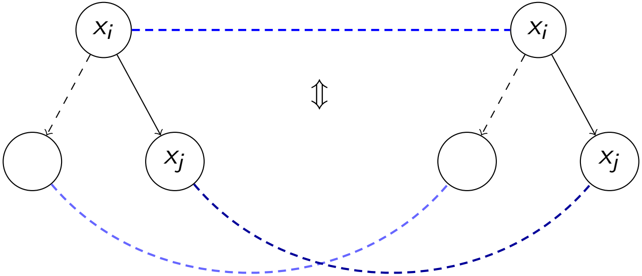
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



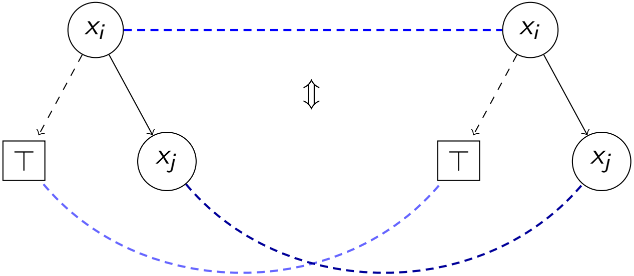
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



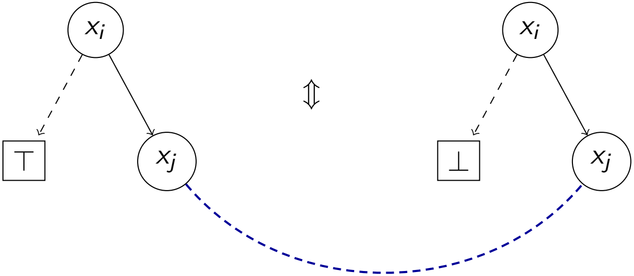
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \rightarrow \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



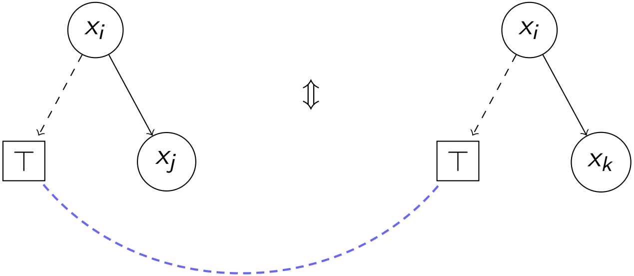
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



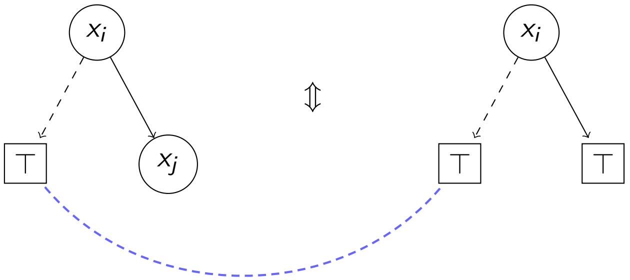
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.

`IsIsomorphic(`f`, `g`)`

- Check whether root v_f of f and root v_g of g have a local violation.
- Check $low(v_f) \sim low(v_g)$ and $high(v_f) \sim high(v_g)$ "recursively".

Return `false` on first violation. If there are no violations then return `true`.

Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.

`IsIsomorphic(`f`, `g`)`

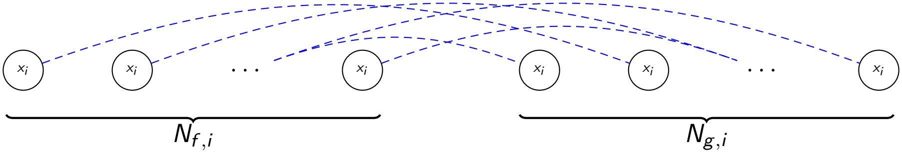
- Check whether root v_f of f and root v_g of g have a local violation.
- Check $low(v_f) \sim low(v_g)$ and $high(v_f) \sim high(v_g)$ "recursively".

Return `false` on first violation. If there are no violations then return `true`.

$$\underbrace{O(\mathrm{sort}(N^2))}_{\texttt{Apply}'} + \underbrace{\cancel{O(\mathrm{sort}(N^2))}}_{\texttt{Reduce}} + \underbrace{\cancel{O(1)}}_{\text{check is } \top} = O(\mathrm{sort}(N^2))$$

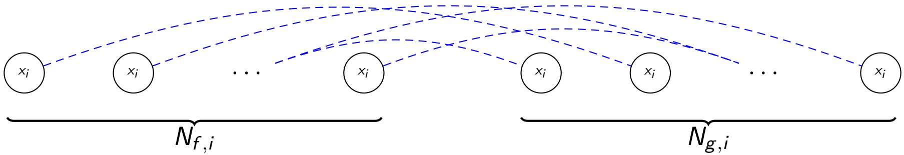
Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



Equality Checking

Theorem (Bryant '86)
Let π be a variable order and $f : \mathbb{B}^n \to \mathbb{B}$ then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing f with ordering π.



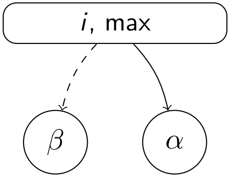
Return `false` if more than $N_{f,i} = N_{g,i}$ pairs of nodes are checked on level i.

$$\underbrace{O(\text{sort}(\Sigma_i\ N_{f,i}))}_{\texttt{Apply}''} = O(\text{sort}(N))$$

Equality Checking

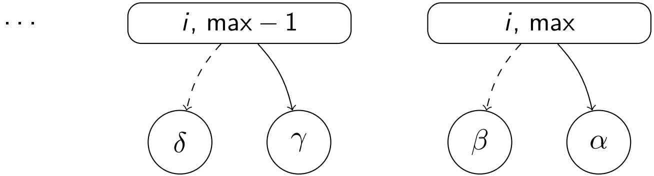
Observation
Each level output by the `Reduce` algorithm has the following properties:

Equality Checking



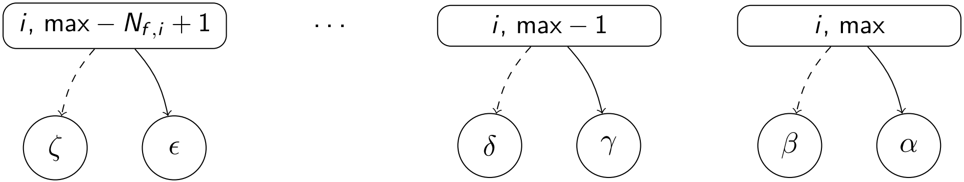
Observation
Each level output by the `Reduce` algorithm has the following properties:

Equality Checking



Observation
Each level output by the `Reduce` algorithm has the following properties:

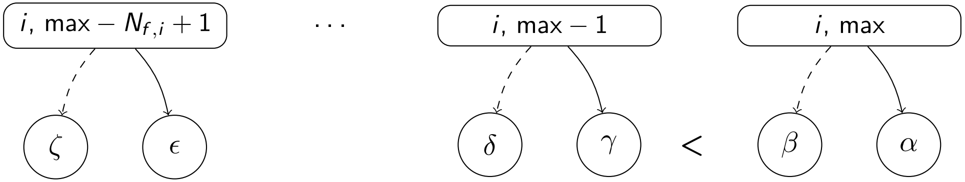
Equality Checking



Observation
Each level output by the `Reduce` algorithm has the following properties:

- Nodes on level i have their identifiers *consecutively* numbered.

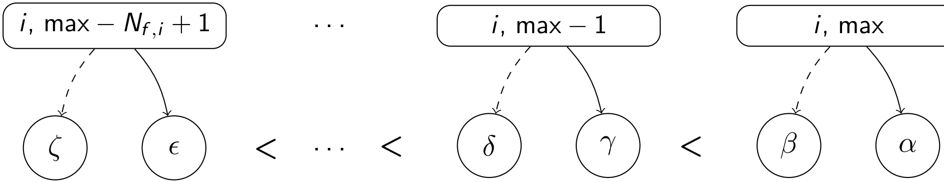
Equality Checking



Observation
Each level output by the `Reduce` algorithm has the following properties:

- Nodes on level i have their identifiers *consecutively* numbered.

Equality Checking



Observation
Each level output by the `Reduce` algorithm has the following properties:

- Nodes on level i have their identifiers *consecutively* numbered.
- Nodes on level i are output sorted by their children.

Equality Checking

Theorem
If G_f and G_g are outputs of `Reduce`*.*

$$G_f \sim G_g \iff \textit{For all } i \in [0; N_f) \textit{ the node } G_f[i] \textit{ matches } G_g[i] \textit{ numerically.}$$

Proof.
$\Leftarrow$: Must describe the exact same graph.

$\Rightarrow$: Strong induction on BDD levels bottom-up. $\square$

Equality Checking

Theorem
If G_f and G_g are outputs of `Reduce`.

$$G_f \sim G_g \iff \text{For all } i \in [0; N_f) \text{ the node } G_f[i] \text{ matches } G_g[i] \text{ numerically.}$$

Proof.
$\Leftarrow$: Must describe the exact same graph.

$\Rightarrow$: Strong induction on BDD levels bottom-up. $\square$

Corollary
If G_f and G_g are outputs of `Reduce` *then $f \equiv g$ is computable using $2 \cdot N/B$ I/Os.*

Equality Checking

Algorithm	Time (s)
$f \leftrightarrow g \equiv \top$	0.38

Checking the (EPFL Benchmark) *voter* circuit's single output gate ($|N_f| = |N_g| = 5.76$ MiB).

Equality Checking

Algorithm	Time (s)
$f \leftrightarrow g \equiv \top$	0.38
$O(\text{sort}(N))$	0.058

Checking the (EPFL Benchmark) *voter* circuit's single output gate ($|N_f| = |N_g| = 5.76$ MiB).

Equality Checking

Algorithm	Time (s)
$f \leftrightarrow g \equiv \top$	0.38
$O(\text{sort}(N))$	0.058
$2N/B$	0.006

Checking the (EPFL Benchmark) *voter* circuit's single output gate ($|N_f| = |N_g| = 5.76$ MiB).

Steffan Christ Sølvsten

`soelvsten@cs.au.dk`

ssoelvsten.github.io

Adiar

github.com/ssoelvsten/adiar

ssoelvsten.github.io/adiar

AARHUS UNIVERSITY

Algorithm	Depth-First	Time-Forward
`bdd_pathcount`	$O(N_f)$	$O(\text{sort}(N_f))$
`bdd_not`	$O(N_f)$	$2N_f/B$
`bdd_restrict`	$O(N_f)$	$O(\text{sort}(N_f))$
`bdd_apply`	$O(N_f N_g)$	$O(\text{sort}(N_f N_g))$
`bdd_equal`	$O(1)$	$2N/B$