

# I/O-efficient Manipulation of Binary Decision Diagrams

Steffan Christ Sølvesten

---

S. C. Sølvesten, J. van de Pol, A. B. Jakobsen, and M. W. B. Thomasen.

*Adiar: Binary Decision Diagrams in External Memory.* 2022





# Contents

What are Binary Decision Diagrams?

Why do they break?

How can we fix it?

- CountPaths

- Apply

- Equality Checking

# Contents

What are Binary Decision Diagrams?

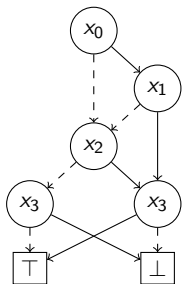
Why do they break?

How can we fix it?

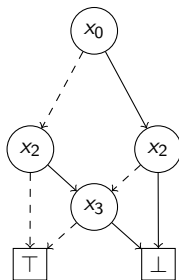
CountPaths

Apply

Equality Checking



**(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:**

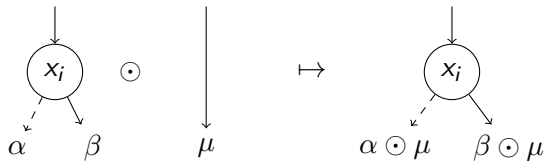


`bdd_apply(f,g,⊙)`

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



**Inductive Case:**





`bdd_apply(f,g,⊙)`

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,⊙)` 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,⊙)` runs in  $O(1)$  time per BDD node.

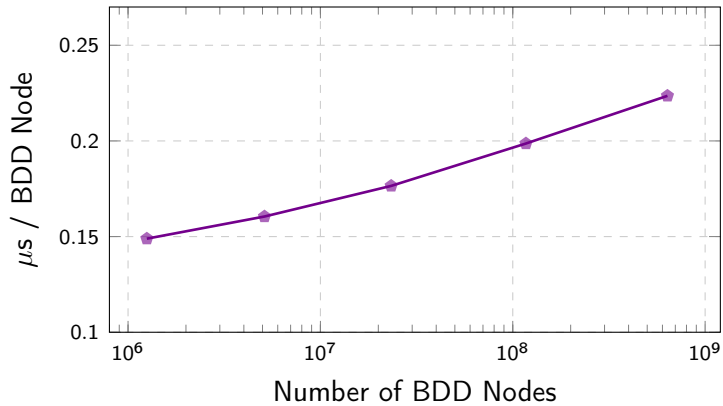


# Adiar

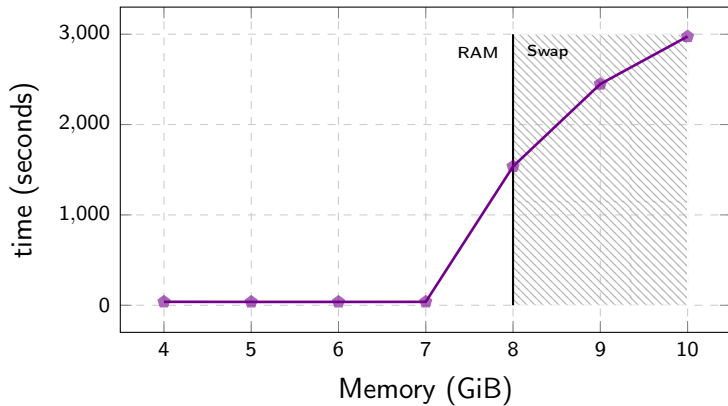
*I/O-efficient Decision Diagrams*

---

[github.com/ssoelvsten/adiar](https://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

What are Binary Decision Diagrams?

Why do they break?

How can we fix it?

CountPaths

Apply

Equality Checking











The I/O model by Aggarwal and Vitter '87

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

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

**Theorem (Aggarwal and Vitter '87)**

*$N$  elements can be sorted in  $\Theta(\text{sort}(N))$  I/Os.*

**Theorem (Arge '95)**

*A Priority Queue can do  $N$  insertions and extractions in  $\Theta(\text{sort}(N))$  I/Os.*

## 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
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*

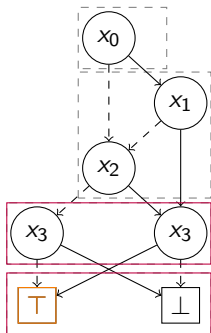


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

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*

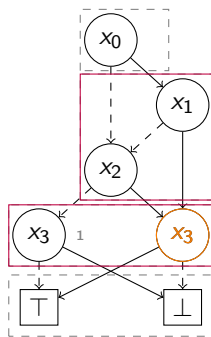


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

node I/Os	cache lookups
5	3

## CountPaths : *Example*

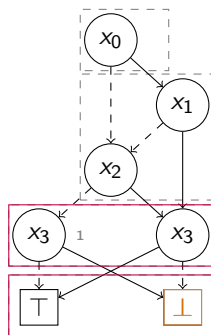


(a)  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

node I/Os	cache lookups
5	4

## CountPaths : *Example*

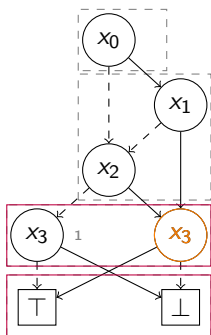


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

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*

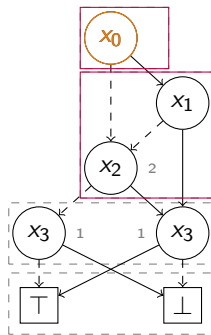


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

node I/Os	cache lookups
7	4

## CountPaths : *Example*

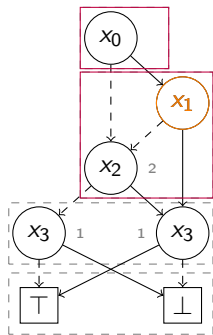


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

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*

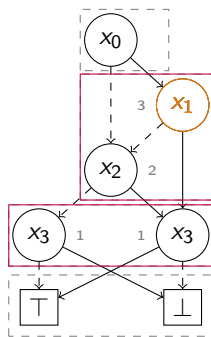


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

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
9	7



## CountPaths : *Example*

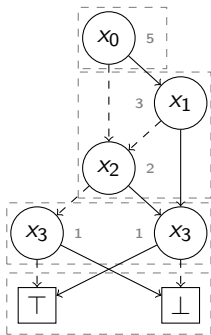


**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

$$M = 4, B = 2$$

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

What are Binary Decision Diagrams?

Why do they break?

How can we fix it?

- CountPaths

- Apply

- Equality Checking

# Contents

What are Binary Decision Diagrams?

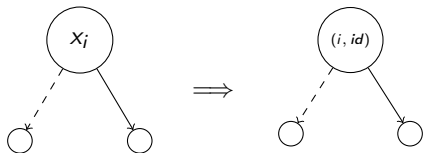
Why do they break?

How can we fix it?

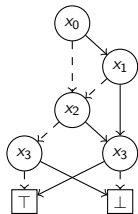
CountPaths

Apply

Equality Checking

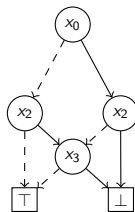


$$(i_1, id_1) < (i_2, id_2) \equiv i_1 < i_2 \vee (i_1 = i_2 \wedge id_i < id_j)$$



[ ((0, 0), (2, 0), (1, 0)) ,  
 ((1, 0), (2, 0), (3, 1)) ,  
 ((2, 0), (3, 0), (3, 1)) ,  
 ((3, 0), ⊤, ⊥) ,  
 ((3, 1), ⊥, ⊤) ]

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



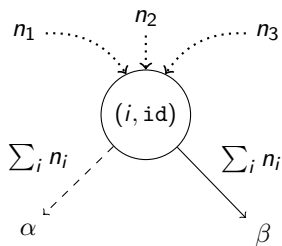
[ ((0, 0), (2, 0), (2, 1)) ,  
 ((2, 0), ⊤, (3, 0)) ,  
 ((2, 1), (3, 0), ⊥) ,  
 ((3, 0), ⊤, ⊥) ]

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

Node-based representation of prior shown BDDs



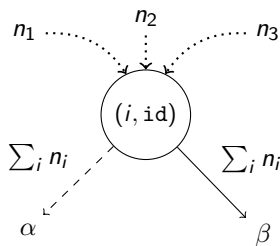
## CountPaths



### Idea

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

## CountPaths



## Time-Forward Processing

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

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

## CountPaths : *Example*



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

## CountPaths : *Example*



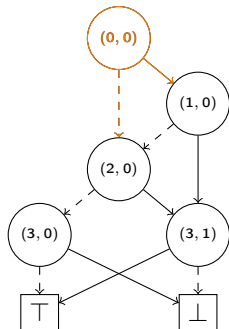
**(a)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

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

Priority Queue:  $Q_{count}$ :

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

]

## CountPaths : *Example*



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

Seek	Sum	Result
(1, 0)	0	0

Priority Queue:  $Q_{count}$ :

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

]

## CountPaths : *Example*



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

Seek	Sum	Result
(1, 0)	0	0

Priority Queue:  $Q_{count}$ :

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

]



## CountPaths : *Example*



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

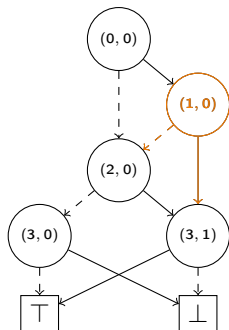
Seek	Sum	Result
(1, 0)	1	0

Priority Queue:  $Q_{count}$ :

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

]

## CountPaths : *Example*



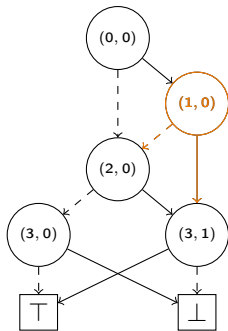
(a)  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

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*



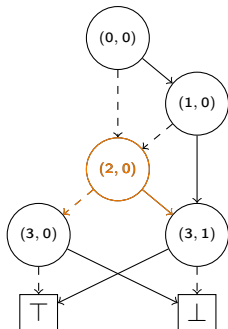
(a)  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	0	0

Priority Queue:  $Q_{count}$ :

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

## 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:  $Q_{count}$ :

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

## CountPaths : *Example*



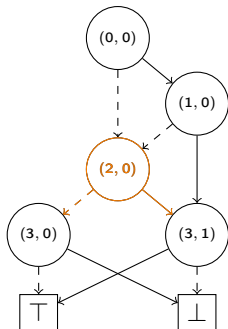
(a)  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Seek	Sum	Result
(2, 0)	1	0

Priority Queue:  $Q_{count}$ :

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

## CountPaths : *Example*



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

Seek	Sum	Result
(2, 0)	2	0

Priority Queue:  $Q_{count}$ :

[

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

## CountPaths : *Example*



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

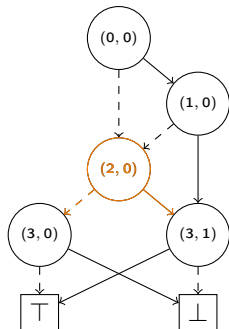
Seek	Sum	Result
(2, 0)	2	0

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

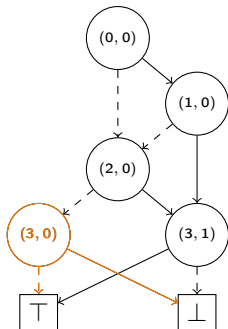
Priority Queue:  $Q_{count}$ :

[

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



## 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:  $Q_{count}$ :

[

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

## CountPaths : *Example*



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

Seek	Sum	Result
(3, 0)	2	2

Priority Queue:  $Q_{count}$ :

[

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

## CountPaths : *Example*



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

Seek	Sum	Result
(3, 1)	0	2

Priority Queue:  $Q_{count}$ :

[

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

## CountPaths : *Example*



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

Seek	Sum	Result
<b>(3, 1)</b>	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:  $Q_{count}$ :

[

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

## 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

What are Binary Decision Diagrams?

Why do they break?

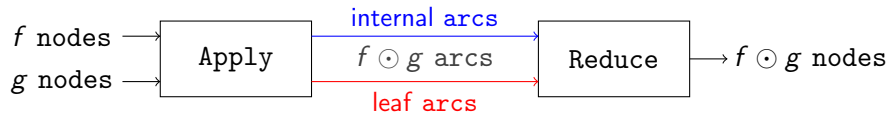
How can we fix it?

CountPaths

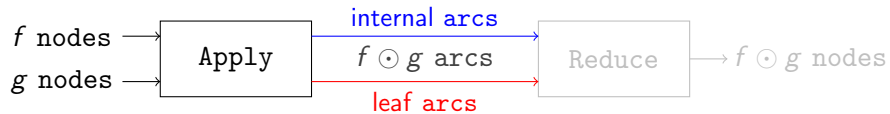
Apply

Equality Checking

# Apply



# Apply



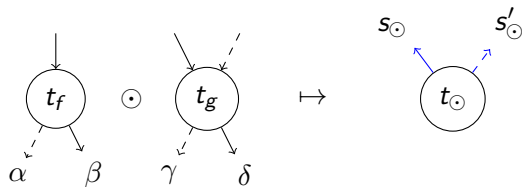
## Apply



## Time-Forward Processing

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

## Apply



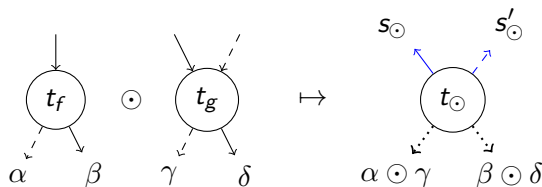
### Observation (semi-transposition)

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

## Time-Forward Processing

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

## Apply



### Observation (semi-transposition)

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

## Time-Forward Processing

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

## 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  $Q_{app:1}, Q_{app:2} : \text{PriorityQueue}\langle (s \rightarrow (t_f, t_g), \dots) \rangle$ .

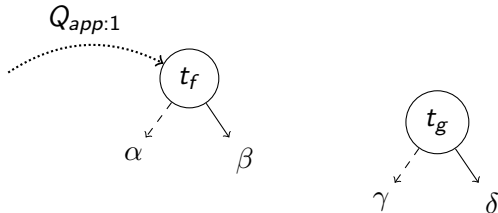


## Apply

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

**Case 1** :

$t_f.\text{var}() \neq t_g.\text{var}()$

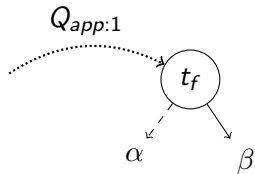


# Apply

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

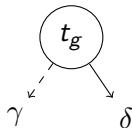
**Case 1** :

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



**Case 2(a):**

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



# Apply

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

**Case 1** :

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



$Q_{app:2}$ : PriorityQueue $\langle (s \rightarrow (t_f, t_g), (\alpha, \beta)) \rangle$   
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)**  $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$



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

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

# Apply : Example



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



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

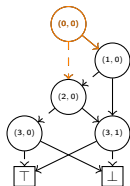
Priority Queue:  $Q_{app:1}$ :

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



(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:

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

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

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



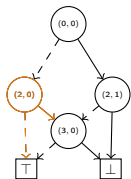
]

(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:

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

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

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

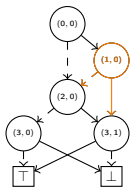


]

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



# Apply : Example



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



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

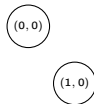
Seek:

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

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

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

]



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

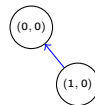
Priority Queue:  $Q_{app:1}$ :

[

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

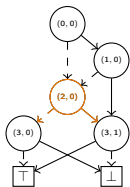
]

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



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

# Apply : Example



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



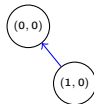
(b)  $\neg(x_0 ? x_2 \vee x_3 : x_2 \wedge x_3)$

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

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

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

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:  $Q_{app:1}$ :

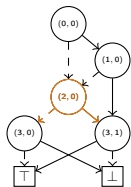
[  
 $(0, 0) \xrightarrow{\perp} ((2, 0), (2, 0))$  ,  
 $(1, 0) \xrightarrow{\perp} ((2, 0), (2, 1))$  ,  
 $(1, 0) \xrightarrow{\top} ((3, 1), (2, 1))$  ,  
 $(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

Output:



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

# Apply : Example



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



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

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

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

[

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

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

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

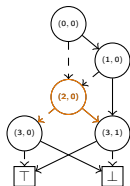
$(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

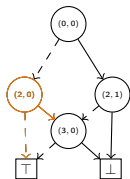


(c)  $(a) \wedge (b)$

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



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

Seek:

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

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

[

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

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

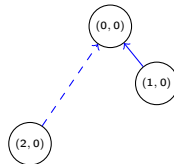
$(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

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

]

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:

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

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

[

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

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

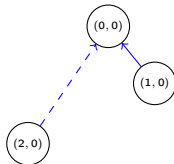
$(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

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

]

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

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:  $Q_{app:2}$ :

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

]

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



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

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:  $Q_{app:1}$ :

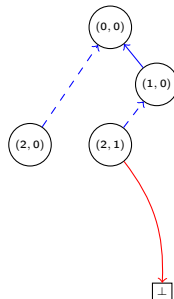
[

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

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

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

Output:



(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:  $Q_{app:1}$ :

[

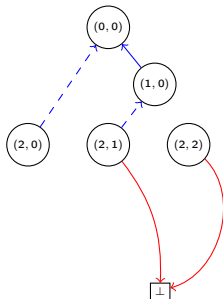
$(1, 0) \xrightarrow{\top} ((3, 1), (2, 1))$  ,  
 $(2, 1) \xrightarrow{\perp} ((3, 0), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 2) \xrightarrow{\perp} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

[

]

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



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

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

[

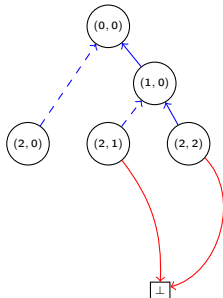
$(2, 1) \xrightarrow{\perp} ((3, 0), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 2) \xrightarrow{\perp} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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:  $Q_{app:1}$ :

[

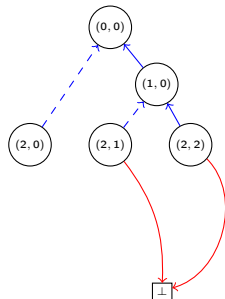
$(2, 1) \xrightarrow{\perp} ((3, 0), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 2) \xrightarrow{\perp} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

[

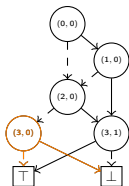
]

Output:

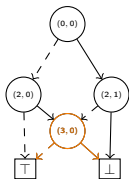


(c)  $(a) \wedge (b)$

# 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:  $Q_{app:1}$ :

[

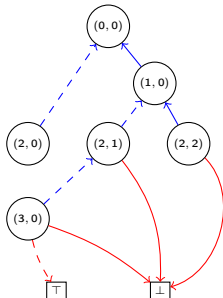
$(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 2) \xrightarrow{\perp} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

[

]

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



(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:  $Q_{app:1}$ :

[

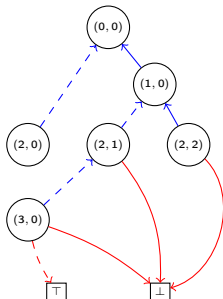
$(2, 0) \xrightarrow{\top} ((3, 1), (3, 0))$  ,  
 $(2, 2) \xrightarrow{\perp} ((3, 1), (3, 0))$  ,  
 $(2, 0) \xrightarrow{\perp} ((3, 0), \top)$  ]

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

[

]

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:  $Q_{app:1}$ :

[

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

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

[

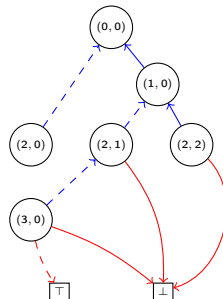
$(2, 0) \xrightarrow{\top} ((3, 1), (3, 0)) \quad (\top, \perp)$

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

,

]

Output:



(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:  $Q_{app:2}$ :

[

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

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

,

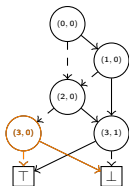
]

Output:

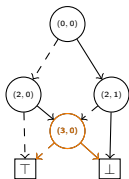


(c)  $(a) \wedge (b)$

# 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:  $Q_{app:2}$ :

[

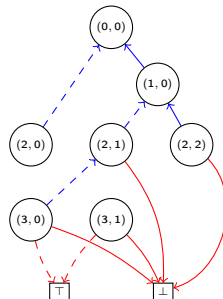
$(2, 0) \xrightarrow{\top} ((3, 1), (3, 0)) \quad (\top, \perp)$

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

,

]

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:  $Q_{app:1}$ :

[

]

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

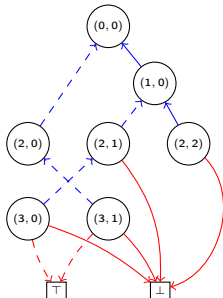
[

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

,

]

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



(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:  $Q_{app:2}$ :

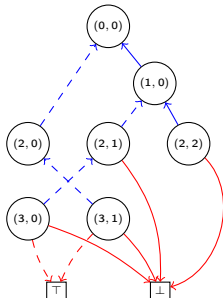
[

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

,

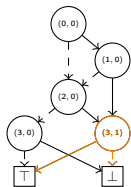
]

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

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

[

]

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

[

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

,

]

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



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

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:  $Q_{app:1}$ :

[

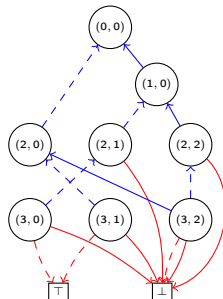
Priority Queue:  $Q_{app:2}$ :

[

]

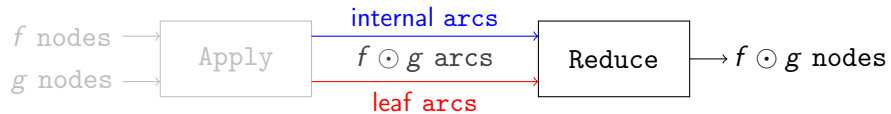
]

Output:



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

# Apply



## Apply (Reduce)



## Time-Forward Processing

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

## Apply (Reduce)



## Time-Forward Processing

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

## Apply (Reduce)



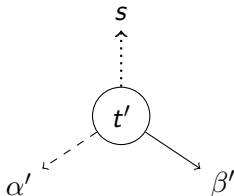
## Time-Forward Processing

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

### Observation (semi-transposition)

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

## Apply (Reduce)



## Time-Forward Processing

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

### Observation (semi-transposition)

$\leftarrow : s \rightarrow t$  (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)

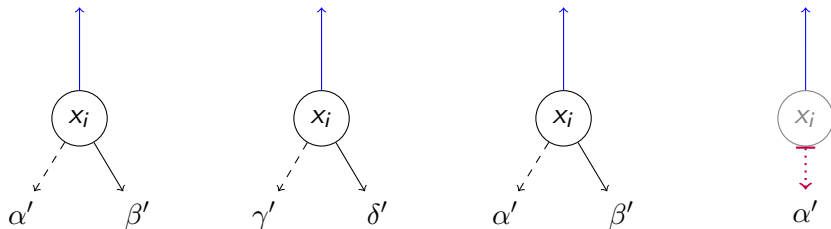


Reduce Level  $i$ :

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



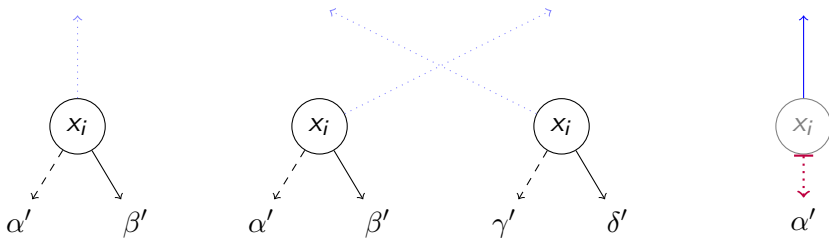
## Apply (Reduce)



Reduce Level  $i$ :

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

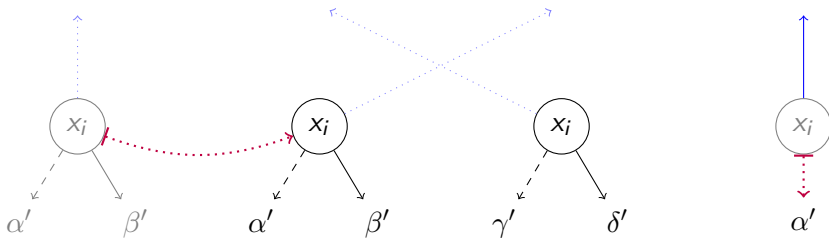
## Apply (Reduce)



### 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.

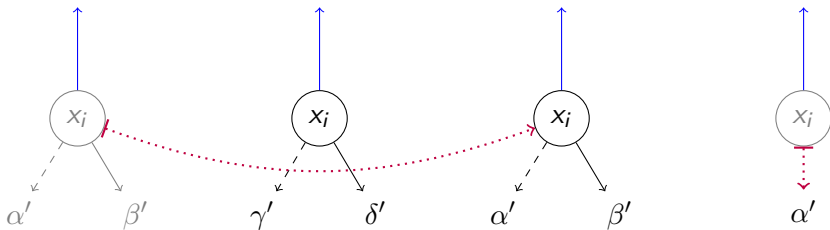
## Apply (Reduce)



### 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  
 $[ \quad [(3, 2) \mapsto \perp] \quad ]$   
 $[ \quad ((3, 1), \top, \perp) \quad , \quad ]$

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



## Apply (Reduce) : *Example*



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

Level: 3  
 $[ \quad [(3, 2) \mapsto \perp] \quad ]$   
 $[ \quad ((3, 1), \top, \perp) \quad , \quad ((3, 0), \top, \perp) \quad ]$

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

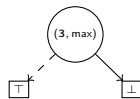
## Apply (Reduce) : *Example*



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

Level: 3  
 $[ \quad [(3, 2) \mapsto \perp] \quad ]$   
 $[ \quad [ (3, 1) \mapsto (3, \max) ] \quad , \quad ]$   
 $\quad \quad \quad ((3, 0), \top, \perp)$

Output:  
 $((3, \max), \top, \perp)$



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

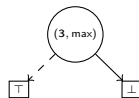
## Apply (Reduce) : *Example*



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

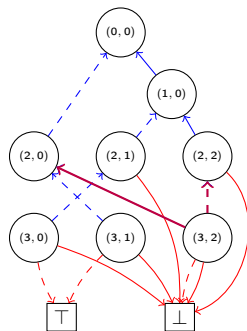
Level: 3  
 $\left[ \begin{array}{c} [(3, 2) \mapsto \perp] \\ [(3, 1) \mapsto (3, \max)] \\ [(3, 0) \mapsto (3, \max)] \end{array} \right]$

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

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

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

]

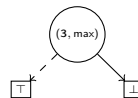
Level: 3

[

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

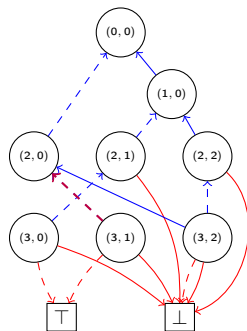
[  $(3, 0) \mapsto (3, \max)$  ] ]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

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

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

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

]

Level: 3

[

]

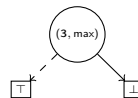
[

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

,

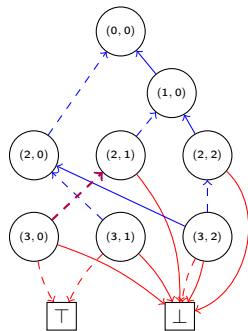
]

Output:



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

## Apply (Reduce) : *Example*



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

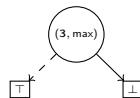
Priority Queue:  $Q_{red}$ :

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

Level: 3

[  
 [  
 ,  
 ]

Output:



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

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

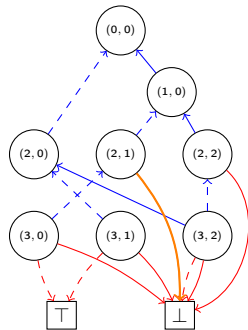
[  
 Level: 2  
 $[(2, 2) \mapsto \perp]$   
 ]

Output:



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

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

]

Level: 2

[

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

]

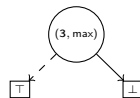
[

$((2, 1), (3, \max), \perp)$

,

]

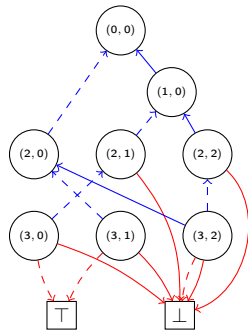
Output:



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



# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

Level: 2

[

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

]

[

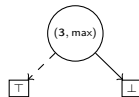
$((2, 1), (3, \max), \perp)$

,

$((2, 0), (3, \max), \perp)$

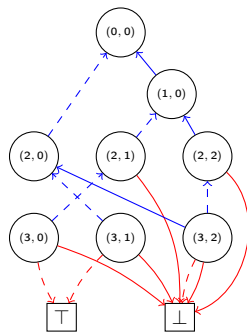
]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

Level: 2

[

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

]

[

$[(2, 1) \mapsto (2, \max)]$

,

$((2, 0), (3, \max), \perp)$

]

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



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

Level: 2

[

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

]

[

$[(2, 1) \mapsto (2, \max)]$

,

$[(2, 0) \mapsto (2, \max)]$

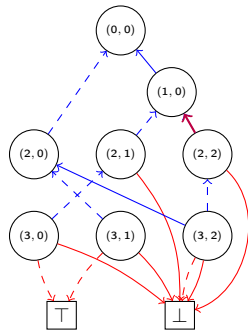
]

Output:



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

]

Level: 2

[

]

[

$[(2, 1) \mapsto (2, \max)]$

,

$[(2, 0) \mapsto (2, \max)]$

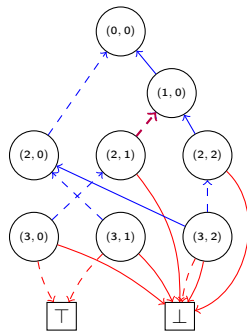
]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

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

]

Level: 2

[

]

[

$[(2, 0) \mapsto (2, \max)]$

,

]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

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

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

Level: 2

[

[

]

,  
]

Output:



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

Level: 1

[

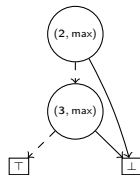
]

[

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

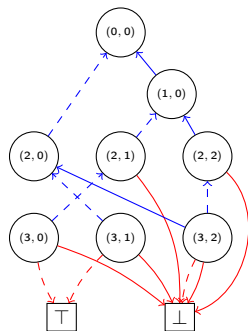
]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

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

Level: 1

[

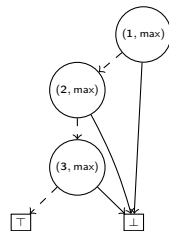
]

[

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

]

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

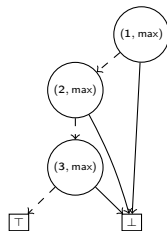
[

]

[

]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

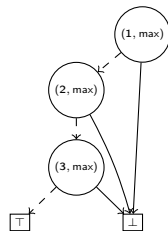
Level: 0

[

]

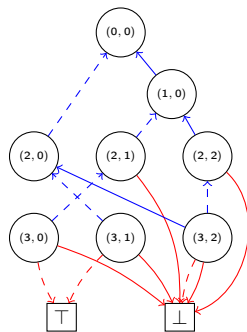
[  $((0, 0), (2, \max), (1, \max))$  ]

Output:



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

# Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

Level: 0

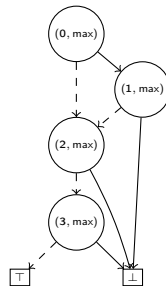
[

]

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

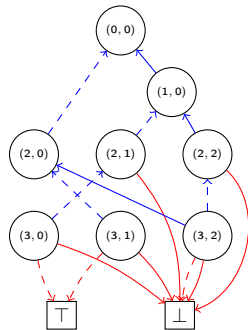
]

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



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

## Apply (Reduce) : *Example*



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

Priority Queue:  $Q_{red}$ :

[

]

Level: 0

[

]

[

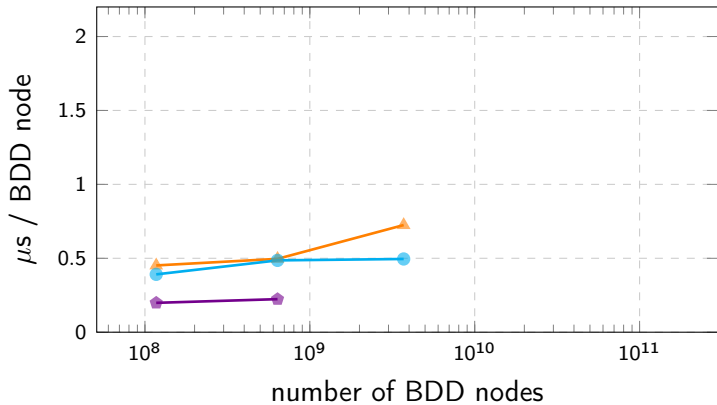
]

Output:



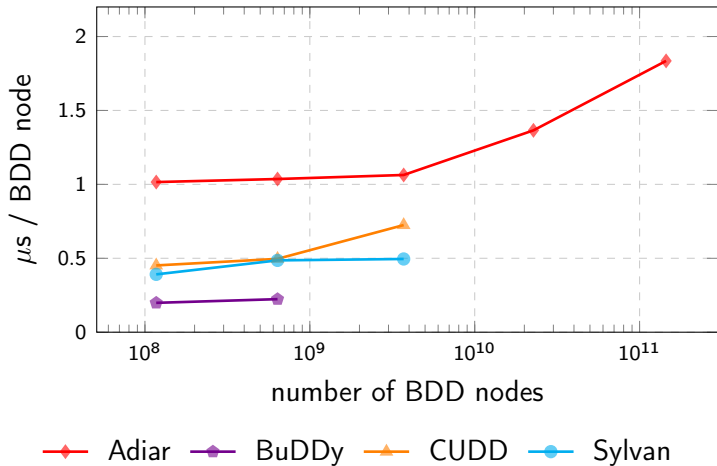
(d)  $(a) \wedge (b)$  reduced

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	$O(\text{sort}(N_f \cdot N_g))$



—◆— Adiar —◆— BuDDy —◆— CUDD —◆— Sylvan

Running time for the *N-Queens* problems.



Running time for the *N-Queens* problems.

# Contents

What are Binary Decision Diagrams?

Why do they break?

How can we fix it?

CountPaths

Apply

Equality Checking



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	$O(\text{sort}(N_f))$
bdd_not	$2N_f/B$
bdd_restrict	$O(\text{sort}(N_f))$
bdd_apply	$O(\text{sort}(N_f \cdot N_g))$
bdd_equal	?

## Equality Checking

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

## Equality Checking

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

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

## Equality Checking

### Theorem (Bryant '86)

*Let  $\pi$  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  $\pi$ .*

# Equality Checking

## Theorem (Bryant '86)

*Let  $\pi$  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  $\pi$ .*

**Trivial cases:  $f \neq g$  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  $i$ th level       $O(L/B)$  I/Os

# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .



# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .

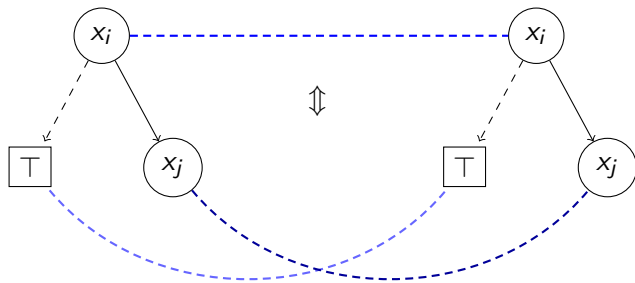




# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .



# Equality Checking

## Theorem (Bryant '86)

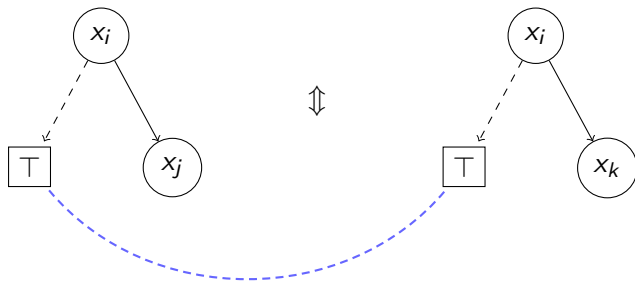
Let  $\pi$  be a variable order and  $f : \mathbb{B}^n \rightarrow \mathbb{B}$  then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing  $f$  with ordering  $\pi$ .



# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .



# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .



## Equality Checking

### Theorem (Bryant '86)

*Let  $\pi$  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  $\pi$ .*

`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  $\pi$  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  $\pi$ .

$\text{IsIsomorphic}(f, g)$

- Check whether root  $v_f$  of  $f$  and root  $v_g$  of  $g$  have a local violation.
- Check  $\text{low}(v_f) \sim \text{low}(v_g)$  and  $\text{high}(v_f) \sim \text{high}(v_g)$  “recursively”.

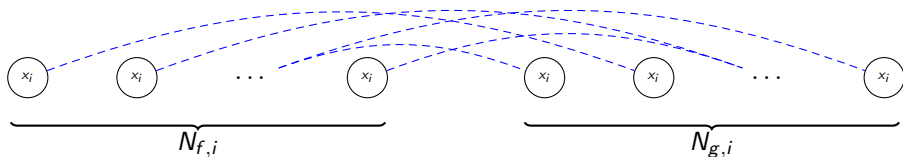
Return false on first violation. If there are no violations then return true.

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

# Equality Checking

## Theorem (Bryant '86)

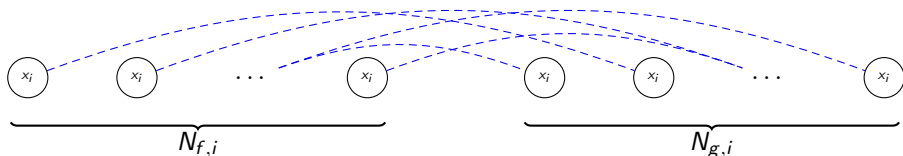
Let  $\pi$  be a variable order and  $f : \mathbb{B}^n \rightarrow \mathbb{B}$  then there exists a unique (up to isomorphism) Reduced Ordered Binary Decision Diagram representing  $f$  with ordering  $\pi$ .



# Equality Checking

## Theorem (Bryant '86)

Let  $\pi$  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  $\pi$ .



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

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



# 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:

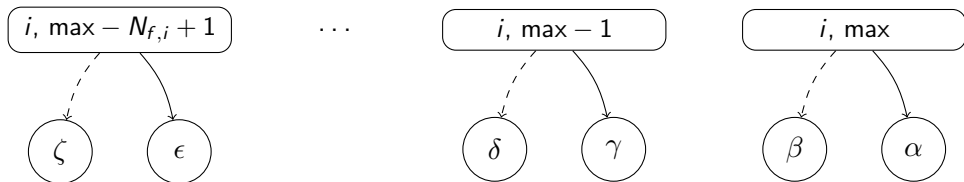
## Equality Checking



### Observation

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

# Equality Checking

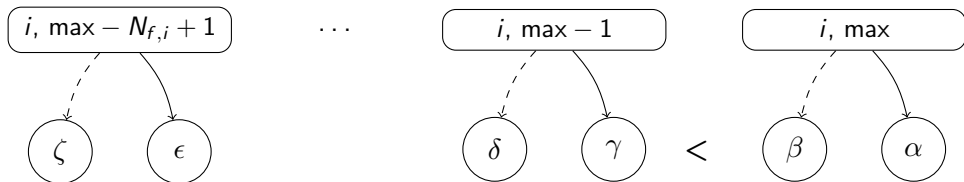


## Observation

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

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

# Equality Checking

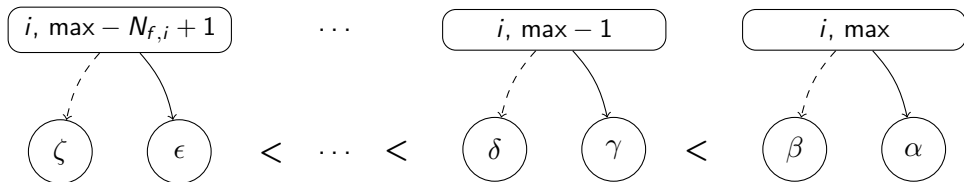


## Observation

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.*

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

## Proof.

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

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



# Equality Checking

## Theorem

*If  $G_f$  and  $G_g$  are outputs of Reduce.*

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

## Proof.

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

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

## 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](mailto:soelvsten@cs.au.dk)

🌐 [ssoelvsten.github.io](https://ssoelvsten.github.io)

## Adiar

🔗 [github.com/ssoelvsten/adiar](https://github.com/ssoelvsten/adiar)

📖 [ssoelvsten.github.io/adiar](https://ssoelvsten.github.io/adiar)



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