Efficient External Memory Algorithms for Binary Decision Diagram Manipulation

Steffan Christ Sølvsten, Jaco van de Pol,
Anna Blume Jakobsen, and Mathias Weller Berg Thomasen

Aarhus University

April 27, 2021

Section 1

Preliminaries

I/O Model

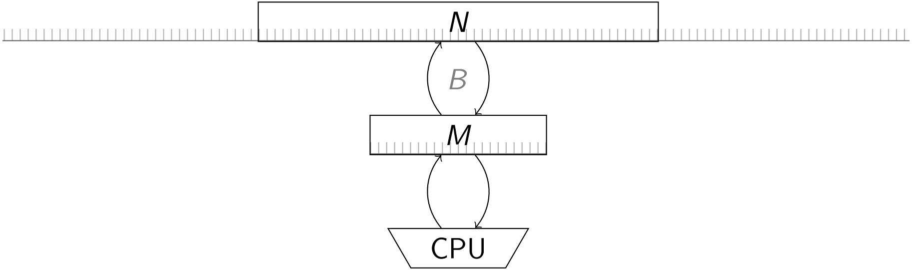


Figure: The I/O model by Aggarwal and Vitter [AV87]

I/O Model

For any realistic values of N, M, and B we have that [AV87; Arg04]

$$B^2 \leq M < N \qquad N/B < \text{sort}(N) \triangleq N/B \cdot \log_{M/B} N/B \ll N \ .$$

Theorem (Aggarwal and Vitter [AV87])

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

Theorem (Arge [Arg95])

N elements can be inserted in and extracted from a Priority Queue in $\Theta(\text{sort}(N))$ I/Os.

Binary Decision Diagrams

BDDs are a graph-based representation of functions $\mathbb{B}^n \rightarrow \mathbb{B}$ by Bryant [Bry86].

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

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

Figure: Examples of (Reduced Ordered) Binary Decision Diagrams.

Binary Decision Diagrams

Theorem (Bryant [Bry86])

For a fixed variable order, if one exhaustively applies the following two rules

1. *skip all nodes with identical children*
2. *remove any duplicate nodes*

then one obtains the Reduced OBDD, which is a unique canonical form of the function.

Binary Decision Diagrams

Theorem (Bryant [Bry86])

For a fixed variable order, if one exhaustively applies the following two rules

1. *skip all nodes with identical children*
2. *remove any duplicate nodes*

then one obtains the Reduced OBDD, which is a unique canonical form of the function.

The two primary algorithms for BDDs are:

- *Apply*: Given f, g : ROBDD and $\odot : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ constructs $f \odot g$: OBDD (create the product construction of f and g and apply $\odot$ on pairs of leaves).
- *Reduce*: Given f : OBDD constructs f : ROBDD (apply bottom-up the above reduction rules).

Binary Decision Diagrams

Common implementations are done with

- Recursive depth-first procedures.
- Memoization table(s) to not reevaluate prior visited nodes.
- Unique nodes managed with a hash table with : $\mathbb{N} \times \mathit{Node} \times \mathit{Node} \to \mathit{Node}$.
  This takes care of both reduction rules within the `find-or-insert` procedure.
  - Garbage Collection

Binary Decision Diagrams

Common implementations are done with

- Recursive depth-first procedures.
- Memoization table(s) to not reevaluate prior visited nodes.
- Unique nodes managed with a hash table with : $\mathbb{N} \times \mathit{Node} \times \mathit{Node} \to \mathit{Node}$.
  This takes care of both reduction rules within the `find-or-insert` procedure.
  - Garbage Collection

Lemma ([Arg96; Sø+21])

For N_f and N_g large enough, these depth-first implementations of BDDs make for the following worst-case I/O behaviour.

Apply: $O(N_f \cdot N_g)$ *I/Os* *Reduce:* $O(N)$ *I/Os*

Section 2

I/O-efficient BDD algorithms

I/O-efficient BDD algorithms

Lars Arge suggested in [Arg95] (see [Arg96] for all the details) to drop the unique node table. Instead, he used the technique of *Time-forward Processing*:

- Sort nodes for each BDD based with a levelized *total* and *topological* ordering.
- Use priority queues to postpone recursion requests until the nodes are encountered.

I/O-efficient BDD algorithms

Lars Arge suggested in [Arg95] (see [Arg96] for all the details) to drop the unique node table. Instead, he used the technique of *Time-forward Processing*:

- Sort nodes for each BDD based with a levelized *total* and *topological* ordering.
- Use priority queues to postpone recursion requests until the nodes are encountered.

Theorem (Arge [Arg96])

Given f is ordered levelized, the Reduce operation is computable in $\Theta(sort(N_f))$ I/Os.

Theorem (Arge [Arg96])

Given f and g are ordered levelized, the Apply operation (followed by Reduce) is computable in $\Theta(sort(N_f \cdot N_g))$ I/Os.

Levelized BDD Representation

Let every node be uniquely identified by a tuple $(label, id) : \mathbb{N} \times \mathbb{N}$.

$$(i_1, id_1) < (i_2, id_2) \equiv i_1 < i_2 \lor (i_1 = i_2 \land id_i < id_j)$$

Lemma

If the id is unique per level, then the above is a total and topological ordering.

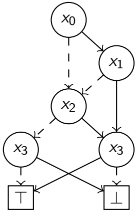
Proof.

A Binary Decision Diagram is a DAG where each level corresponds to the *label*. Hence, the ordering is topological. Totality follows from *id* being unique within each level. $\square$

Levelized BDD Representation

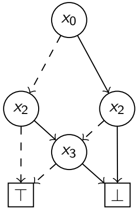
A node is represented as a tuple $(uid, low, high)$, where

- $uid : \mathbb{N} \times \mathbb{N}$
- $low, high : (\mathbb{N} \times \mathbb{N}) + \mathbb{B}$



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

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

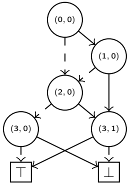


$$[\ ((0,0),(2,0),(2,1)),\ ((2,0),\top,(3,0)),\ ((2,1),(3,0),\bot),\ ((3,0),\top,\bot)\ ]$$

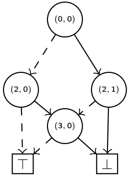
(b) $\neg(x_0\ ?\ x_2 \lor x_3\ :\ x_2 \land x_3)$

Figure: Node-based representation of prior shown BDDs

BDD Manipulation by Time-forward Processing : Apply ($\wedge$)

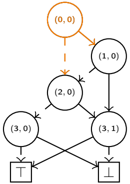


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

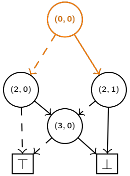


(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

BDD Manipulation by Time-forward Processing : Apply ($\wedge$)

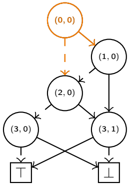


(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

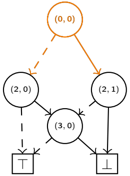


(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

BDD Manipulation by Time-forward Processing : Apply (∧)



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

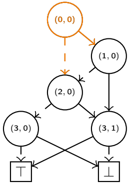


(b) $\neg(x_0 \ ? \ x_2 \lor x_3 \ : \ x_2 \land x_3)$

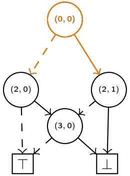
Priority Queue: $Q_{app:1}$:

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

BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



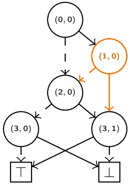
(b) $\neg(x_0\ ?\ x_2 \vee x_3\ :\ x_2 \wedge x_3)$

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

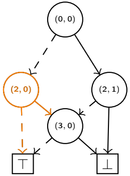
Priority Queue: $Q_{app:1}$:

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

BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



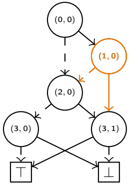
(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

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

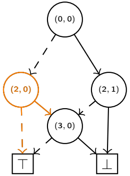
Priority Queue: $Q_{app:1}$:

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

BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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

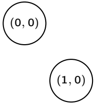


(b) $\neg(x_0\ ?\ x_2 \vee x_3\ :\ x_2 \wedge x_3)$

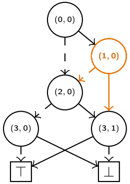
Seek:
$\min((1,0),(2,1))$

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

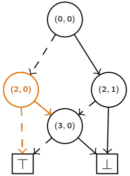
$$[\quad (0,0) \xrightarrow{\top} ((1,0),(2,1)),\quad (0,0) \xrightarrow{\bot} ((2,0),(2,0)),\quad (1,0) \xrightarrow{\bot} ((2,0),(2,1)),\quad (1,0) \xrightarrow{\top} ((3,1),(2,1)),\quad]$$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



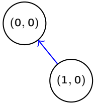
(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

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

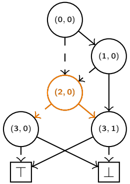
Priority Queue: $Q_{app:1}$:

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

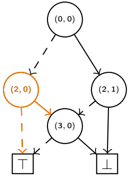
Output:
$(0,0) \xrightarrow{\top} (1,0)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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

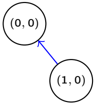
Priority Queue: $Q_{app:1}$:

[

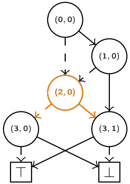
$(0,0) \xrightarrow{\perp} ((2,0),(2,0))$,
$(1,0) \xrightarrow{\perp} ((2,0),(2,1))$,
$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,

]

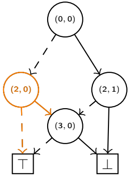
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



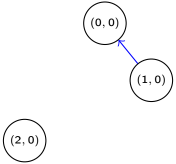
(b) $\neg(x_0 \ ? \ x_2 \vee x_3 \ : \ x_2 \wedge x_3)$

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

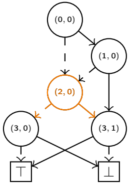
Priority Queue: $Q_{app:1}$:

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

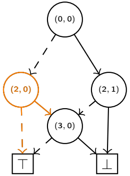
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



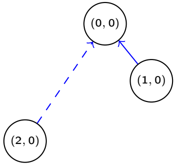
(b) $\neg(x_0 \;?\; x_2 \vee x_3 \;:\; x_2 \wedge x_3)$

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

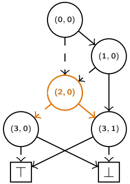
Priority Queue: $Q_{app:1}$:

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

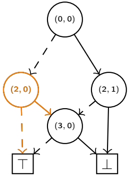
Output:
$(0,0) \xrightarrow{\perp} (2,0)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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



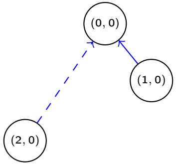
(b) $\neg(x_0 \; ? \; x_2 \vee x_3 \; : \; x_2 \wedge x_3)$

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

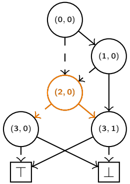
Priority Queue: $Q_{app:1}$:

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

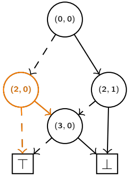
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

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

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

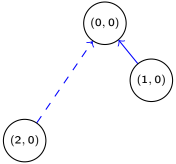
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

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

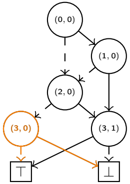
[$(1,0) \xrightarrow{\perp} ((2,0),(2,1))$ $((3,0),(3,1))$,

]

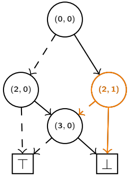
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

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

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

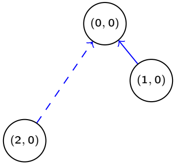
$(2, 0) \xrightarrow{\perp} ((3, 0), \top)$]

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

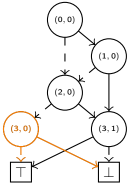
[$(1, 0) \xrightarrow{\perp} ((2, 0), (2, 1))$ $((3, 0), (3, 1))$,

]

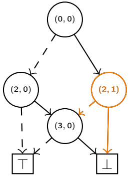
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

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

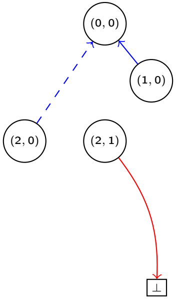
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

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

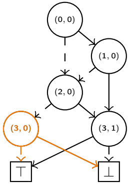
[$(1,0) \xrightarrow{\perp} ((2,0),(2,1)) \quad ((3,0),(3,1))$,

]

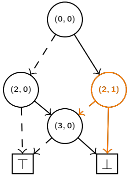
Output:
$(2,1) \xrightarrow{\top} \perp$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

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

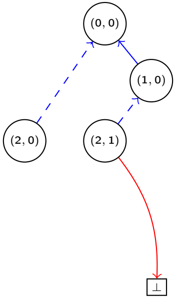
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

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

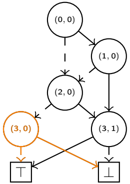
[

]

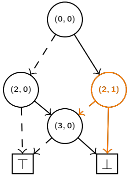
Output:
$(1,0) \xrightarrow{\bot} (2,1)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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))

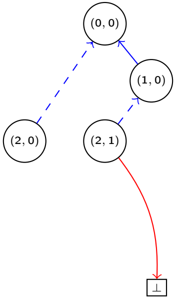
Priority Queue: $Q_{app:1}$:

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

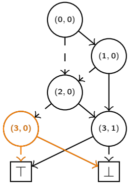
Priority Queue: $Q_{app:2}$:

[ ]

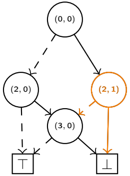
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

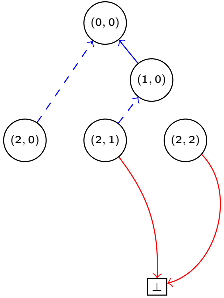
$(1,0) \xrightarrow{\top} ((3,1),(2,1))$,
$(2,1) \xrightarrow{\bot} ((3,0),(3,0))$,
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\bot} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

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

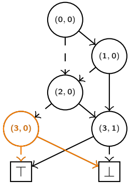
[

]

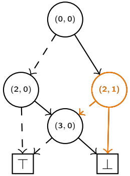
Output:
$(2,2) \xrightarrow{\top} \bot$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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

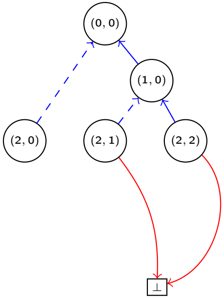
Priority Queue: $Q_{app:1}$:

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

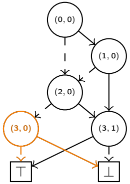
Priority Queue: $Q_{app:2}$:

[ ]

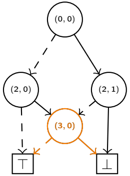
Output:
$(1,0) \xrightarrow{\top} (2,2)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

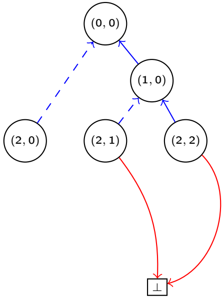
$$
\begin{aligned}
[\ & \\
& (2,1) \xrightarrow{\perp} ((3,0),(3,0)) \quad , \\
& (2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad , \\
& (2,2) \xrightarrow{\perp} ((3,1),(3,0)) \quad , \\
& (2,0) \xrightarrow{\perp} ((3,0),\top) \quad]
\end{aligned}
$$

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

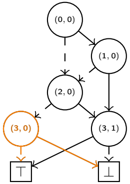
[

]

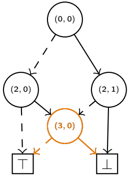
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



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

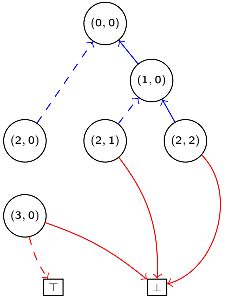
Priority Queue: $Q_{app:1}$:

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

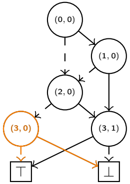
Priority Queue: $Q_{app:2}$:

[ ]

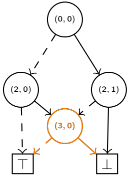
Output:
$(3,0) \xrightarrow{\perp} \top, (3,0) \xrightarrow{\top} \perp$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

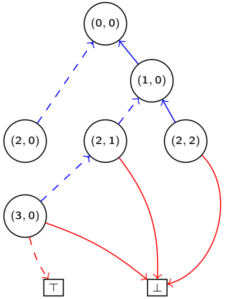
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\perp} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\perp} ((3,0),\top)$]

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

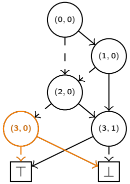
[

]

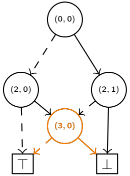
Output:
$(2,1) \xrightarrow{\perp} (3,0)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

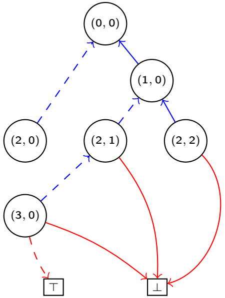
$(2,0) \xrightarrow{\top} ((3,1),(3,0))$,
$(2,2) \xrightarrow{\bot} ((3,1),(3,0))$,
$(2,0) \xrightarrow{\bot} ((3,0),\top)$]

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

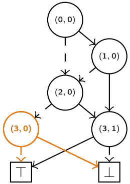
[

]

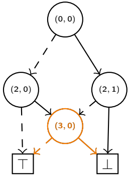
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

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

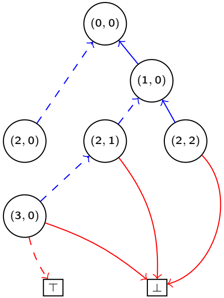
Priority Queue: $Q_{app:2}$:

[

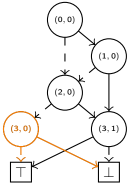
$(2,0) \xrightarrow{\top} ((3,1),(3,0)) \quad (\top, \bot)$,

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

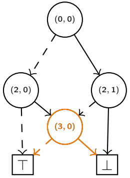
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

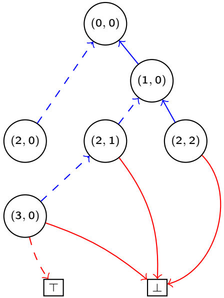
$(2,0) \xrightarrow{\perp} ((3,0), \top)$]

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

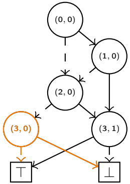
[

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

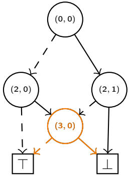
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

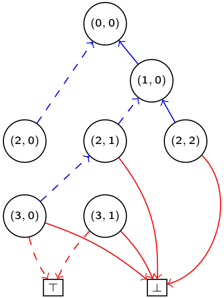
$(2,0) \xrightarrow{\perp} ((3,0), \top)$]

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

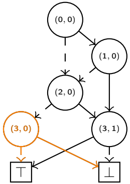
[

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

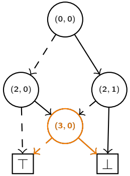
Output:
$(3,1) \xrightarrow{\perp} \top, (3,1) \xrightarrow{\top} \perp$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

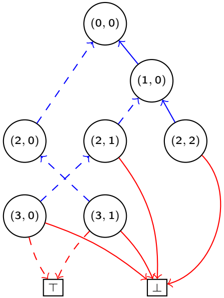
[

]

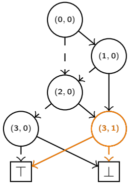
Priority Queue: $Q_{app:2}$:

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

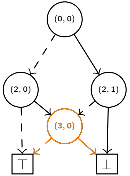
Output:
$(2,0) \xrightarrow{\bot} (3,1)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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:

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

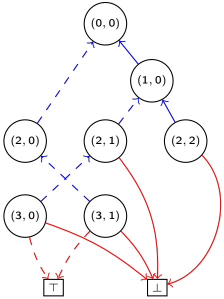
Priority Queue: $Q_{app:1}$:

[ ]

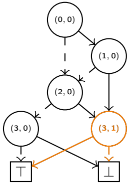
Priority Queue: $Q_{app:2}$:

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

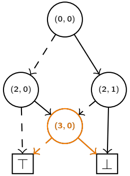
Output:



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

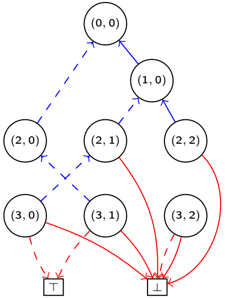
[

]

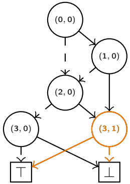
Priority Queue: $Q_{app:2}$:

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

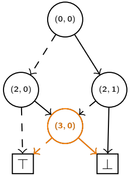
Output:
$(3,2) \xrightarrow{\bot} \bot, (3,2) \xrightarrow{\top} \bot$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

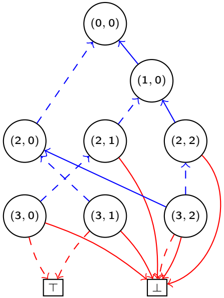
[ ]

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

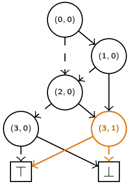
[ ]

Output:

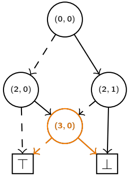
$(2,0) \xrightarrow{\top} (3,2), (2,2) \xrightarrow{\perp} (3,2)$



BDD Manipulation by Time-forward Processing : Apply ($\wedge$)



(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: $Q_{app:1}$:

[

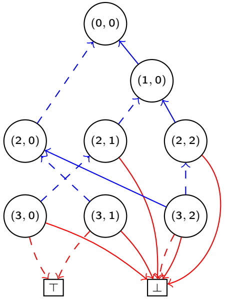
]

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

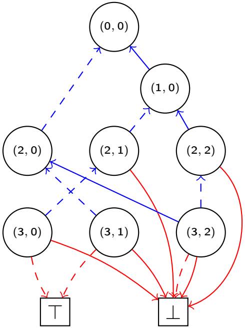
[

]

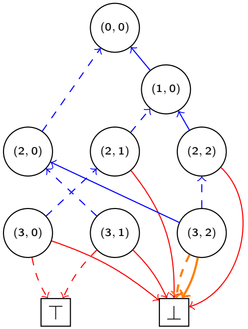
Output:



BDD Manipulation by Time-forward Processing : Reduce



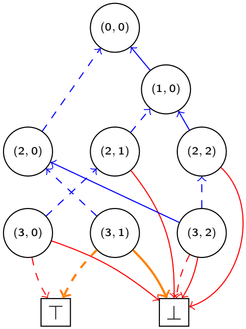
BDD Manipulation by Time-forward Processing : Reduce



Level: 3

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

BDD Manipulation by Time-forward Processing : Reduce

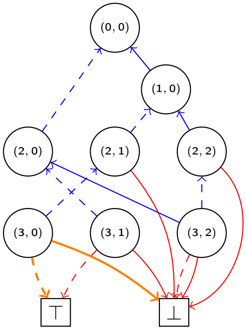


Level: 3

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

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

BDD Manipulation by Time-forward Processing : Reduce

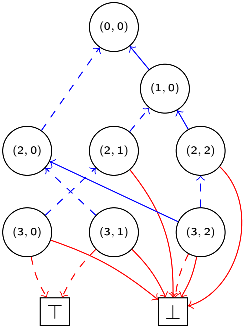


Level: 3

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

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

BDD Manipulation by Time-forward Processing : Reduce

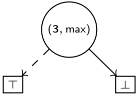


Output:
$((3, \max), \top, \bot)$

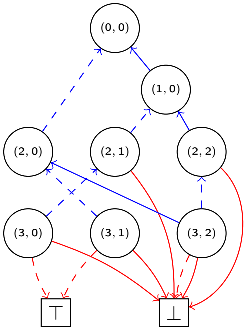
Level: 3

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

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



BDD Manipulation by Time-forward Processing : Reduce

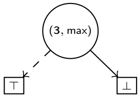


Level: 3

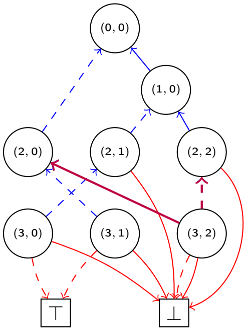
[$[(3,2) \mapsto \bot]$]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

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

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

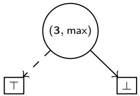
]

Level: 3

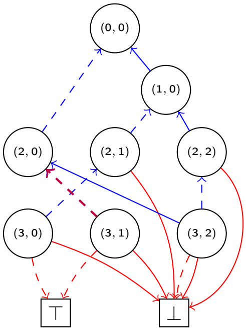
[ ]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

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

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

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

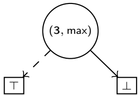
$]$

Level: 3

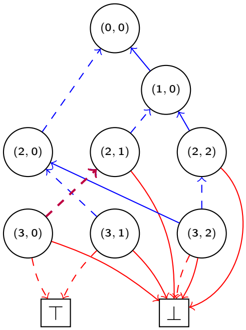
$[\quad]$

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

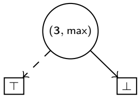
$$
\begin{array}{l}
[\quad (2,2) \xrightarrow{\perp} \perp \quad , \\
\quad (2,1) \xrightarrow{\perp} (3,\max) \quad , \\
\quad (2,0) \xrightarrow{\top} \perp \quad , \\
\quad (2,0) \xrightarrow{\perp} (3,\max) \quad , \\
\quad]
\end{array}
$$

Level: 3

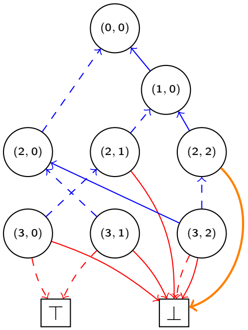
[ ]

[,
]

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}**:**

[

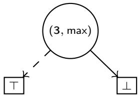
$(2,1) \xrightarrow{\perp} (3, \mathrm{max})$,
$(2,0) \xrightarrow{\top} \perp$,
$(2,0) \xrightarrow{\perp} (3, \mathrm{max})$,

]

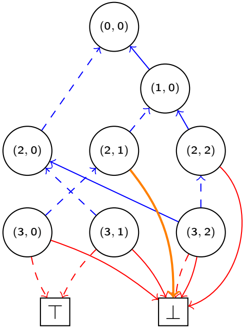
Level: 2

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

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

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

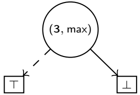
]

Level: 2

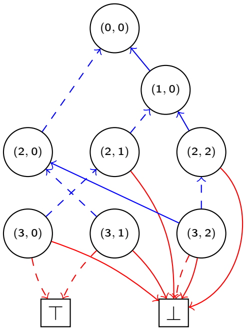
[$[(2,2) \mapsto \bot]$]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

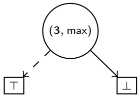
]

Level: 2

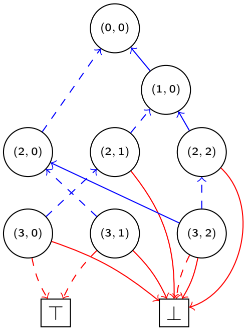
[$[(2,2) \mapsto \bot]$]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

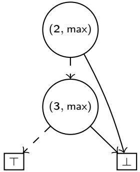
]

Level: 2

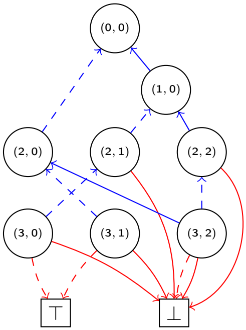
[[(2, 2) ↦ ⊥]]

[[(2, 1) ↦ (2, max)] ,
((2, 0), (3, max), ⊥)]

Output:
((2, max), (3, max), ⊥)



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

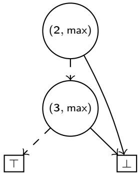
]

Level: 2

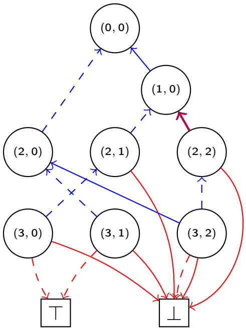
[[(2, 2) $\mapsto$ ⊥]]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

$[$

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

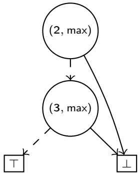
$]$

Level: 2

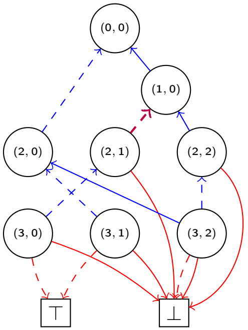
$[$ $]$

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

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

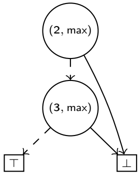
]

Level: 2

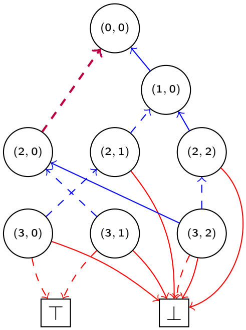
[ ]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

$[$

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

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

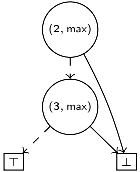
$(0,0) \xrightarrow{\bot} (2,\max)$ $]$

Level: 2

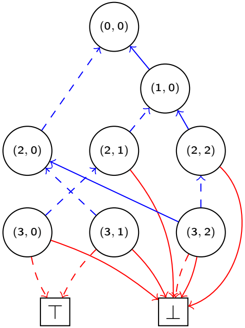
[ ]

[,]

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

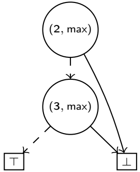
$(0,0) \xrightarrow{\perp} (2, \text{max})$]

Level: 1

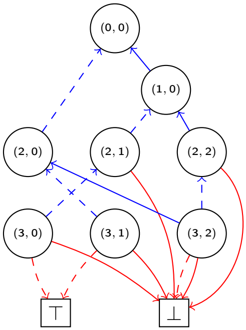
[ ]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

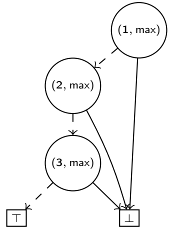
[$(0,0) \xrightarrow{\perp} (2, \max)$]

Level: 1

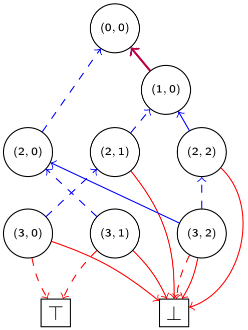
[ ]

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

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



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

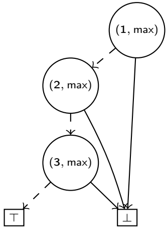
$$[\; (0,0) \xrightarrow{\top} (1,\max) \;, \; (0,0) \xrightarrow{\perp} (2,\max) \;]$$

Level: 1

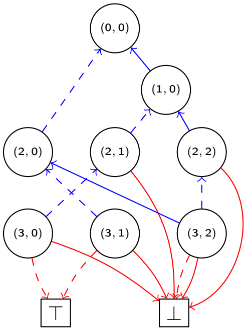
[ ]

[ ]

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

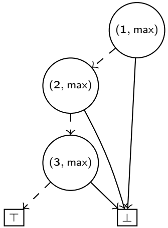
[ ]

Level: 0

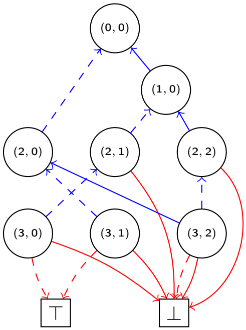
[ ]

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

Output:



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

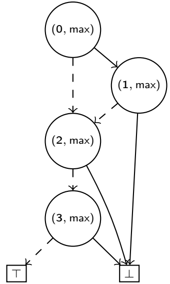
[ ]

Level: 0

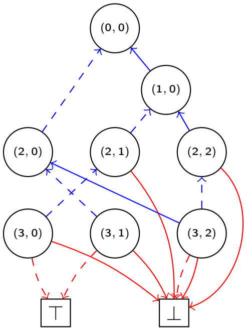
[ ]

[[(0, 0) ↦ (0, max)]]

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



BDD Manipulation by Time-forward Processing : Reduce



Priority Queue: Q_{red}:

[

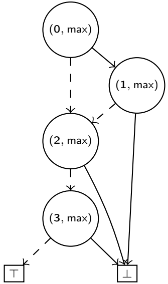
]

Level: 0

[ ]

[ ]

Output:



Our contributions

Ship of Theseus

See the full paper [Sø+21] for the subtle, yet important, improvements we have found for Reduce and Apply compared to the original algorithms in [Arg95; Arg96].

Our contributions: Extending technique to all basic BDD operations

Algorithm		I/O Complexity
Using Time-forward Processing		
If-Then-Else	$f \; ? \; g \; : \; h$	$O(\text{sort}(N_f \cdot N_g \cdot N_h))$
Restrict	$f\|_{x_i = v}$	$O(\text{sort}(N))$
Negation	$\neg f$	$O(1)$
Quantification	$\exists / \forall v : f\|_{x_i = v}$	$O(\text{sort}(N^2))$
Composition	$f\|_{x_i = g}$	$O(\text{sort}(N_f^2 \cdot N_g))$
Count Paths	#paths in f to $\top$	$O(\text{sort}(N))$
Count SAT	$\#x : f(x)$	$O(\text{sort}(N))$
Equality	$f \equiv g$	$O(\text{sort}(N^2))$
Using a single iteration		
Evaluate	$f(x)$	$O(N/B)$
Min/Max SAT	$\min / \max\{x \mid f(x)\}$	$O(N/B)$

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are “frozen” when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

[, , ,]

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are “frozen” when processing level i.

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

Level: 1

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

Level: 2

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

Level: 3

$[\quad , \quad , \quad , \quad]$

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are “frozen” when processing level i.

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

Level: 1

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

Level: 2

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

Level: 3

$[\quad , \quad , \quad , \quad]$

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

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

Level: 3

[, , ,]

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

$[\qquad]$

Level: 2

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

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are “frozen” when processing level i.

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

Level: 1

[ ]

Level: 2

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

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

$[\quad]$

Level: 2

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

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

$[\qquad\qquad]$

Level: 2

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

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

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

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

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

Our contributions: Levelized Priority Queue

The elements for a given level i in the priority queues $Q_{app:1}$ and Q_{red} are "frozen" when processing level i.

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

Level: 1

[ ]

Level: 2

[, ,]

Level: 3

[, , ,]

Our contributions: Memory layout and efficient sorting

The unique identifier of nodes and leafs can be represented in a single 64-bit integer.

1	`v`	`f`

(a) Unique identifier of a leaf `v`

0	`label`	`identifier`	`f`

(b) Unique identifier of an internal node

The `f` bit-flag is used to store the *is_high* boolean inside of the `source` of an arc.

This reduces everything to elements by comparison of integers:

- Top-down traversal: Ascending order
- Bottom-up traversal: Descending order

Section 3

Adiar: An implementation

Adiar: An implementation

Source code:
github.com/ssoelvsten/adiar

Documentation:
ssoelvsten.github.io/adiar

Experimental Evaluation

N-Queens Problem: Count the number of ways N queens can be placed on an $N \times N$ chess board, such that no queen threatens another.

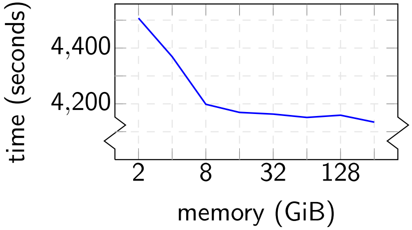
Our implementation follows the description in [KSC10]: Let x_{ij} represent whether a queen is placed on the i'th row and the j'th column. The solution corresponds to the number of satisfying assignments of the following formula.

$$\bigwedge_{i=0}^{n-1} \bigvee_{j=0}^{n-1} (x_{ij} \wedge \neg has\_threat(i,j))$$

Source code:

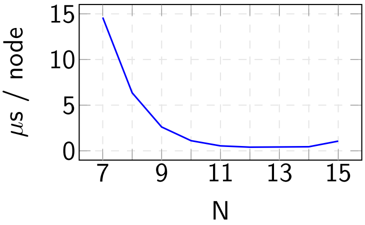
github.com/ssoelvsten/bdd-benchmark

Experimental Evaluation: Scaling beyond main memory



(a) N fixed to 15.

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.



(b) Memory fixed to 7 GiB.

Experiment run on a *Consumer grade laptop* with one quadro-core 2.6 GHz Intel i7-4720HQ processor, 8 GiB of RAM, 230 GiB of available SSD disk.

Figure: Performance on the N-Queens problem in relation to available memory.

Experimental Evaluation: Comparison to conventional BDD packages

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.

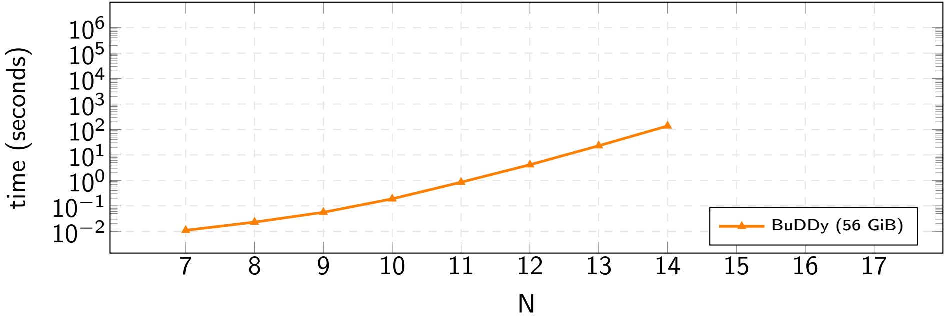


Figure: Average running times for the N-Queens problem.

Experimental Evaluation: Comparison to conventional BDD packages

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.

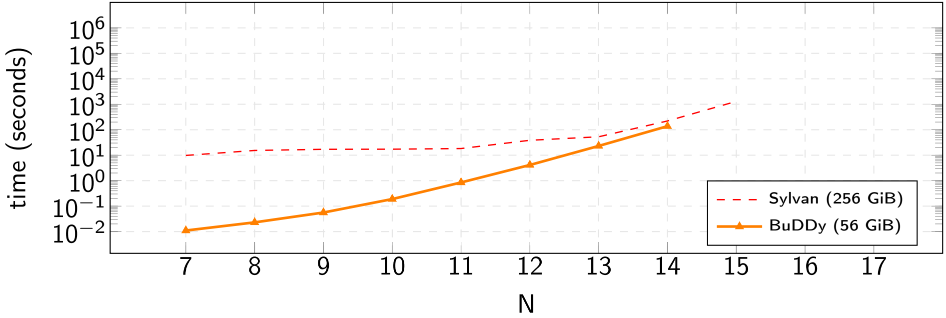


Figure: Average running times for the *N*-Queens problem.

Experimental Evaluation: Comparison to conventional BDD packages

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.

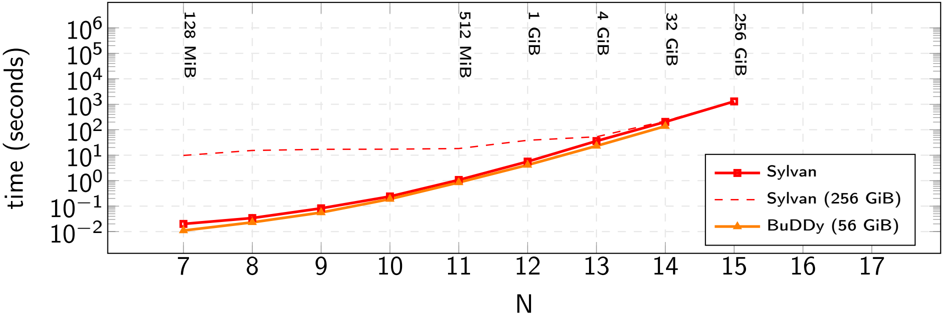


Figure: Average running times for the N-Queens problem.

Experimental Evaluation: Comparison to conventional BDD packages

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.

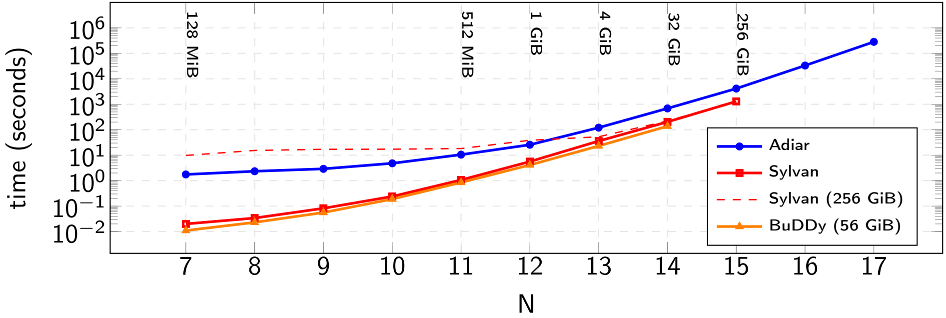


Figure: Average running times for the *N*-Queens problem.

Experimental Evaluation: Comparison to conventional BDD packages

Experiment run on *Grendel-S cluster node* with two 48-core 3.0 GHz Intel Xeon Gold 6248R processors, 384 GiB of RAM, and 3.5 TiB of available SSD disk.

Memory (GiB)	8	32	64	256
Adiar (s)	4198.3	4163.2	4151.2	4134.3
Sylvan (s)	–	–	1904.4	1295.6
Factor	–	–	2.2	3.2

Table: 15-Queens performance given different amounts of memory.

Section 4

Conclusions and Future Work

Conclusions and Future Work

We provide I/O efficient algorithms that scale BDD manipulation beyond main memory.

Algorithm		Depth-first	Time-forwarded
Reduce		$O(N)$	$O(\text{sort}(N))$
BDD Manipulation			
Apply	$f \odot g$	$O(N_f \cdot N_g)$	$O(\text{sort}(N_f \cdot N_g))$
If-Then-Else	$f \ ? \ g \ : \ h$	$O(N_f \cdot N_g \cdot N_h)$	$O(\text{sort}(N_f \cdot N_g \cdot N_h))$
Restrict	$f\vert_{x_i=v}$	$O(N)$	$O(\text{sort}(N))$
Negation	$\neg f$	$O(1)$	$O(1)$
Quantification	$\exists/\forall v : f\vert_{x_i=v}$	$O(N^2)$	$O(\text{sort}(N^2))$
Counting			
Count Paths	$\#$paths in f to $\top$	$O(N)$	$O(\text{sort}(N))$
Count SAT	$\#x : f(x)$	$O(N)$	$O(\text{sort}(N))$
Other			
Equality	$f \equiv g$	$O(1)$	$O(\text{sort}(N^2))$
Evaluate	$f(x)$	$O(L)$	$O(N/B)$
Min/Max SAT	$\min / \max\{x \mid f(x)\}$	$O(L)$	$O(N/B)$

Future Work: Usable in fixpoint algorithms used in symbolic model checking.

- Address the $O(\text{sort}(N^2))$ non-constant equality checking.
- Develop external memory multi-variable quantification and *Relational Product*.

Lars Arge. "External Geometric Data Structures". In: *Computing and Combinatorics: 10th Annual International Conference* (Aug. 2004).

Lars Arge. "The buffer tree: A new technique for optimal I/O-algorithms". In: *Algorithms and Data Structures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 334–345. isbn: 978-3-540-44747-4.

Lars Arge. "The I/O-complexity of Ordered Binary-Decision Diagram Manipulation". In: *Efficient External-Memory Data Structures and Applications (PhD Thesis)*. Aarhus Universitet, Datalogisk Institut, Denmark, Aug. 1996, pp. 123 –145.

A. Aggarwal and Jeffrey Vitter. *The input/output complexity of sorting and related problems*. Tech. rep. INRIA, Jan. 1987.

Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677 –691.

Daniel Kunkle, Vlad Slavici, and Gene Cooperman. "Parallel Disk-Based Computation for Large, Monolithic Binary Decision Diagrams". In: *PASCO'2010 - Proceedings of the 2010 International Workshop on Parallel Symbolic Computation* (Nov. 2010), pp. 63–72.

Steffan Christ Sølvsten et al. *Efficient Binary Decision Diagram Manipulation in External Memory*. 2021. arXiv: `2104.12101 [cs.DS]`.