Algorithms and Satisfiability

9. Binary Decision Diagrams

Álvaro Torralba

AALBORG UNIVERSITET

Spring 2023

Agenda

1. Introduction
2. Binary Decision Diagrams
3. Queries
4. Operations
5. Conclusions

Agenda

1. Introduction
2. Binary Decision Diagrams
3. Queries
4. Operations
5. Conclusions

How To Represent Logical Formulas?

Option 1: Logical Formula (e.g. in CNF)

$(\neg x_1 \vee x_2) \wedge x_3$

Advantage: Compact representation

Disadvantage: SAT (and other operations) is hard

How To Represent Logical Formulas?

Option 1: Logical Formula (e.g. in CNF)

$(\neg x_1 \lor x_2) \land x_3$

Advantage: Compact representation

Disadvantage: SAT (and other operations) is hard

Option 2: Truth table

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

Advantage: SAT is easy (linear in the size of the table)

Disadvantage: Table is BIG (exponentially longer than CNF formula)

How To Represent Logical Formulas?

Option 1: Logical Formula (e.g. in CNF)

$(\neg x_1 \vee x_2) \wedge x_3$

Advantage: Compact representation

Disadvantage: SAT (and other operations) is hard

Option 2: Truth table

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

Advantage: SAT is easy (linear in the size of the table)

Disadvantage: Table is BIG (exponentially longer than CNF formula)

→So, does there exist some compact representation under which SAT is easy (in the size of the representation)?

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \to \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \to \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	$\top$
0	0	1	$\perp$
0	1	0	$\top$
0	1	1	$\perp$
1	1	0	$\top$
1	1	1	$\perp$
1	0	0	$\perp$
1	0	1	$\perp$

- Notation: We use $0, 1$ for the inputs and $\perp$, $\top$ for the output

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	$\top$
0	0	1	$\bot$
0	1	0	$\top$
0	1	1	$\bot$
1	1	0	$\top$
1	1	1	$\bot$
1	0	0	$\bot$
1	0	1	$\bot$

- Notation: We use $0, 1$ for the inputs and $\bot$, $\top$ for the output
- Represents sets of assignments to the variables in a logical formula
- Represents a set of sets of elements (over k elements)

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	$\top$
0	0	1	$\perp$
0	1	0	$\top$
0	1	1	$\perp$
1	1	0	$\top$
1	1	1	$\perp$
1	0	0	$\perp$
1	0	1	$\perp$

- Notation: We use $0, 1$ for the inputs and $\perp$, $\top$ for the output
- Represents sets of assignments to the variables in a logical formula
- Represents a set of sets of elements (over k elements)
  - $\rightarrow$ Example: given k products each with a price, which sets of products can be bought with the available money?

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	$\top$
0	0	1	$\bot$
0	1	0	$\top$
0	1	1	$\bot$
1	1	0	$\top$
1	1	1	$\bot$
1	0	0	$\bot$
1	0	1	$\bot$

- Notation: We use $0, 1$ for the inputs and $\bot$, $\top$ for the output
- Represents sets of assignments to the variables in a logical formula
- Represents a set of sets of elements (over k elements)
  - $\rightarrow$ Example: given k products each with a price, which sets of products can be bought with the available money? which sets of products satisfy our requirements?

Boolean Functions

Definition

A Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$ maps k Boolean values to true or false.

x_1	x_2	x_3	value
0	0	0	$\top$
0	0	1	$\bot$
0	1	0	$\top$
0	1	1	$\bot$
1	1	0	$\top$
1	1	1	$\bot$
1	0	0	$\bot$
1	0	1	$\bot$

- Notation: We use $0, 1$ for the inputs and $\bot$, $\top$ for the output
- Represents sets of assignments to the variables in a logical formula
- Represents a set of sets of elements (over k elements)
  - $\rightarrow$ Example: given k products each with a price, which sets of products can be bought with the available money? which sets of products satisfy our requirements? what's the intersection of the previous sets?

Agenda

(Reduced Ordered) Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams (or BDDs for short): representation of logical functions that follows the following key ideas:

1. Decision Diagram: Use Directed Acyclic Graph

(Reduced Ordered) Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams (or BDDs for short): representation of logical functions that follows the following key ideas:

1. Decision Diagram: Use Directed Acyclic Graph
2. Reduced: Apply Reduction Rules

(Reduced Ordered) Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams (or BDDs for short): representation of logical functions that follows the following key ideas:

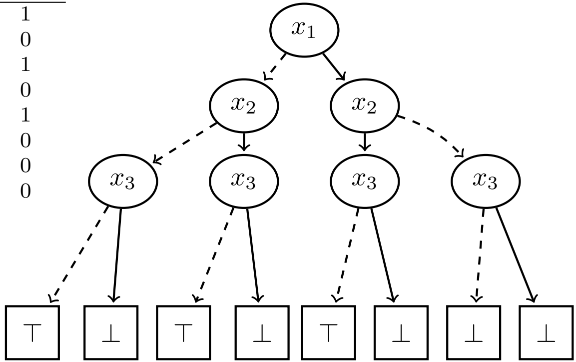
1. Decision Diagram: Use Directed Acyclic Graph
2. Reduced: Apply Reduction Rules
3. Ordered: Fixed Variable Ordering

"BDDs are one of the only really fundamental data structures that came out in the last twenty-five years" — Donald Knuth

Representing Logical Functions as a Decision Tree

Turn a truth table for the Boolean function into a decision tree.

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

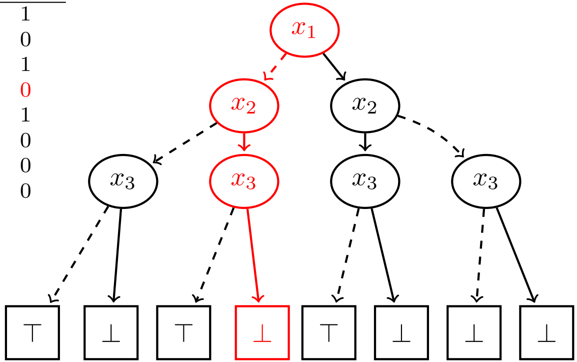


Notation: Nodes are labelled with the variable they checked (x_i), we use solid edge for $x_i = 1$ and dashed edge for $x_i = 0$.

Representing Logical Functions as a Decision Tree

Turn a truth table for the Boolean function into a decision tree.

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0



Notation: Nodes are labelled with the variable they checked (x_i), we use solid edge for $x_i = 1$ and dashed edge for $x_i = 0$.

Shannon Expansion

For any Boolean formula f and variable x, it can be written as:

$$f \equiv (\neg x \wedge f[x = 0]) \vee (x \wedge f[x = 1])$$

This is the Shannon expansion of f (originally due to G. Boole).

Shannon Expansion

For any Boolean formula f and variable x, it can be written as:

$$f \equiv (\neg x \wedge f[x = 0]) \vee (x \wedge f[x = 1])$$

This is the Shannon expansion of f (originally due to G. Boole).

Deja vu? Same principle used by DPLL!

Shannon Expansion

For any Boolean formula f and variable x, it can be written as:

$$f \equiv (\neg x \wedge f[x = 0]) \vee (x \wedge f[x = 1])$$

This is the Shannon expansion of f (originally due to G. Boole).

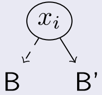
Deja vu? Same principle used by DPLL!

→Inner nodes correspond to a function represented via the Shannon expansion!

Notation

Node $A = (x_i, B, B')$ represents $f_A = (\overline{x_i} \wedge f_B) \vee (x_i \wedge f_{B'})$

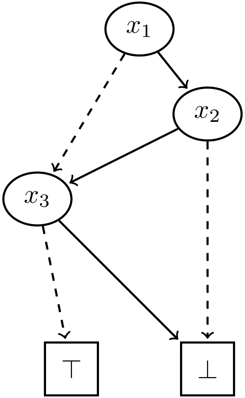
- Variable $v(A) = x_i$ (nodes are labelled with the variable they check)
- Low child $l(A) = B$ (represented by a dashed edge)
- High child $r(A) = B'$ (represented by a solid edge)



Representing Logical Functions as a Decision Diagram

Diagram: Directed Acyclic Graph (DAG)
$\rightarrow$We can share a node in different parts of the diagram

x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

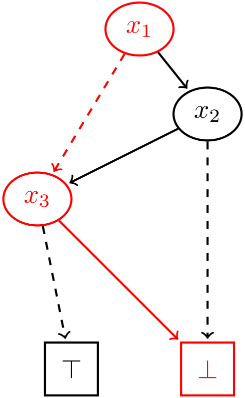


Notation: We denote $|B|$ to the number of nodes of a DAG B.

Representing Logical Functions as a Decision Diagram

Diagram: Directed Acyclic Graph (DAG)
$\rightarrow$We can share a node in different parts of the diagram

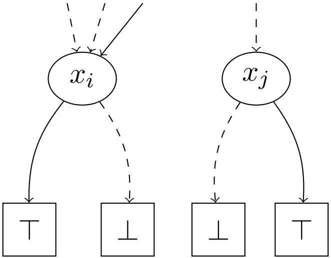
x_1	x_2	x_3	value
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0



Notation: We denote $|B|$ to the number of nodes of a DAG B.

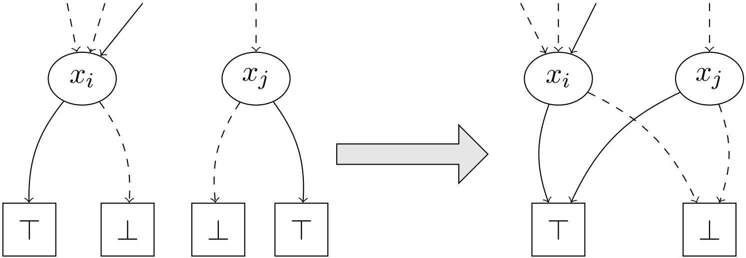
Reduction Rule #1: Eliminate Equivalent Leaves

There are only two unique leaves: $\perp$ and $\top$



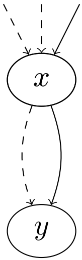
Reduction Rule #1: Eliminate Equivalent Leaves

There are only two unique leaves: $\perp$ and $\top$



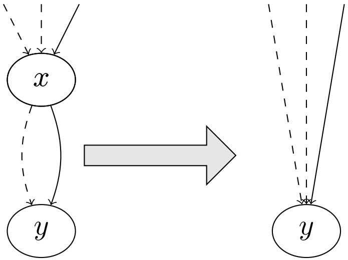
Reduction Rule #2: Eliminate Redundant Nodes

l(n) = r(n), then eliminate n and re-direct all its references to l(n)



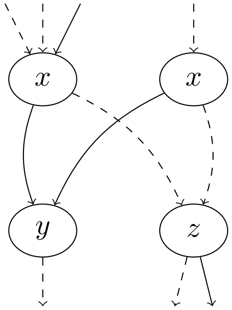
Reduction Rule #2: Eliminate Redundant Nodes

l(n) = r(n), then eliminate n and re-direct all its references to l(n)



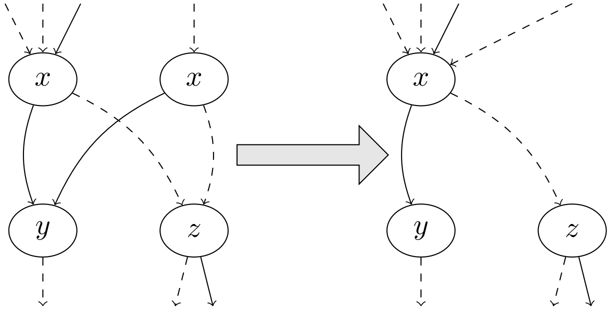
Reduction Rule #3: Eliminate Equivalent Inner Nodes

Two inner nodes n, n' are equivalent if and only if $v(n) = v(n')$, $l(n) = l(n')$, and $r(n) = r(n')$

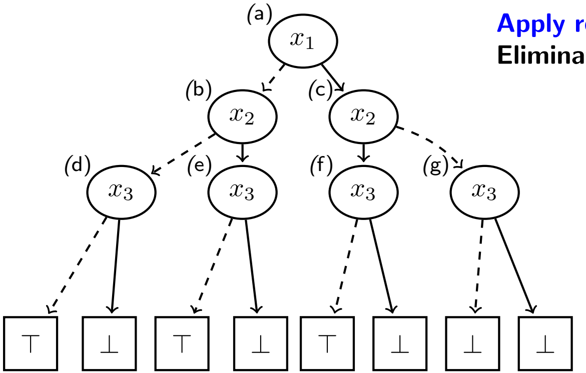


Reduction Rule #3: Eliminate Equivalent Inner Nodes

Two inner nodes n, n' are equivalent if and only if $v(n) = v(n')$, $l(n) = l(n')$, and $r(n) = r(n')$

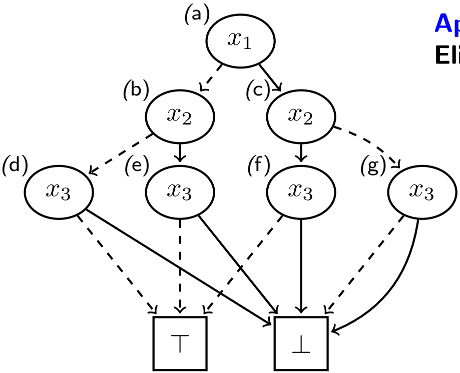


Reduction Rules Example



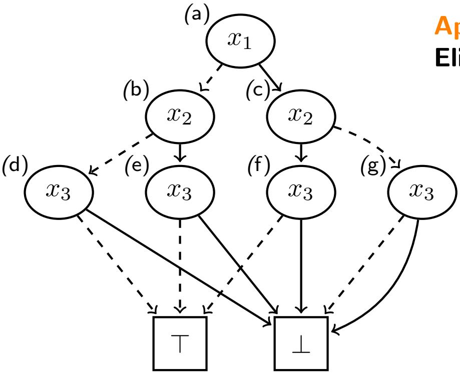
Apply reduction rule 1:
Eliminate equivalent leaves

Reduction Rules Example



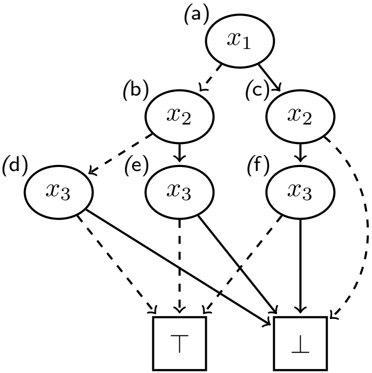
Apply reduction rule 1:
Eliminate equivalent leaves

Reduction Rules Example



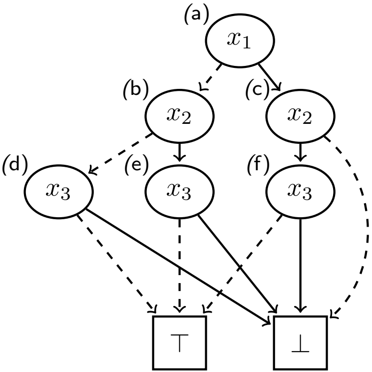
Apply reduction rule 2:
Eliminate redundant nodes

Reduction Rules Example



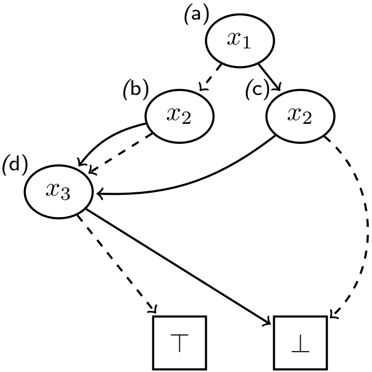
Apply reduction rule 2:
Eliminate redundant nodes

Reduction Rules Example



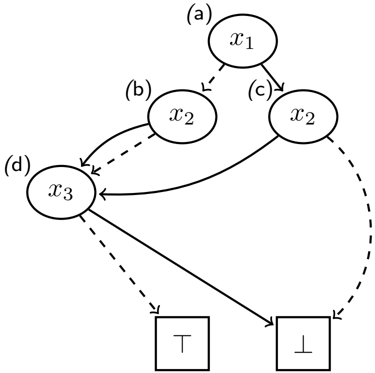
Apply reduction rule 3:
Eliminate equivalent inner nodes

Reduction Rules Example



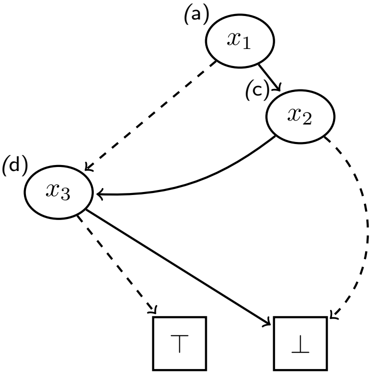
Apply reduction rule 3:
Eliminate equivalent inner nodes

Reduction Rules Example



Apply reduction rule 2:
Eliminate redundant nodes

Reduction Rules Example



Apply reduction rule 2:
Eliminate redundant nodes

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**

    // Eliminate equivalent leaves

    return UniqueTerminal$(A.value)$

$n_l = \textit{reduce}(l(A))$

$n_r = \textit{reduce}(r(A))$

if $n_l == n_r$ **then**

    // Eliminate redundant nodes

    result $= n_l$

else

    // Eliminate equivalent nodes

    result $=$ NodeUnique $(v(A), n_l, n_r)$

return result

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**
  `// Eliminate equivalent leaves`
  return UniqueTerminal$(A.value)$

$n_l = \mathit{reduce}(l(A))$
$n_r = \mathit{reduce}(r(A))$
if $n_l == n_r$ **then**
  `// Eliminate redundant nodes`
  result $= n_l$
else
  `// Eliminate equivalent nodes`
  result $=$ NodeUnique $(v(A), n_l, n_r)$

return result

NodeUnique(v, n_l, n_r):

if (v, n_l, n_r) *not in GlobalCache* **then**
  GlobalCache$[(v, n_l, n_r)] =$ new Node (v, n_l, n_r)
return GlobalCache$[(v, n_l, n_r)]$

- GlobalCache is a single cache for all BDDs and operations! →Ensures the Equivalence reduction rule

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**
  // Eliminate equivalent leaves
  return UniqueTerminal$(A.value)$
if *A in Cache* **then**
  return Cache[A]
$n_l = \mathit{reduce}(l(A))$
$n_r = \mathit{reduce}(r(A))$
if $n_l == n_r$ **then**
  // Eliminate redundant nodes
  result $= n_l$
else
  // Eliminate equivalent nodes
  result $=$ NodeUnique $(v(A), n_l, n_r)$
Cache[A] $=$ result
return result

NodeUnique(v, n_l, n_r):

if (v, n_l, n_r) *not in GlobalCache* **then**
  GlobalCache$[(v, n_l, n_r)] =$ new Node (v, n_l, n_r)
return GlobalCache$[(v, n_l, n_r)]$

- GlobalCache is a single cache for all BDDs and operations! →Ensures the Equivalence reduction rule

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**
    `// Eliminate equivalent leaves`
    return UniqueTerminal$(A.value)$
if *A in Cache* **then**
    return Cache[A]
$n_l = \mathit{reduce}(l(A))$
$n_r = \mathit{reduce}(r(A))$
if $n_l == n_r$ **then**
    `// Eliminate redundant nodes`
    result $= n_l$
else
    `// Eliminate equivalent nodes`
    result $=$ NodeUnique $(v(A), n_l, n_r)$
Cache[A] $=$ result
return result

NodeUnique(v, n_l, n_r):

if (v, n_l, n_r) *not in GlobalCache* **then**
    GlobalCache$[(v, n_l, n_r)] =$ new Node (v, n_l, n_r)
return GlobalCache$[(v, n_l, n_r)]$

- GlobalCache is a single cache for all BDDs and operations! →Ensures the Equivalence reduction rule

Question: How many calls to reduce(A) we perform?

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**
  // Eliminate equivalent leaves
  return UniqueTerminal($A.value$)
if *A in Cache* **then**
  return Cache[A]
$n_l = reduce(l(A))$
$n_r = reduce(r(A))$
if $n_l == n_r$ **then**
  // Eliminate redundant nodes
  result $= n_l$
else
  // Eliminate equivalent nodes
  result = NodeUnique $(v(A), n_l, n_r)$
Cache[A] = result
return result

NodeUnique(v, n_l, n_r):

if (v, n_l, n_r) *not in GlobalCache* **then**
  GlobalCache$[(v, n_l, n_r)]$ = new Node (v, n_l, n_r)
return GlobalCache$[(v, n_l, n_r)]$

- GlobalCache is a single cache for all BDDs and operations! $\rightarrow$Ensures the Equivalence reduction rule

Question: How many calls to reduce(A) we perform? $\mathcal{O}(|A|)$: due to the cache, we will never do recursive calls over the same node twice

The Reduce Operation: Pseudocode

reduce(A):

if *A is terminal* **then**
  `// Eliminate equivalent leaves`
  return UniqueTerminal$(A.value)$
if *A in Cache* **then**
  return Cache[A]
$n_l = \mathit{reduce}(l(A))$
$n_r = \mathit{reduce}(r(A))$
if $n_l == n_r$ **then**
  `// Eliminate redundant nodes`
  result $= n_l$
else
  `// Eliminate equivalent nodes`
  result $=$ NodeUnique $(v(A), n_l, n_r)$
Cache[A] $=$ result
return result

NodeUnique(v, n_l, n_r):

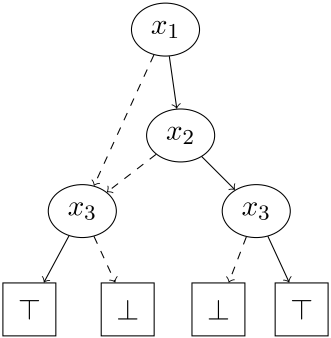
if (v, n_l, n_r) *not in GlobalCache* **then**
  GlobalCache$[(v, n_l, n_r)]$ $=$ new Node (v, n_l, n_r)
return GlobalCache$[(v, n_l, n_r)]$

- GlobalCache is a single cache for all BDDs and operations! →Ensures the Equivalence reduction rule

Question: How many calls to reduce(A) we perform? $\mathcal{O}(|A|)$: due to the cache, we will never do recursive calls over the same node twice

Overall time complexity: $\mathcal{O}(|A| \log(|A|))$

Questionnaire



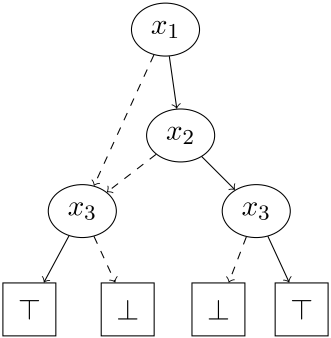
Question!

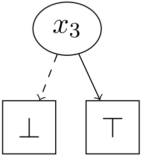
How many nodes would have the resulting BDD after applying Reduce?

(A): 3 (B): 4

(C): 5 (D): 6

Questionnaire





Question!

How many nodes would have the resulting BDD after applying Reduce?

(A): 3 (B): 4

(C): 5 (D): 6

Fixed Variable Ordering

Decide arbitrary total order on variables $p_1 < p_2 < \cdots < p_n$. Always decompose a function based on the first variable.

Fixed Variable Ordering

Decide arbitrary total order on variables $p_1 < p_2 < \cdots < p_n$. Always decompose a function based on the first variable.

Definition (Canonicity). *We say that a representation R of a logical formula φ is canonical if there is a unique representation:*

$$\varphi \equiv \varphi' \implies R(\varphi) = R(\varphi')$$

Fixed Variable Ordering

Decide arbitrary total order on variables $p_1 < p_2 < \cdots < p_n$. Always decompose a function based on the first variable.

Definition (Canonicity). *We say that a representation R of a logical formula φ is canonical if there is a unique representation:*

$$\varphi \equiv \varphi' \implies R(\varphi) = R(\varphi')$$

Theorem (ROBDDs are a canonical representation). For a function $f : \{0,1\}^n \to \{0,1\}$ there is exactly one ROBDD u with ordering $p_1 < \cdots < p_n$ such that u represents $f(p_1, \ldots, p_n)$.

Fixed Variable Ordering

Decide arbitrary total order on variables $p_1 < p_2 < \cdots < p_n$. Always decompose a function based on the first variable.

Definition (Canonicity). *We say that a representation R of a logical formula φ is canonical if there is a unique representation:*

$$\varphi \equiv \varphi' \implies R(\varphi) = R(\varphi')$$

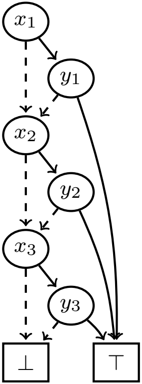
Theorem (ROBDDs are a canonical representation). For a function $f : \{0,1\}^n \to \{0,1\}$ there is exactly one ROBDD u with ordering $p_1 < \cdots < p_n$ such that u represents $f(p_1, \ldots, p_n)$.

Proof sketch. By induction (base case is trivial for terminal nodes). Assume ROBDDs for functions $f(p_i, \ldots, p_n)$ are canonical, then show that ROBDDs for $f(p_{i-1}, \ldots, p_n)$ are canonical as well.

Importance of Variable Ordering

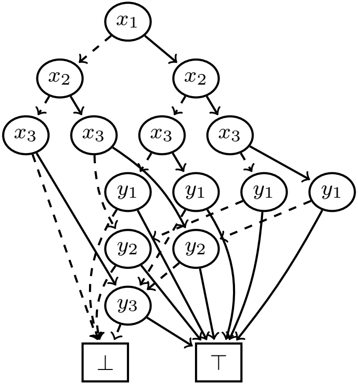
Choice of variable ordering can cause the representation of a function to be exponential or polynomial in the number of variables.

Example: $f(x_1, \dots, x_n, y_1, \dots, y_n) = (x_1 \wedge y_1) \vee \dots \vee (x_n \wedge y_n)$ $(n = 3)$.



Polynomial size:

$x_1, y_1, x_2, y_2, x_3, y_3$



Exponential size:

$x_1, x_2, x_3, y_1, y_2, y_3$

Variable Ordering Heuristics

- Static Variable Ordering:
  - Put “related” variables as close as possible in the variable ordering

Variable Ordering Heuristics

- Static Variable Ordering:
  - Put "related" variables as close as possible in the variable ordering
- Dynamic Variable Ordering:
  - Find the best ordering for a given BDD (or set of BDDs)
  - Optimization problem that is NP-complete
  - But we can just use some approximation

Agenda

Queries

- Tautology: Is $\varphi = \top$?

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

$\rightarrow$Constant time: just check if $n_\varphi \neq \bot$

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

$\rightarrow$Constant time: just check if $n_\varphi \neq \bot$

- Equivalence ($\varphi \equiv \psi$?): Is $M(\varphi) = M(\psi)$?

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

$\rightarrow$Constant time: just check if $n_\varphi \neq \bot$

- Equivalence ($\varphi \equiv \psi$?): Is $M(\varphi) = M(\psi)$?

$\rightarrow$Constant time: just check if $n_\varphi = n_\psi$

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

$\rightarrow$Constant time: just check if $n_\varphi \neq \bot$

- Equivalence ($\varphi \equiv \psi$?): Is $M(\varphi) = M(\psi)$?

$\rightarrow$Constant time: just check if $n_\varphi = n_\psi$

- Model counting: Number of satisfying assignments: $|M(\varphi)|$

Queries

- Tautology: Is $\varphi = \top$?

$\rightarrow$Constant time: just check if $n_\varphi = \top$

- SAT: Is $M(\varphi) \neq \emptyset$?

$\rightarrow$Constant time: just check if $n_\varphi \neq \bot$

- Equivalence ($\varphi \equiv \psi$?): Is $M(\varphi) = M(\psi)$?

$\rightarrow$Constant time: just check if $n_\varphi = n_\psi$

- Model counting: Number of satisfying assignments: $|M(\varphi)|$

$\rightarrow$Linear time: dynamic programming algorithm

Model Counting: Example

$MC(f)$: number of assignments over $x_1, \ldots, x_n$ that make f true.

Model Counting: Example

$MC(f)$: number of assignments over $x_1, \ldots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

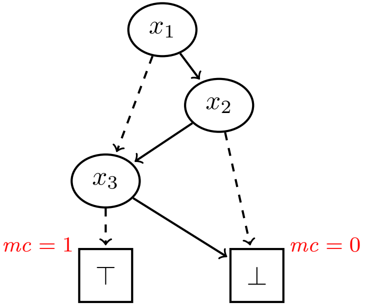
- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then

Model Counting: Example

$MC(f)$: number of assignments over $x_1, \ldots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then

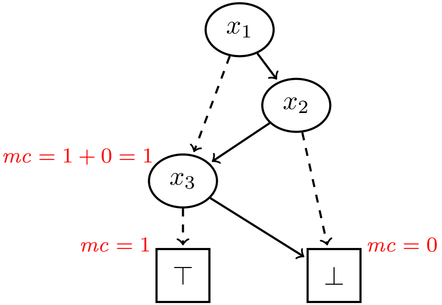


Model Counting: Example

$MC(f)$: number of assignments over $x_1, \ldots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then

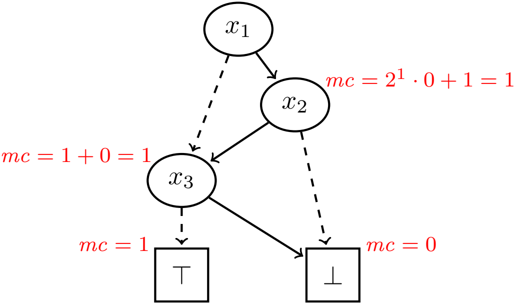


Model Counting: Example

$MC(f)$: number of assignments over $x_1, \dots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then

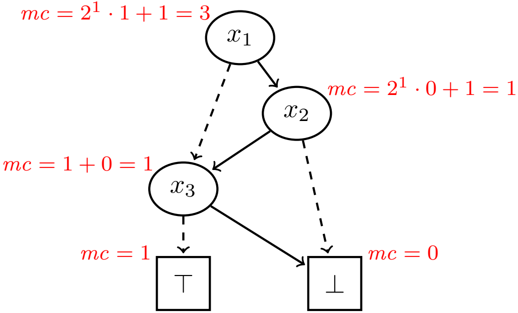


Model Counting: Example

$MC(f)$: number of assignments over $x_1, \dots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then

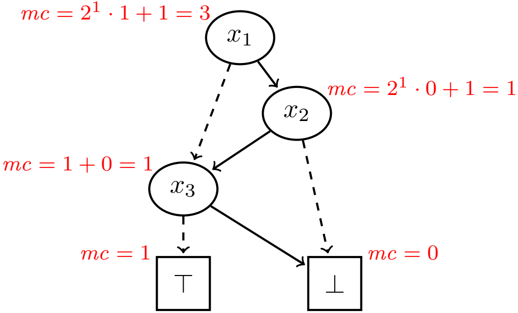


Model Counting: Example

$MC(f)$: number of assignments over $x_1, \dots, x_n$ that make f true.

$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then



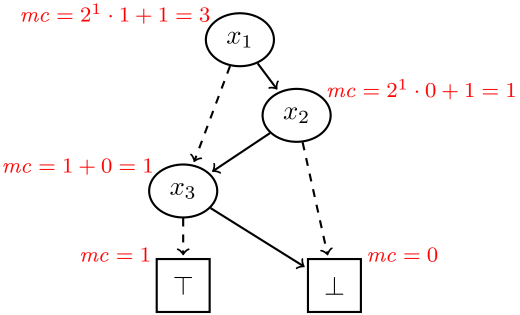
$MC(n_1) =$

Model Counting: Example

$MC(f)$: number of assignments over $x_1, \ldots, x_n$ that make f true.

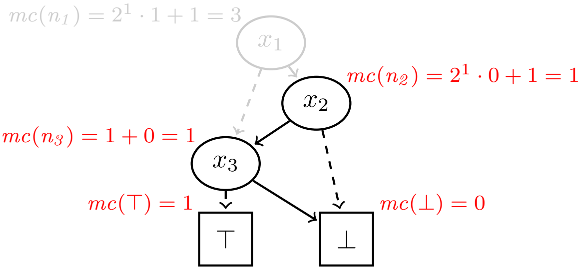
$MC(f) = 2^{v(f)} mc(f)$ where:

- Leaf nodes: $mc(\top) = 1, mc(\bot) = 0$ and define $v(\bot) = v(\top) = x_{n+1}$
- Inner nodes: $mc(f) = 2^{j-i-1} \cdot mc(f_0) + 2^{k-i-1} \cdot mc(f_1)$ where $f = (x_i, f_0, f_1)$, $v(f_0) = x_j$, $v(f_1) = x_k$, then



$MC(n_1) = 3$

Questionnaire



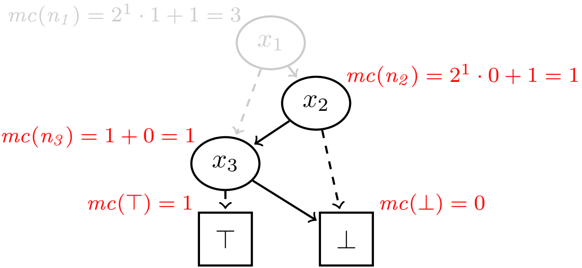
Question!

How many models does n_2 have over $x_1, \dots, x_3$? (MC (n_2)))

(A): 1 (B): 2

(C): 3 (D): 4

Questionnaire



Question!

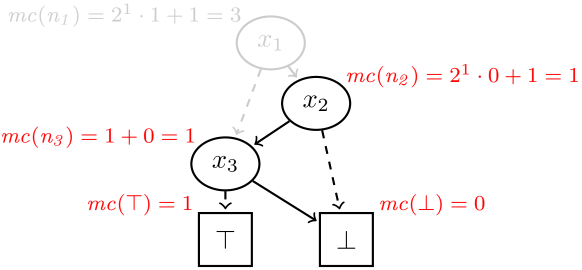
How many models does n_2 have over $x_1, \dots, x_3$? (MC (n_2)))

(A): 1 (B): 2

(C): 3 (D): 4

2 assignments (010 and 110), so MC $(n_2) = 2$.
Similarly, $MC(n_3) =$

Questionnaire



Question!

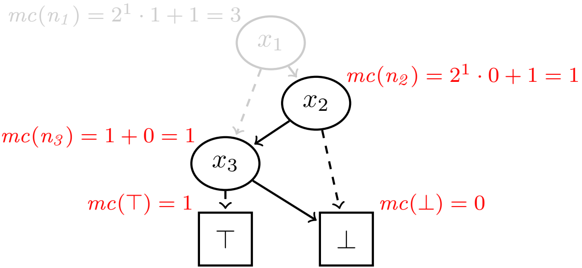
How many models does n_2 have over $x_1, \dots, x_3$? (MC (n_2)))

(A): 1 (B): 2

(C): 3 (D): 4

2 assignments (010 and 110), so MC $(n_2) = 2$.
Similarly, $MC(n_3) = 2^2 \cdot 1 = 4$, $MC(\top) =$

Questionnaire



Question!

How many models does n_2 have over $x_1, \dots, x_3$? (MC (n_2)))

(A): 1 (B): 2

(C): 3 (D): 4

2 assignments (010 and 110), so MC $(n_2) = 2$.

Similarly, $MC(n_3) = 2^2 \cdot 1 = 4$, $MC(\top) = 2^3 \cdot 1 = 8$

$\rightarrow$We must specify over how many variables we perform model counting

Agenda

BDD Operations

- Reduce (f)
  $\rightarrow$Reduces the canonical form of a function f represented as a non-reduced BDD

BDD Operations

- Reduce (f)
  $\rightarrow$Reduces the canonical form of a function f represented as a non-reduced BDD
- Apply (f_1, f_2)
  $\rightarrow$Generic implementation for binary operations $(\vee, \wedge, \dots)$

BDD Operations

- Reduce (f)
  →Reduces the canonical form of a function f represented as a non-reduced BDD
- Apply (f_1, f_2)
  →Generic implementation for binary operations $(\vee, \wedge, \dots)$
- If-Then-Else (ITE) (f_1, f_2, f_3)
  →Generic implementation for ternary operations

BDD Operations

- Reduce (f)
  $\rightarrow$Reduces the canonical form of a function f represented as a non-reduced BDD
- Apply (f_1, f_2)
  $\rightarrow$Generic implementation for binary operations $(\vee, \wedge, \dots)$
- If-Then-Else (ITE) (f_1, f_2, f_3)
  $\rightarrow$Generic implementation for ternary operations
- Exists (X, f)
  $\rightarrow$Forgets variables X from the BDD

BDD Operations

- Reduce (f)
  $\rightarrow$Reduces the canonical form of a function f represented as a non-reduced BDD
- Apply (f_1, f_2)
  $\rightarrow$Generic implementation for binary operations $(\vee, \wedge, \dots)$
- If-Then-Else (ITE) (f_1, f_2, f_3)
  $\rightarrow$Generic implementation for ternary operations
- Exists (X, f)
  $\rightarrow$Forgets variables X from the BDD

Keys:

- Recursively iterate over the BDD
- Dynamic programming: Use cache to store intermediate results. Never re-do the same work twice!

But. . . how to obtain a BDD in the first place?

BDDs are always reduced

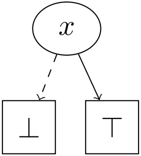
But. . . how to obtain a BDD in the first place?

BDDs are always reduced (the reduce function is never actually used!)

But. . . how to obtain a BDD in the first place?

BDDs are always reduced (the reduce function is never actually used!)

You can easily obtain the BDDs that represent simple functions:



And then, use operations on them to construct more complex functions!

$$(\neg x_1 \lor x_2) \land x_3$$

The Apply Operation

Given compatible OBDDs B_f and B_g that represent formulas f and g, apply($\Box, B_f, B_g$) computes an OBDD representing $f \Box g$.

- Compatible: operands must use the same variable ordering
- $\Box$ is some binary operation on Boolean formulas (e.g. $\vee, \wedge, \oplus$)
- Unary operations can be handled too.
  for example, negation: $\neg x = x \oplus 1$

Using the Shannon expansion, $f \Box g$ can be expanded as:

$$f \Box g \equiv (\neg x \wedge (f[x=0] \Box g[x=0])) \vee (x \wedge (f[x=1] \Box g[x=1]))$$

The Apply Operation: Pseudocode

apply$(\Box, A, B)$:

if *A is terminal and B is terminal* **then**
  return UniqueTerminal$(A.value \Box B.value)$

if $v(A) == v(B)$ **then**
  $v = v(A)$
  $n_l = apply(\Box, l(A), l(B))$
  $n_r = apply(\Box, r(A), r(B))$
else if *B is terminal or* $v(B) > v(A)$ **then**
  $v = v(A)$
  $n_l = apply(\Box, l(A), B)$
  $n_r = apply(\Box, r(A), B)$
else
  $v = v(B)$
  $n_l = apply(\Box, A, l(B))$
  $n_r = apply(\Box, A, r(B))$

result $=$ NodeUnique (v, n_l, n_r)

return result

The Apply Operation: Pseudocode

apply($\Box, A, B$):

if *A is terminal and B is terminal* **then**
    return UniqueTerminal($A.value \Box B.value$)
if *(A,B) in Cache* **then**
    return Cache[(A, B)]
if $v(A) == v(B)$ **then**
    $v = v(A)$
    $n_l = apply(\Box, l(A), l(B))$
    $n_r = apply(\Box, r(A), r(B))$
else if *B is terminal or* $v(B) > v(A)$ **then**
    $v = v(A)$
    $n_l = apply(\Box, l(A), B)$
    $n_r = apply(\Box, r(A), B)$
else
    $v = v(B)$
    $n_l = apply(\Box, A, l(B))$
    $n_r = apply(\Box, A, r(B))$
result = NodeUnique (v, n_l, n_r)
Cache[(A,B)] = result
return result

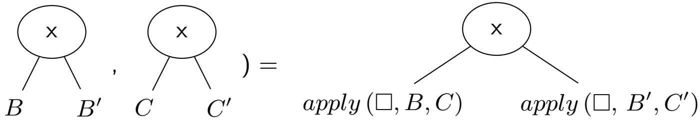
The Apply Operation: Cases

1. apply $(\square, \boxed{\text{u}}, \boxed{\text{v}}) = \boxed{u \square v}$

The Apply Operation: Cases

1. apply $(\Box, \boxed{u}, \boxed{v}) = \boxed{u \Box v}$

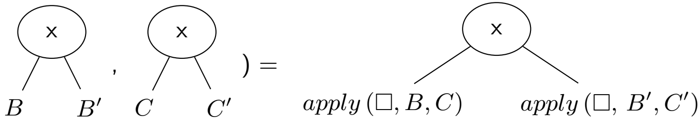
2. apply $(\Box,$



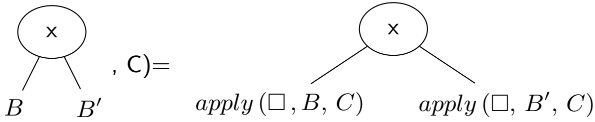
The Apply Operation: Cases

1. apply $(\Box, \boxed{\text{u}}, \boxed{\text{v}}) = \boxed{u \Box v}$

2. apply $(\Box,$



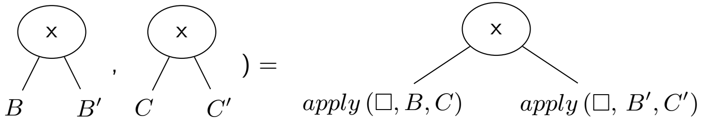
3. apply $(\Box,$

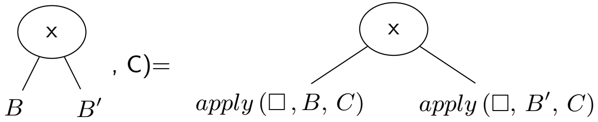


when C is terminal node, or non-terminal with $v(C) > x$

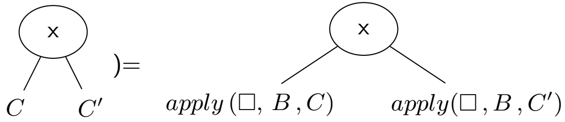
The Apply Operation: Cases

1. apply $(\square, \boxed{u}, \boxed{v}) = \boxed{u \square v}$

2. apply $(\square,$ 

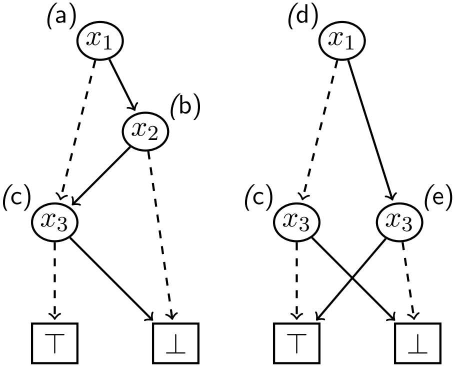
3. apply $(\square,$ 

   when C is terminal node, or non-terminal with $v(C) > x$

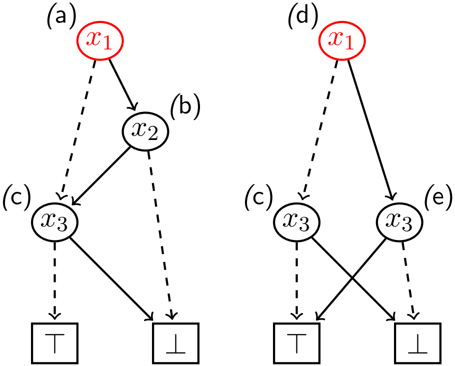
4. apply $(\square,\ B,$ 

   when B is terminal node, or non-terminal with $v(B) > x$

The Apply Operation: Example (Conjunction)

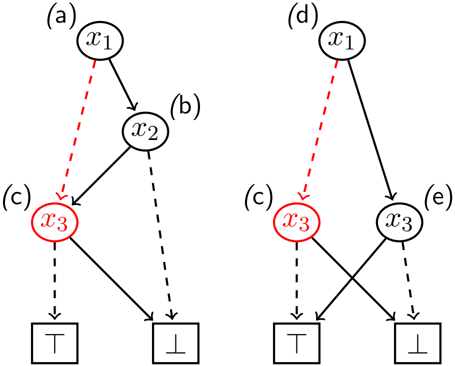


The Apply Operation: Example (Conjunction)



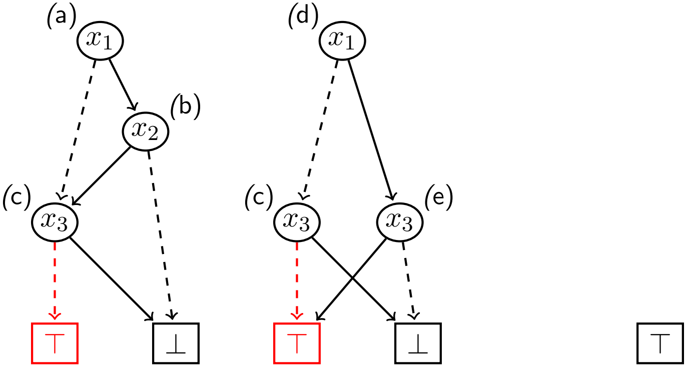
1. $(a, d) \quad \mapsto (x_1, \_\ , \_\quad)$

The Apply Operation: Example (Conjunction)



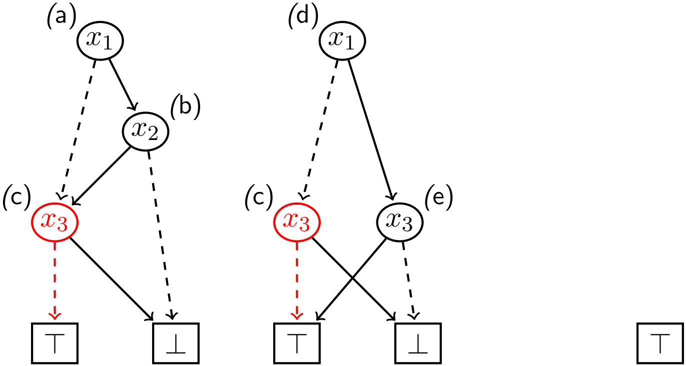
1. $(a, d) \mapsto (x_1, \_ , \_)$
2. $(c, c) \mapsto (x_3, \_ , \_)$

The Apply Operation: Example (Conjunction)



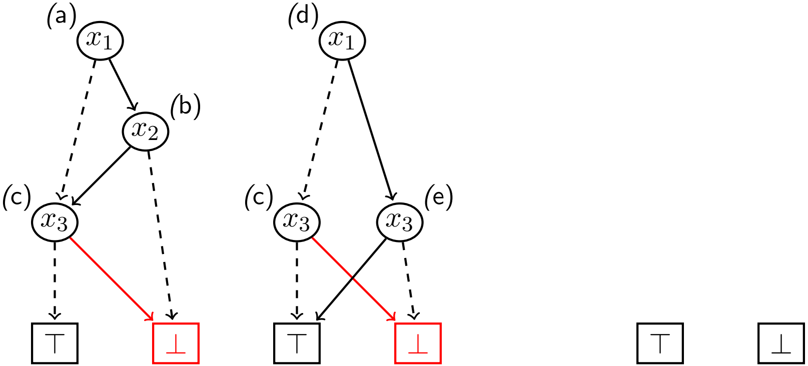
1. $(a, d) \mapsto (x_1, \_\ , \_\ )$
2. $(c, c) \mapsto (x_3, \_\ , \_\ )$
3. $(\top, \top) \mapsto \top$

The Apply Operation: Example (Conjunction)



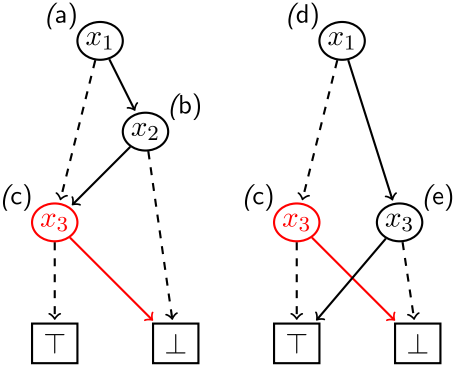
1. $(a, d) \mapsto (x_1, \_ , \_)$
2. $(c, c) \mapsto (x_3, \top, \_)$
3. $(\top, \top) \mapsto \top$

The Apply Operation: Example (Conjunction)



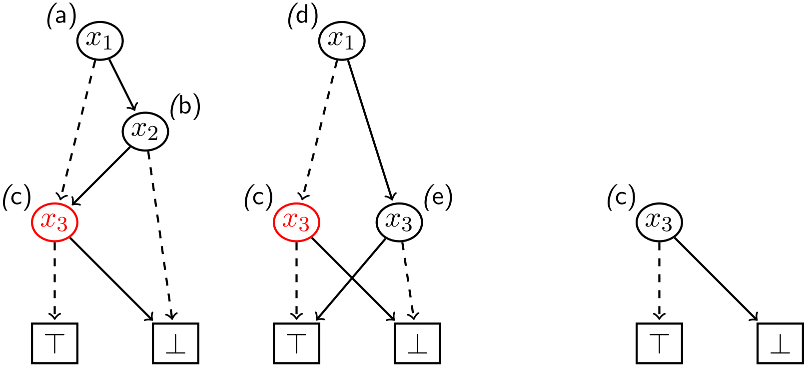
1. $(a, d) \mapsto (x_1, \_\,, \_\ )$
2. $(c, c) \mapsto (x_3, \top, \_\ )$
3. $(\top, \top) \mapsto \top$
4. $(\perp, \perp) \mapsto \perp$

The Apply Operation: Example (Conjunction)



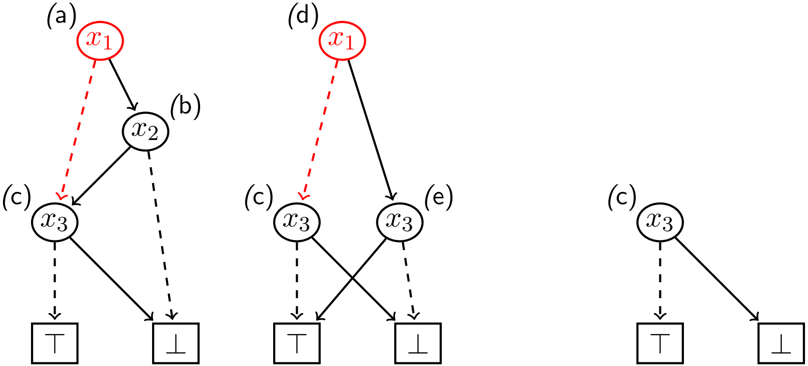
1. $(a, d) \mapsto (x_1, \_ , \_)$
2. $(c, c) \mapsto (x_3, \top, \bot)$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



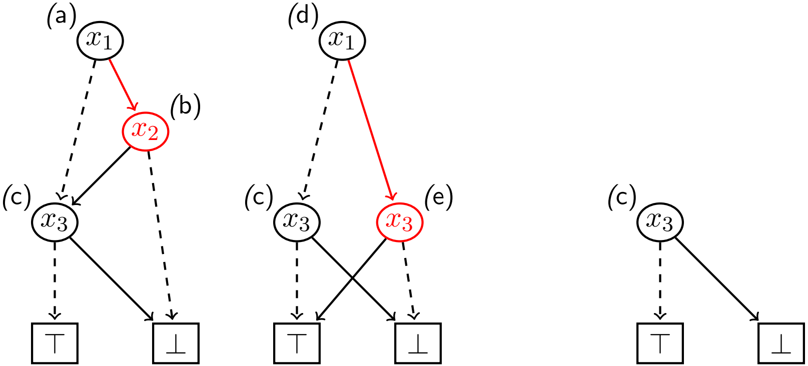
1. $(a, d) \mapsto (x_1, \_\ , \_\ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



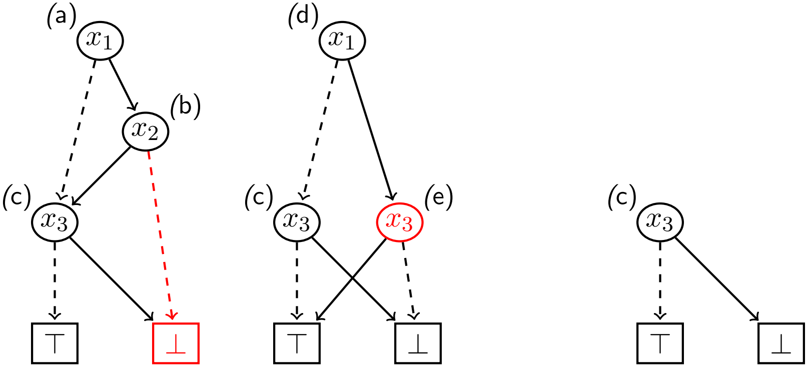
1. $(a, d) \mapsto (x_1,\ c,\ \_\ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



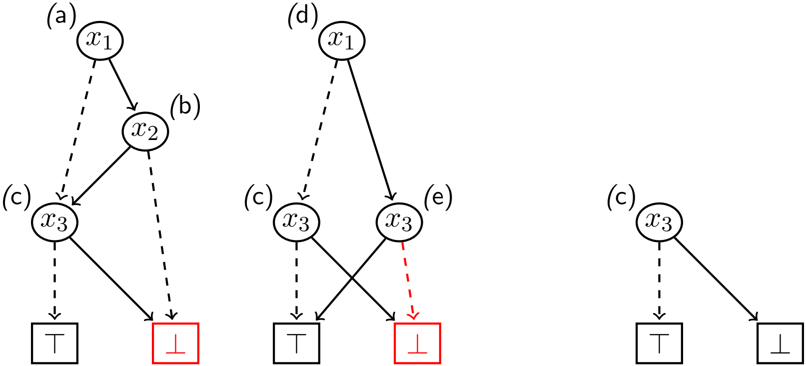
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2, \_\ \ , \_\ \ )$

The Apply Operation: Example (Conjunction)



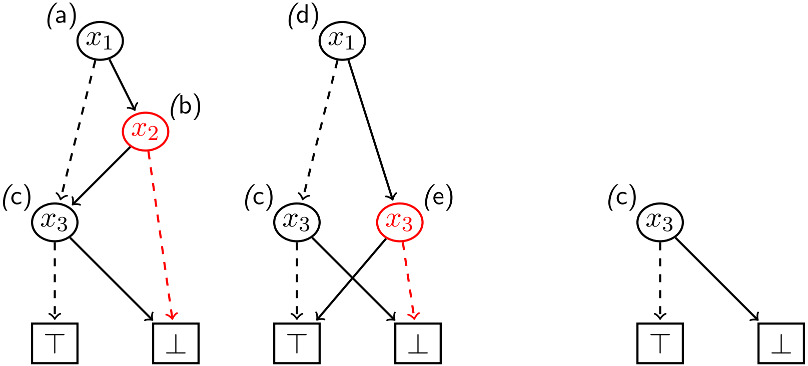
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2, \_\ \ , \_\ \ )$
6. $(\bot, e) \mapsto$

The Apply Operation: Example (Conjunction)



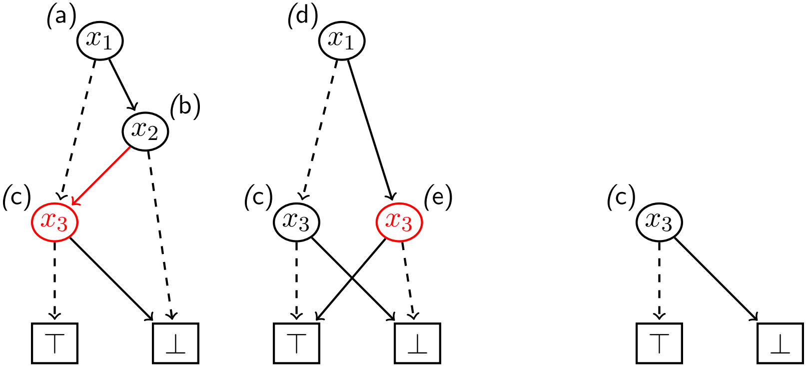
1. $(a, d) \mapsto (x_1, \ c, \_ \quad)$
2. $(c, c) \mapsto (x_3, \ \top, \ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2, \quad , \_ \quad)$
6. $(\bot, e) \mapsto$
7. $(\bot, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



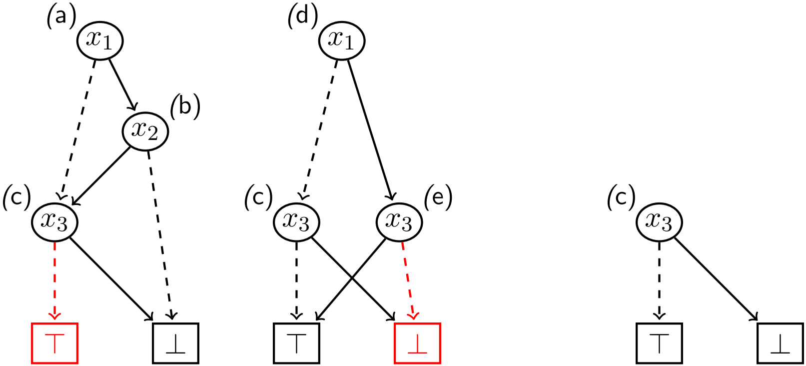
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



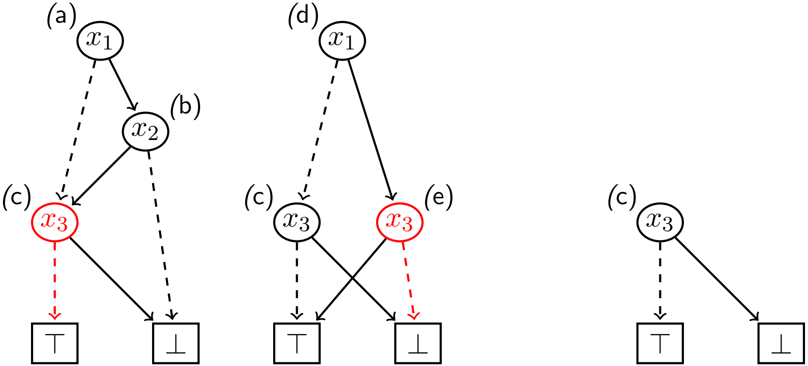
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3, \_\ \ , \_\ \ )$

The Apply Operation: Example (Conjunction)



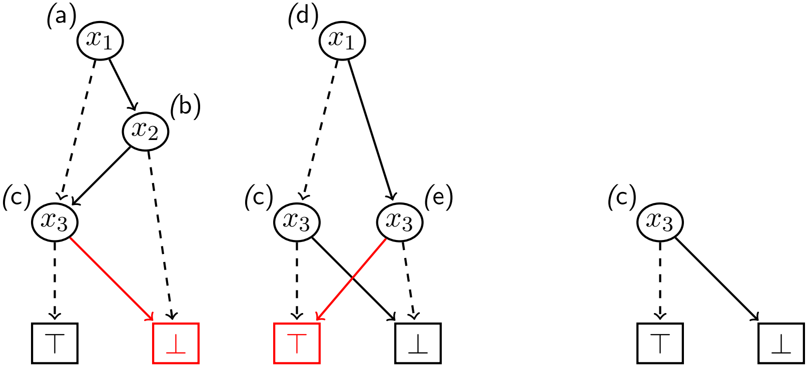
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3, \_\ \ , \_\ \ )$
9. $(\top, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



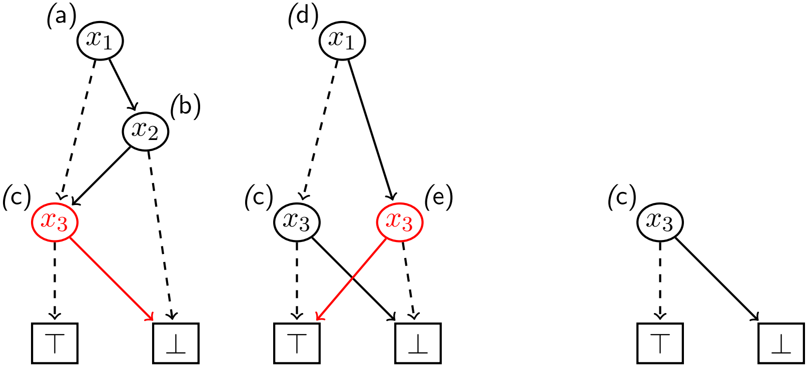
1. $(a, d) \mapsto (x_1,\ c,\ \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot,\ \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \_\ \ )$
9. $(\top, \bot) \mapsto \bot$

The Apply Operation: Example (Conjunction)



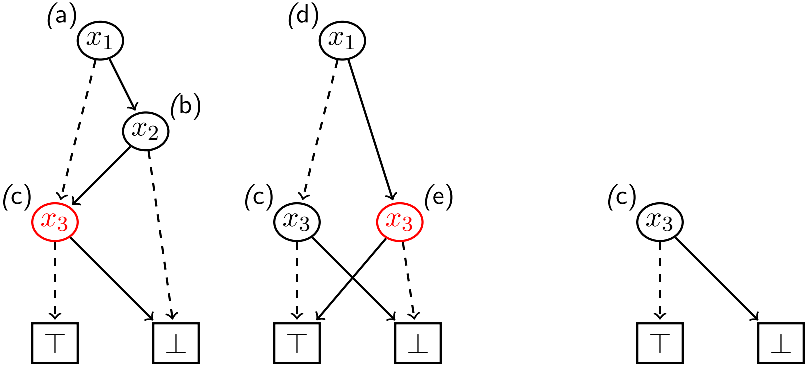
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot, \_\ \ )$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



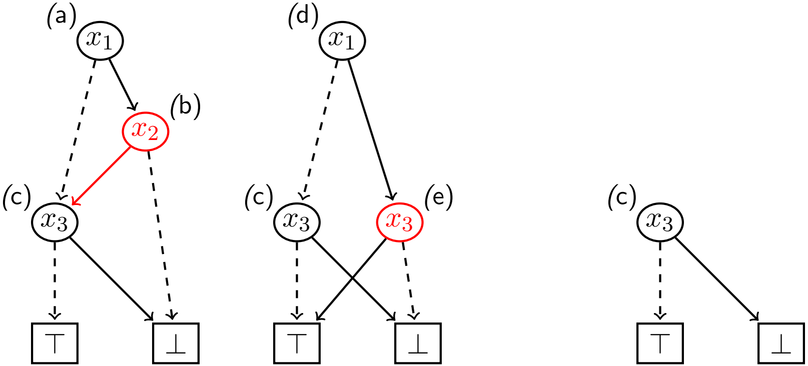
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot)$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



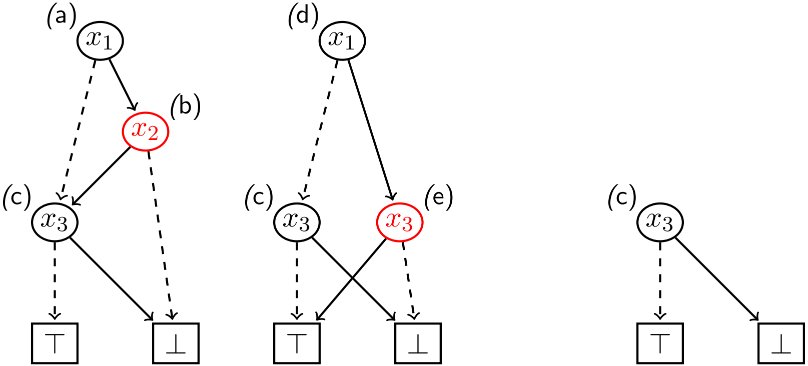
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot, \_\ \ )$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot) \mapsto \bot$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



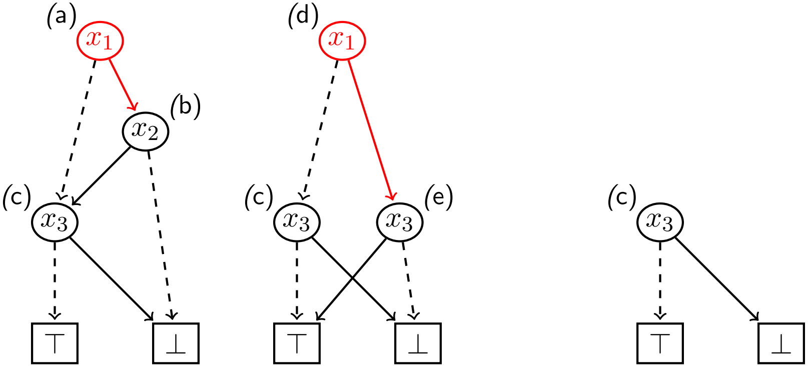
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot,\ \bot)$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot) \mapsto \bot$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



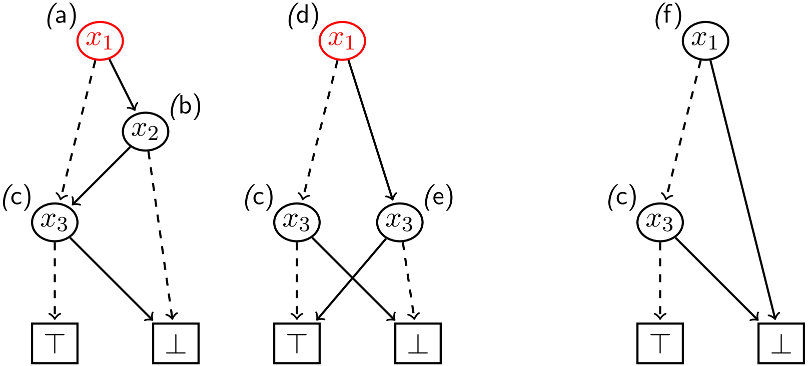
1. $(a, d) \mapsto (x_1,\ c, \_\ \ )$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot,\ \bot) \mapsto \bot$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot) \mapsto \bot$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



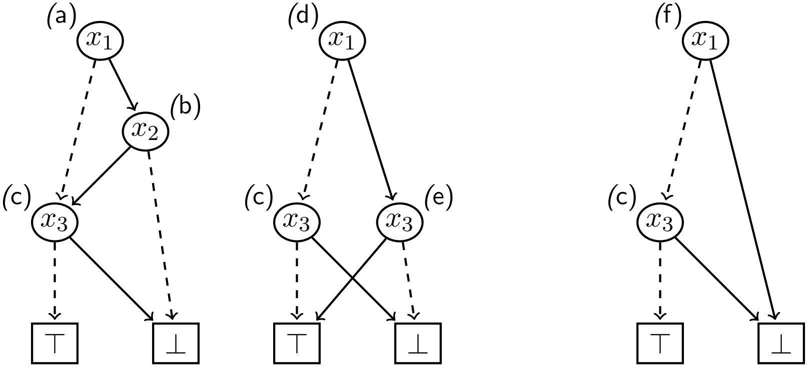
1. $(a, d) \mapsto (x_1,\ c,\ \bot)$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot,\ \bot) \mapsto \bot$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot) \mapsto \bot$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



1. $(a, d) \mapsto (x_1,\ c,\ \bot) \mapsto f$
2. $(c, c) \mapsto (x_3,\ \top,\ \bot) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\bot, \bot) \mapsto \bot$
5. $(b, e) \mapsto (x_2,\ \bot,\ \bot) \mapsto \bot$
6. $(\bot, e) \mapsto \bot$
7. $(\bot, \bot) \mapsto \bot$
8. $(c, e) \mapsto (x_3,\ \bot,\ \bot) \mapsto \bot$
9. $(\top, \bot) \mapsto \bot$
10. $(\bot, \top) \mapsto \bot$

The Apply Operation: Example (Conjunction)



1. $(a, d) \mapsto (x_1,\ c,\ \perp) \mapsto f$
2. $(c, c) \mapsto (x_3,\ \top,\ \perp) \mapsto c$
3. $(\top, \top) \mapsto \top$
4. $(\perp, \perp) \mapsto \perp$
5. $(b, e) \mapsto (x_2,\ \perp,\ \perp) \mapsto \perp$
6. $(\perp, e) \mapsto \perp$
7. $(\perp, \perp) \mapsto \perp$
8. $(c, e) \mapsto (x_3,\ \perp,\ \perp) \mapsto \perp$
9. $(\top, \perp) \mapsto \perp$
10. $(\perp, \top) \mapsto \perp$

The Apply Operation: Running Time

Question!

What is the running time of $apply(\Box, B_f, B_g)$**?**

The Apply Operation: Running Time

Question!

What is the running time of $apply(\Box, B_f, B_g)$**?**

Answer: Quadratic, $\mathcal{O}(|B_f||B_g|)$. The number of calls to the recursive function in which we do not return immediately is bounded by the pairs of nodes, because we cache the results for every pair of input nodes.

The Apply Operation: Running Time

Question!

What is the running time of $apply(\Box, B_f, B_g)$?

Answer: Quadratic, $\mathcal{O}(|B_f||B_g|)$. The number of calls to the recursive function in which we do not return immediately is bounded by the pairs of nodes, because we cache the results for every pair of input nodes.

Question!

What is the maximum size of the BDD representing $f \wedge g$?

The Apply Operation: Running Time

Question!

What is the running time of $apply(\square, B_f, B_g)$?

Answer: Quadratic, $\mathcal{O}(|B_f||B_g|)$. The number of calls to the recursive function in which we do not return immediately is bounded by the pairs of nodes, because we cache the results for every pair of input nodes.

Question!

What is the maximum size of the BDD representing $f \wedge g$?

Answer: $|B_f||B_g|$, same reason as above.

The Apply Operation: Running Time

Question!

What is the running time of $apply(\Box, B_f, B_g)$**?**

Answer: Quadratic, $\mathcal{O}(|B_f||B_g|)$. The number of calls to the recursive function in which we do not return immediately is bounded by the pairs of nodes, because we cache the results for every pair of input nodes.

Question!

What is the maximum size of the BDD representing $f \wedge g$**?**

Answer: $|B_f||B_g|$, same reason as above.

Question!

So, is the BDD representing each CNF formula of polynomial size?

The Apply Operation: Running Time

Question!

What is the running time of $apply(\Box, B_f, B_g)$?

Answer: Quadratic, $\mathcal{O}(|B_f||B_g|)$. The number of calls to the recursive function in which we do not return immediately is bounded by the pairs of nodes, because we cache the results for every pair of input nodes.

Question!

What is the maximum size of the BDD representing $f \wedge g$?

Answer: $|B_f||B_g|$, same reason as above.

Question!

So, is the BDD representing each CNF formula of polynomial size?

Answer: NO! If the CNF formula has N clauses, each represented with a BDD with K nodes then in the worst case we have K^N nodes.

Complexity of Operations (extracted from Darwiche and Marquis (2002)

	Queries				Transformations				
L	CO	VA	EQ	CT	$\bigwedge_n$	$\wedge$	$\bigvee_n$	$\vee$	$\neg C$
NNF	$\circ$	$\circ$	$\circ$	$\circ$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$
OBDD$<$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\bullet$	$\checkmark$	$\bullet$	$\checkmark$	$\checkmark$
DNF	$\checkmark$	$\circ$	$\circ$	$\circ$	$\bullet$	$\checkmark$	$\checkmark$	$\checkmark$	$\bullet$
CNF	$\circ$	$\checkmark$	$\circ$	$\circ$	$\checkmark$	$\checkmark$	$\bullet$	$\checkmark$	$\bullet$

$\checkmark$: Polynomial Time, $\bullet$ ($\circ$) Exponential time (unless P=NP)

Agenda

Summary

Reduced Ordered Binary Decision Diagrams (BDDs for short)

- Directed acyclic graph with two leave nodes $\perp$, $\top$
- Each node decomposes the function using the Shannon expansion
- Reduced
  any node with two identical children is removed
  equivalent nodes are merged
- Ordered
  Splitting variables always follow the same order along all paths
  $x_1 < x_2 < x_3 < \ldots < x_n$
- Properties:
  - Canonical representation
  - Efficient queries
  - Efficient operations

Some Caveats

We made some simplifying assumptions, and didn't mention a lot of optimizations. For example:

- Typically, negation is encoded on the edges
  - The same node is used to represent a function and its negation
  - Negation can be performed in constant time
  - One needs to handle this during the apply operation
- In our version of apply we always reach the terminals. However, we can return early in some cases by encoding some additional if conditions
  - $f \vee f = f$
  - $f \vee \bot = f$
  - $f \vee \top = \top$
  - $f \wedge f = f$
  - ...

Some Extensions

- What if instead of two terminals ($\perp, \top$), I have many (e.g. Numbers) $\rightarrow$Algebraic Decision Diagrams (ADD)

  (you can also use multiple BDDs or auxiliary variables in the BDD)

Some Extensions

- What if instead of two terminals ($\perp, \top$), I have many (e.g. Numbers)
  $\rightarrow$Algebraic Decision Diagrams (ADD)

  (you can also use multiple BDDs or auxiliary variables in the BDD)
- What if my variables are not Boolean but multi-valued?
  $\rightarrow$Multi-valued decision diagrams (MDDs)

  (you can also use as auxiliary variables in the BDD)

Some Extensions

- What if instead of two terminals ($\bot, \top$), I have many (e.g. Numbers)
  $\rightarrow$Algebraic Decision Diagrams (ADD)

  (you can also use multiple BDDs or auxiliary variables in the BDD)

- What if my variables are not Boolean but multi-valued?
  $\rightarrow$Multi-valued decision diagrams (MDDs)

  (you can also use as auxiliary variables in the BDD)

- Are there other possible reduction rules?
  $\rightarrow$Zero-Suppressed Decision Diagrams (ZDDs)

Some Extensions

- What if instead of two terminals ($\perp, \top$), I have many (e.g. Numbers)
  $\rightarrow$Algebraic Decision Diagrams (ADD)

  (you can also use multiple BDDs or auxiliary variables in the BDD)
- What if my variables are not Boolean but multi-valued?
  $\rightarrow$Multi-valued decision diagrams (MDDs)

  (you can also use as auxiliary variables in the BDD)
- Are there other possible reduction rules?
  $\rightarrow$Zero-Suppressed Decision Diagrams (ZDDs)
- Can I generalize Shannon Expansion?
  $\rightarrow$Sentential Decision Diagrams (SDDs)

Some Extensions

- What if instead of two terminals ($\bot, \top$), I have many (e.g. Numbers)
  $\rightarrow$Algebraic Decision Diagrams (ADD)

  (you can also use multiple BDDs or auxiliary variables in the BDD)

- What if my variables are not Boolean but multi-valued?
  $\rightarrow$Multi-valued decision diagrams (MDDs)

  (you can also use as auxiliary variables in the BDD)

- Are there other possible reduction rules?
  $\rightarrow$Zero-Suppressed Decision Diagrams (ZDDs)

- Can I generalize Shannon Expansion?
  $\rightarrow$Sentential Decision Diagrams (SDDs)

- Can I represent more complex arithmetic expressions?
  $\rightarrow$Edge-valued Multi-valued Decision Diagrams (EVMDDs)

Reading

- The art of Computer Programming. Knuth. Section 7.1.4. Binary Decision Diagrams
- Graph-based algorithms for Boolean function manipulation [Bryant (1986)].
  →Original paper exploring BDDs
- A Knowledge Compilation Map [Darwiche and Marquis (2002)].
  →A very good summary of how BDDs relate to other alternative representations

References I

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.