CS-E3220 Declarative Programming

Jussi Rintanen

Department of Computer Science
Aalto University

October 2, 2019

Plan for the Next Lectures

1. Logic as a data structure: Binary Decision Diagrams (today)
2. Transition system models
3. Symbolic state-space reachability in transition systems
4. Introduction to Modal Logic, including Linear Temporal Logic
5. Symbolic model-checking with LTL (computer-aided verification)

Motivation

- Complex systems (both SW and physical ones) have astronomic state spaces: 10^{10}, 10^{20}, 10^{30}, ... states
- Most of the interesting problems are NP-hard or harder (PSPACE-hard, EXP-hard, ...)
  - Does program have an execution such that ... (SW validation)
  - Choose a sequence of actions such that ... (control, decision-making)
  - Create a program such that ... (program synthesis)

Goal of the Lectures

- Methods for managing the complexity
- Methods for reasoning about very large state spaces
- Foundations of specification languages

Formulas as a Data Structure

- Uses of logic so far: reasoning (logical consequence)
- Other use: Formulas as a data structure:
  - sets of Boolean vectors
  - relations on Boolean vectors
  - computation with sets, relations

  Formula with n propositional variables = Set with up to 2^n elements
- Operations:
  - Set operations: $\cup$, $\cap$, intersection, complementation, cardinality, ...
  - Relational operations: $\cup$, $\cap$, join, projection, ...
- Applications:
  - computer-aided verification and validation
  - automated planning, intelligent control, ...

Application: Symbolic Breadth-First Search

Compute set of all states reachable from S_0, when one-step reachability relation is R:

1. $i := 0$
2. $S_0 :=$ initial states
3. $i := i + 1$
4. $S_i := S_{i-1} \cup \pi_2(S_{i-1} \bowtie R)$
5. if $S_i \not\subseteq S_{i-1}$, go to 3.

(This procedure will be discussed in more detail in next lectures.)

Application: Symbolic Breadth-First Search

Compute set of all states reachable from S_0, when one-step reachability relation is R:

1. $i := 0$
2. $S_0 :=$ formula describing initial state(s)
3. $i := i + 1$
4. $S_i := S_{i-1} \vee (\text{subst}_{X'/X}(\exists X.(S_{i-1} \wedge R)))$
5. if $S_i \not\models S_{i-1}$, go to 3.

(This procedure will be discussed in more detail in next lectures.)

Formulas as Data Structure: Operations

- Constructing a formula
  - Create atomic formula from a propositional variable x.
  - Given formulas ϕ_1 and ϕ_2, construct $\phi_1 \wedge \phi_2$
  - Given formulas ϕ_1 and ϕ_2, construct $\phi_1 \vee \phi_2$
  - Given formula ϕ, construct $\neg\phi$
- Manipulation
  - Given $f : X \to Y$, replace every occurrence of x by $f(x)$ (Renaming)
  - Given ϕ and $x \in X$, construct ψ such that ... ($\exists$-abstraction)
- Tests
  - Does $\phi_1 \equiv \phi_2$ hold? (Logical equivalence)
  - Does $\phi_1 \models \phi_2$ hold? (Logical consequence)
  - How many models does ϕ have? (Model counting)

Formulas as Data Structures: Normal Forms

Operations on formulas in some cases best done with formulas is some normal form, rather than arbitrary formulas.

- Negation Normal Form (NNF)
  - Practical alternative to unlimited formulas
  - Computational properties same as unlimited formulas
  - Basis of other normal forms: DNNF, d-DNNF
- (Reduced Ordered) Binary Decision Diagrams (OBDD)
  - Widely used in Computer-Aided Verification: circuits, state-space reachability
  - Properties very different from unlimited propositional logic
  - Implemented efficiently as a directed acyclic graph

Negation Normal Form

Definition

A formula with connectives $\vee$, $\wedge$ and $\neg$ is in negation normal form (NNF) if $\neg$ occurs only directly in front of a propositional variable.

Example

- $\neg(a \vee b)$ and $\neg(b \wedge c)$ are not in NNF.
- a, $\neg b$, $\neg a \vee b$ and $(\neg a \vee b) \wedge (c \vee (\neg d \wedge e))$ are in NNF.

Theorem

Any formula in DNF is in NNF. Any formula in CNF is in NNF.

Negation Normal Form

Theorem

For any formula ϕ there is an equivalent formula ψ that is in NNF.

Proof.

(Sketch) If a formula is not in NNF, at least one of the following rules is applicable.

1. $\neg(\phi_1 \vee \phi_2) \rightsquigarrow \neg\phi_1 \wedge \neg\phi_2$
2. $\neg(\phi_1 \wedge \phi_2) \rightsquigarrow \neg\phi_1 \vee \neg\phi_2$
3. $\neg\neg\phi \rightsquigarrow \phi$

Each application decreases the number of connectives inside $\neg$. If none of the rules is applicable, then the formula is in NNF. $\square$

Example: Unrestricted PL, Positive Occurrences

```
datatype fma = And of fma * fma      (* Define logic formulas *)
             | Or of fma * fma
             | Not of fma
             | Atom of int           (* Atoms as integers *)

fun posOccur (v,And(f1,f2)) = posOccur (v,f1) orelse posOccur (v,f2)
  | posOccur (v,Or(f1,f2)) = posOccur (v,f1) orelse posOccur (v,f2)
  | posOccur (v,Not f) = negOccur (v,f)
  | posOccur (v,Atom w) = (v=w)

and negOccur (v,And(f1,f2)) = negOccur (v,f1) orelse negOccur (v,f2)
  | negOccur (v,Or(f1,f2)) = negOccur (v,f1) orelse negOccur (v,f2)
  | negOccur (v,Not f) = posOccur (v,f)
  | negOccur (v,Atom _) = false
```

Example: NNF, Positive Occurrences

```
datatype fma = And of fma * fma      (* Define NNF formulas *)
             | Or of fma * fma
             | Atom of int           (* Atoms as integers *)
             | NegAtom of int

fun Neg (Atom i) = NegAtom i
  | Neg (NegAtom i) = Atom i
  | Neg (And(f1,f2)) = Or(Neg f1,Neg f2)
  | Neg (Or(f1,f2)) = And(Neg f1,Neg f2)

fun posOccur (v,And(f1,f2)) = posOccur (v,f1) orelse posOccur (v,f2)
  | posOccur (v,Or(f1,f2)) = posOccur (v,f1) orelse posOccur (v,f2)
  | posOccur (v,Atom w) = (v=w)
  | posOccur (v,NegAtom w) = false
```

Decomposability of NNF formulas

Definition (Decomposability)

A formula ϕ in NNF is decomposable if for every subformula $\psi_1 \wedge \psi_2$ of ϕ, the subformulas ψ_1 and ψ_2 don't share propositional variables.

Unlike general NNF, decomposable NNF has a poly-time test for SAT.

$$
\begin{array}{rclrcl}
SAT(x) & = & \text{true} & SAT(\neg x) & = & \text{true} \\
SAT(\top) & = & \text{true} & SAT(\phi_1 \wedge \phi_2) & = & SAT(\phi_1) \text{ and } SAT(\phi_2) \\
SAT(\bot) & = & \text{false} & SAT(\phi_1 \vee \phi_2) & = & SAT(\phi_1) \text{ or } SAT(\phi_2)
\end{array}
$$

Fails for general NNF: SAT$(x \wedge \neg x)$ will incorrectly return *true*.

Determinism of NNF formulas

Definition (Determinism)

A formula ϕ in NNF is deterministic if for every subformula $\psi_1 \vee \psi_2$ of ϕ, the subformulas ψ_1 and ψ_2 are mutually contradictory.

Determinism + Decomposability $\rightarrow$ poly-time model counting.

With determinism, $MC(\phi_1 \vee \phi_2) = MC(\phi_1) + MC(\phi_2)$.
With decomposability, $MC_{vars(\phi_1 \wedge \phi_2)}(\phi_1 \wedge \phi_2) = MC_{vars(\phi_1)}(\phi_1) \times MC_{vars(\phi_2)}(\phi_2)$.

Normal forms: DNNF and d-DNNF

Definition

DNNF is the subclass of NNF with decomposability.

Definition

d-DNNF is the subclass of DNNF with determinism.

Next we will consider a still stricter subclass of NNF, DNNF and d-DNNF, called Binary Decision Diagrams.

Boole's Expansion Theorem

Theorem (Boole's Expansion)

For any formula ϕ over X and any propositional variable $x \in X$, the formula $(x \wedge \phi[\top/x]) \vee (\neg x \wedge \phi[\bot/x])$ is logically equivalent to ϕ.

Expressible also in terms of ITE (IF-THEN-ELSE)

$$\mathsf{ITE}(x, \phi_1, \phi_2) = (x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$$

as

$$\phi \equiv \mathsf{ITE}(x, \phi[\top/x], \phi[\bot/x]).$$

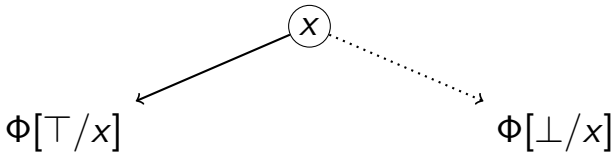
Binary Decision Diagrams: Underlying Idea

Binary Decision Diagrams arise from repeated elimination of variables by first translating ϕ into $(x \wedge \phi[\top/x]) \vee (\neg x \wedge \phi[\bot/x])$, and then eliminating other propositional variables in $\phi[\top/x]$ and $\phi[\bot/x]$.

Φ

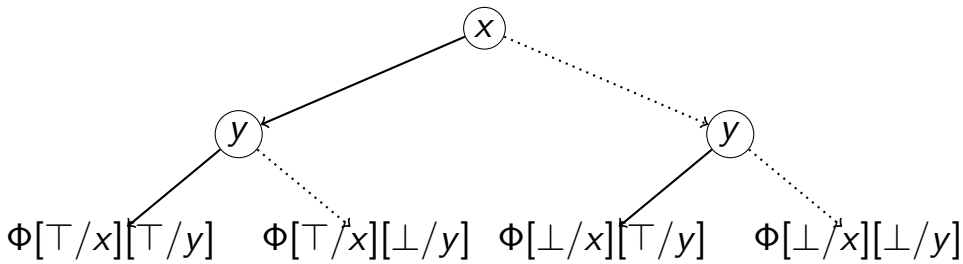
Binary Decision Diagrams: Underlying Idea

Binary Decision Diagrams arise from repeated elimination of variables by first translating ϕ into $(x \wedge \phi[\top/x]) \vee (\neg x \wedge \phi[\bot/x])$, and then eliminating other propositional variables in $\phi[\top/x]$ and $\phi[\bot/x]$.



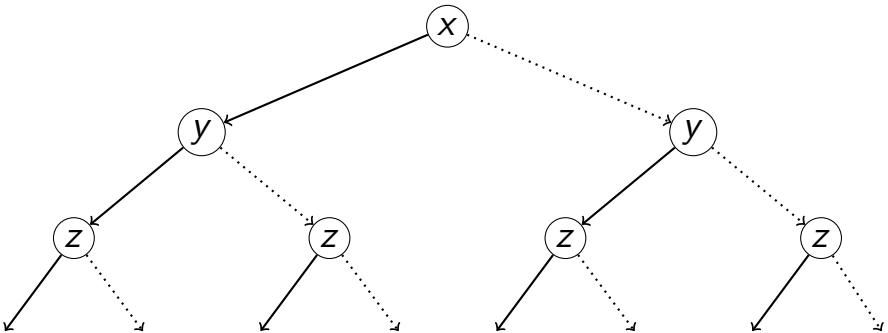
Binary Decision Diagrams: Underlying Idea

Binary Decision Diagrams arise from repeated elimination of variables by first translating ϕ into $(x \wedge \phi[\top/x]) \vee (\neg x \wedge \phi[\bot/x])$, and then eliminating other propositional variables in $\phi[\top/x]$ and $\phi[\bot/x]$.



Binary Decision Diagrams: Underlying Idea

Binary Decision Diagrams arise from repeated elimination of variables by first translating ϕ into $(x \wedge \phi[\top/x]) \vee (\neg x \wedge \phi[\bot/x])$, and then eliminating other propositional variables in $\phi[\top/x]$ and $\phi[\bot/x]$.



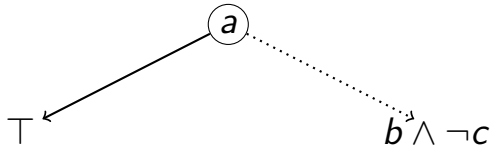
Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision tree.

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

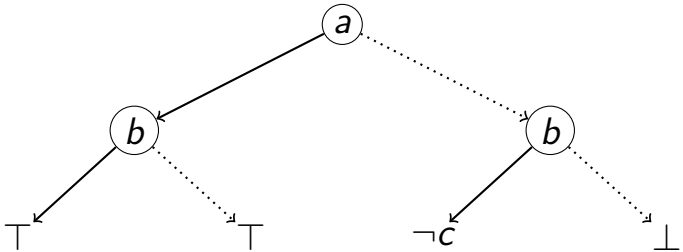
Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision tree.



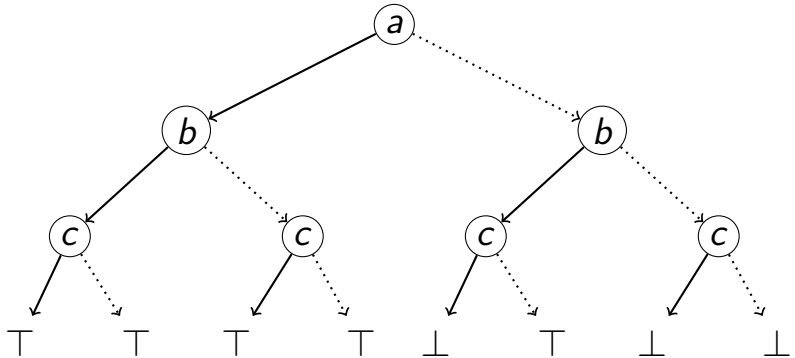
Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision tree.



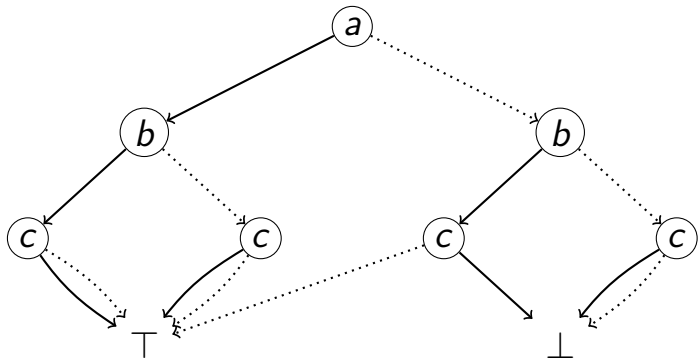
Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision tree.



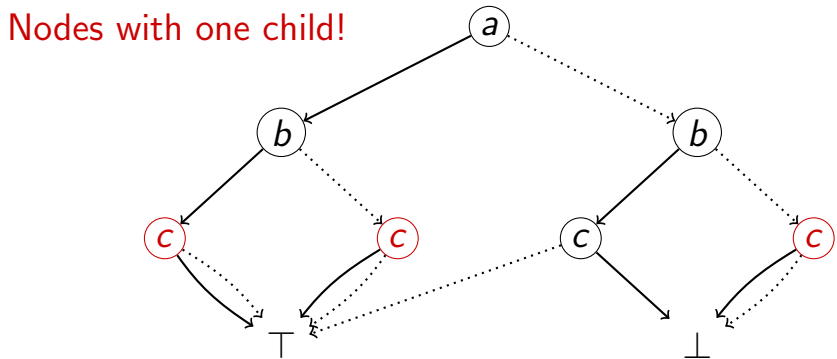
Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.



Binary Decision Diagrams: Construction

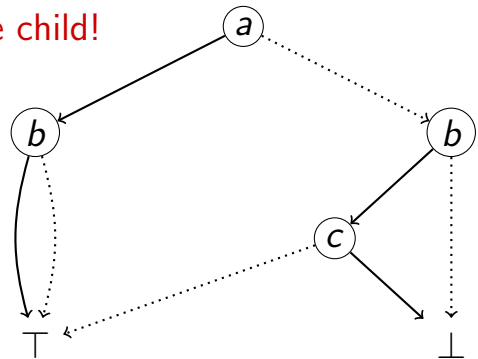
For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.



Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.

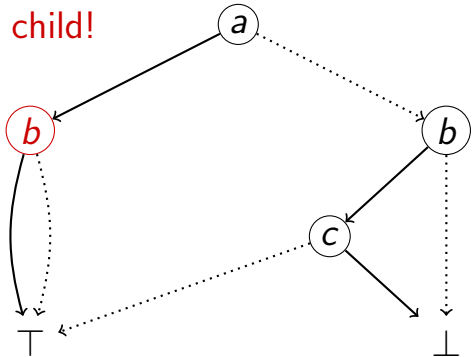
Nodes with one child!
Remove!



Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.

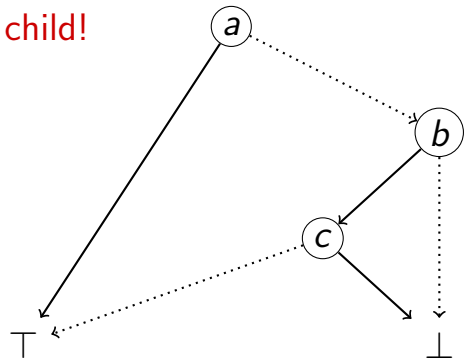
Nodes with one child!



Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.

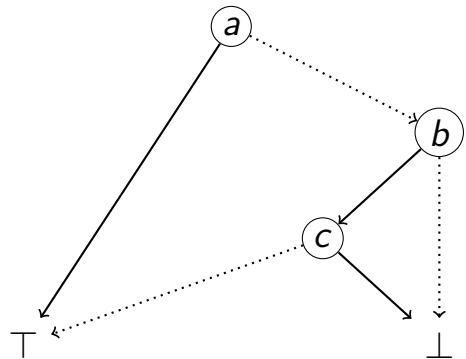
Nodes with one child!
Remove!



Binary Decision Diagrams: Construction

For $a \vee (b \wedge \neg c)$ we get the following binary decision diagram.

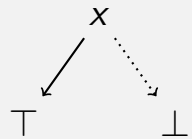
OBDD is complete!



$$ITE(a, \top, ITE(b, ITE(c, \bot, \top), \bot))$$

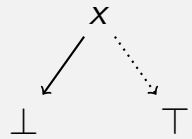
Constructing OBDDs Bottom-Up (Practical)

OBDD for a propositional variable x



Negating an OBDD

OBDD is negated by swapping $\top$ and $\bot$. Example: OBDD for $\neg x$ is



Constructing OBDDs Bottom-Up: Apply

By Boole's expansion theorem, for any binary connective $\oslash$:

$$\phi_1 \oslash \phi_2$$

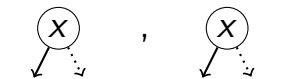
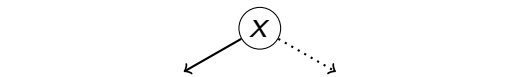
is logically equivalent to

$$(x \wedge (\phi_1[\top/x] \oslash \phi_2[\top/x])) \vee (\neg x \wedge (\phi_1[\bot/x] \oslash \phi_2[\bot/x])).$$

This is utilized in bottom-up construction of OBDDs in a procedure known as *Apply*.

Constructing OBDDs Bottom-Up: Apply

If root nodes of both constituent OBDDs have x:

$\mathsf{apply}(\oslash,$  $) =$ 

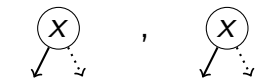
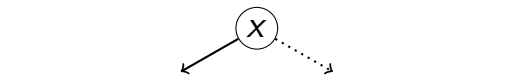
If root node of C comes later than x:

$\mathsf{apply}(\oslash,$ $, C) =$

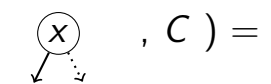
If both OBDDs are $\top$ or $\bot$: $\mathsf{apply}(\oslash, b, b') = b \oslash b'$

Constructing OBDDs Bottom-Up: Apply

If root nodes of both constituent OBDDs have x:

$\mathrm{apply}(\oslash,$  $) =$ 

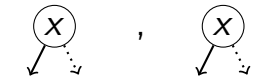
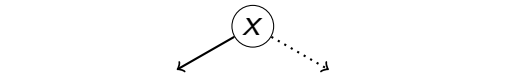
If root node of C comes later than x:

$\mathrm{apply}(\oslash,$  $, C) =$ 

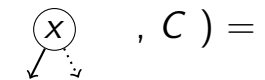
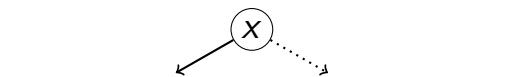
If both OBDDs are $\top$ or $\bot$: $\mathrm{apply}(\oslash, b, b') = b \oslash b'$

Constructing OBDDs Bottom-Up: Apply

If root nodes of both constituent OBDDs have x:

$\mathsf{apply}(\oslash,$  $) =$ 

If root node of C comes later than x:

$\mathsf{apply}(\oslash,$  $, C) =$ 

If both OBDDs are $\top$ or $\bot$: $\mathsf{apply}(\oslash, b, b') = b \oslash b'$

Constructing OBDDs Bottom-Up: Apply

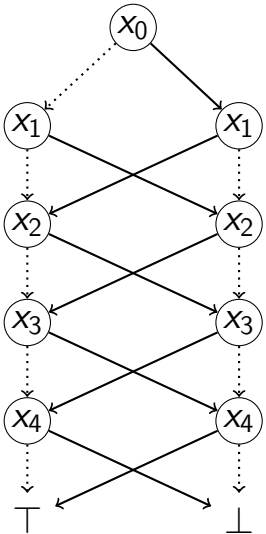
Implementation has to eliminate 1-child nodes and duplicate nodes:

- Memoization: If an OBDD node already exists, do not create a second one with same x and same children.
- If the two recursive calls apply($\oslash$, ,) and apply($\oslash$, ,) return the same OBDD B, then return B instead of constructing node for x with duplicate children B and B.

Properties of OBDDs

- For a given Boolean function and given variable ordering, there is exactly one OBDD (modulo graph isomorphism). (Canonicity)
- An OBDD is valid iff it is the single node $\top$.
- An OBDD is satisfiable iff it is not the single node $\bot$.
- With memoization, two OBDDs are logically equivalent iff they have the same root node.

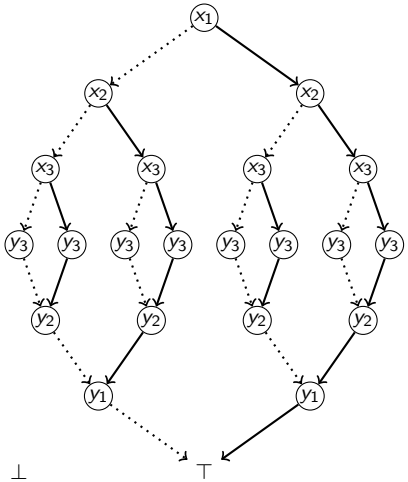
Size of OBDDs



This OBDD is true iff the number of true propositional variables is even.

An equivalent propositional formula over $x_0, \ldots, x_n$ has exponential size. Compact representation requires auxiliary variables as in the Tseitin transformation.

Size of OBDDs

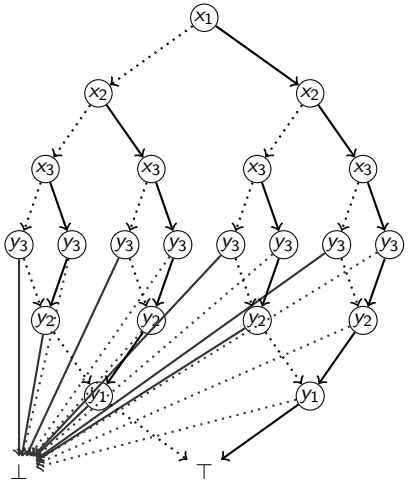


OBDD for
$(x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge (x_3 \leftrightarrow y_3)$

It has $> 2^n$ nodes. Not good!

Each y_3 node encodes one valuation of $x_1, \ldots, x_n$.

Size of OBDDs

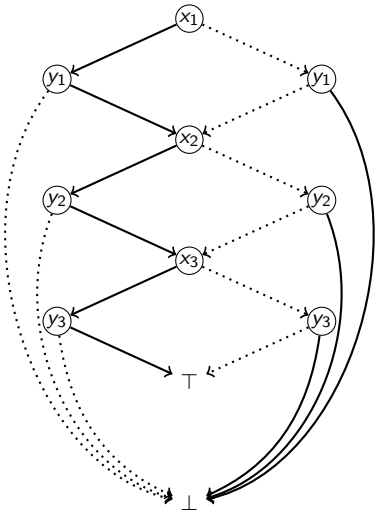


OBDD for
$(x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge (x_3 \leftrightarrow y_3)$

It has $> 2^n$ nodes. Not good!

Each y_3 node encodes one valuation of $x_1, \ldots, x_n$.

Size of OBDDs



OBDD for $(x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge (x_3 \leftrightarrow y_3)$

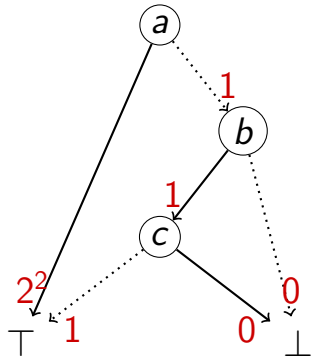
It has $3n$ non-leaf nodes, which is linear in n.

Ordering $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$ much better than $x_1, \ldots, x_n, y_1, \ldots, y_n$.

How to Find a Good Variable Ordering?

- Ad hoc variable ordering
  - Determined by programmer
  - General principle: related variables next to each other
- Automatic variable ordering methods
  - Standard feature of OBDD implementations
  - Improves a given variable ordering by swapping neighboring variables
  - Can be invoked periodically as the OBDDs grow and change

Model-Counting with OBDDs



Model count for $a \vee (b \wedge \neg c)$ (read from the root node) is $2^2 + 1 = 5$.

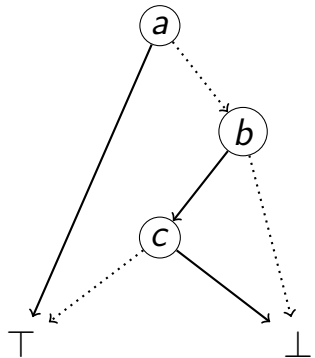
Model-Counting with OBDDs

For OBDD B with variables $1, \ldots, n$, MC(B,1) returns the number of valuations that make B true. (Initially, memo[b] := -1 for all nodes b.))

```
 2: PROCEDURE MC(B : OBDD,i : int)
 3: BEGIN
 4:   IF B = ⊤ THEN RETURN 2^(n+1−i);
 5:   IF B = ⊥ THEN RETURN 0;
 6:   IF memo[B] ≥ 0 THEN RETURN memo[B];
 7:   i_B := B.nodeVar;
 8:   M := 2^(i_B − i);
 9:   memo[B] := M×(MC(B.posChild,i_b + 1) + MC(B.negChild,i_b + 1));
10:   RETURN memo[i_b];
11: END
```

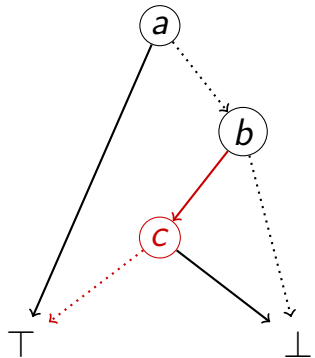
Restrict – Fixing One Propositional Variable

From $\phi = a \vee (b \wedge \neg c)$ construct $\phi[\bot/c]$ i.e. $a \vee (b \wedge \neg \bot)$ i.e. $a \vee b$.



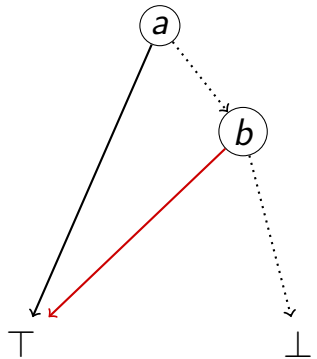
Restrict – Fixing One Propositional Variable

From $\phi = a \vee (b \wedge \neg c)$ construct $\phi[\bot/c]$ i.e. $a \vee (b \wedge \neg \bot)$ i.e. $a \vee b$.



Restrict – Fixing One Propositional Variable

From $\phi = a \vee (b \wedge \neg c)$ construct $\phi[\bot/c]$ i.e. $a \vee (b \wedge \neg \bot)$ i.e. $a \vee b$.



Restrict – Fixing One Propositional Variable

Restrict constructs $\phi[\top/x]$ or $\phi[\bot/x]$ given ϕ and a literal on x.

```
 2: PROCEDURE restrict(B : OBDD,x : variable, positive : bool)
 3: BEGIN
 4:     IF positive
 5:     THEN
 6:       FOR EACH node n in B DO
 7:         IF n.posChild.nodeVar = x THEN n.posChild := n.posChild.posChild;
 8:         IF n.negChild.nodeVar = x THEN n.negChild := n.negChild.posChild;
 9:       END DO
10:     ELSE
11:       FOR EACH node n in B DO
12:         IF n.posChild.nodeVar = x THEN n.posChild := n.posChild.negChild;
13:         IF n.negChild.nodeVar = x THEN n.negChild := n.negChild.negChild;
14:       END DO
15: END
```

Existential Abstraction

Definition

$\exists x.\phi$ is defined as $\phi[\top/x] \vee \phi[\bot/x]$

For any valuation v, $v \models \phi$ iff both $v_t \models \exists x.\phi$ and $v_f \models \exists x.\phi$, where v_f and v_t are like v except that $v_f(x) = 0$ and $v_t(x) = 1$.

- Example: $\exists b.((a \to b) \wedge (b \to c))$ is equivalent to $a \to c$
- Example: $\exists a \exists b.(a \vee b)$ is equivalent to $\top$
- $\exists x.B$ is computed as restrict(B,x,true) $\vee$ restrict(B,x,false)
- Applications (next lecture!):
  - Relational projection (as part of composing relations)
  - Boolean matrix multiplication

Universal Abstraction

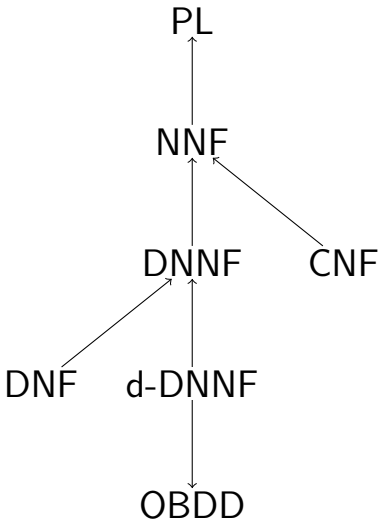
Definition

$\forall x.\phi$ is defined as $\phi[\top/x] \wedge \phi[\bot/x]$

For any valuation v, if $v_t \models \phi$ and $v_f \models \phi$, where v_f and v_t are like v except that $v_f(x) = 0$ and $v_t(x) = 1$, then $v \models \forall x.\phi$.

- Example: $\forall b.((a \rightarrow b) \wedge (b \rightarrow c))$ is equivalent to $c \wedge \neg a$
- Example: $\forall a \forall b.(a \vee b)$ is equivalent to $\bot$
- $\forall x.B$ is computed as restrict(B,x,true) $\wedge$ restrict(B,x,false)
- Applications: some relational operations

Normal Forms for the Propositional Logic



Normal Forms: Complexity Summary

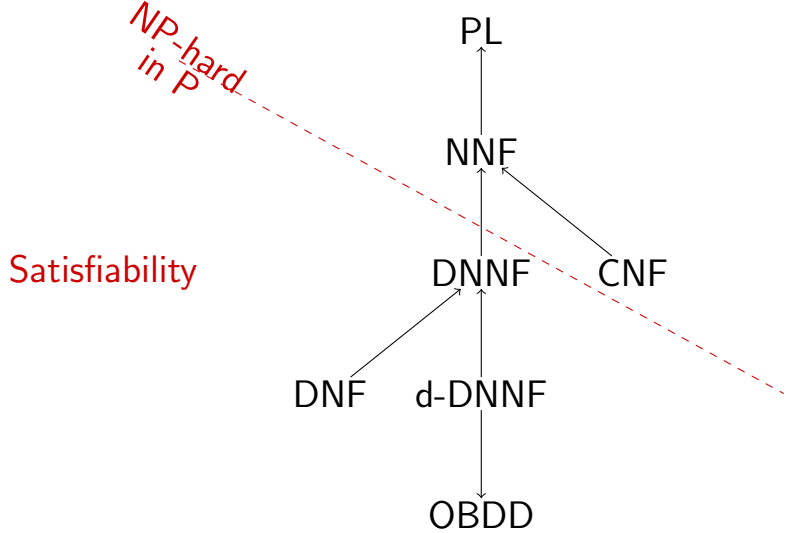
operation	PL	NNF	CNF	DNF	DNNF	d-DNNF	OBDD
$\phi \in$SAT	NP	NP	NP	P	P	P	P
$\phi \in$TAUT	co-NP	co-NP	P	co-NP	co-NP	P	P
$\phi \models c$	co-NP	co-NP	co-NP	P	P	P	P
model counting	NP	NP	NP	NP	NP	P	P
$n \times \wedge$	P	P	P	exp	exp	exp	exp
$n \times \vee$	P	P	exp	P	P	exp	exp
$\neg$	P	P	exp	exp	exp	?	P

Construction with $\vee$, $\wedge$ and $\neg$ with the result in the same normal form.

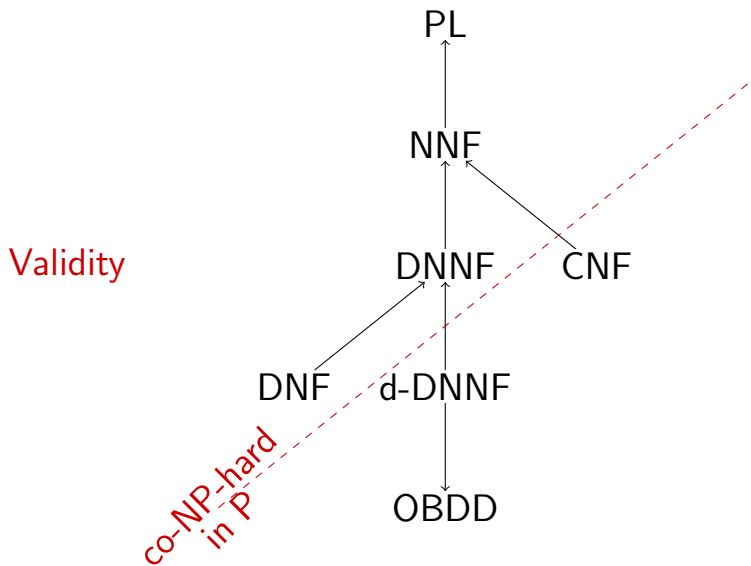
Above, NP denotes "NP-hard", P denotes "polynomial time".

More restrictive normal form $\rightarrow$ more expensive construction
More restrictive normal form $\rightarrow$ more efficient tests

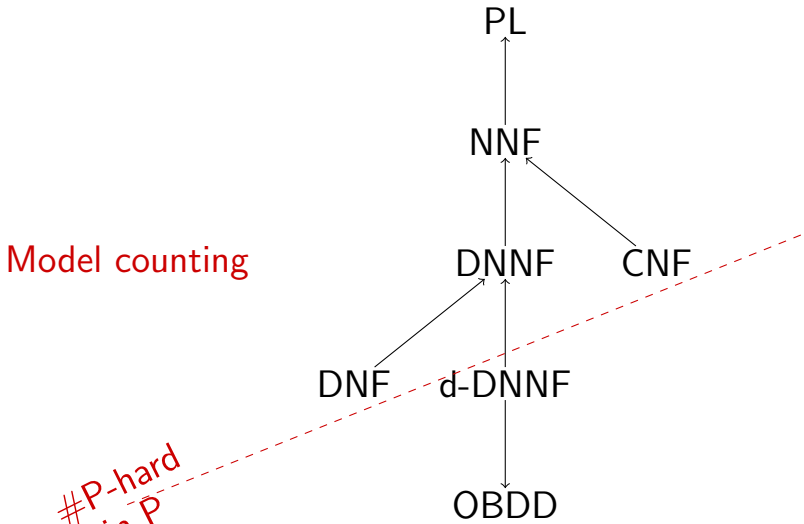
Normal Forms for the Propositional Logic



Normal Forms for the Propositional Logic



Normal Forms for the Propositional Logic



Generalizations of OBDDs

OBDD generalizations (with no direct counterpart in prop. logic):

- Algebraic Decision Diagrams (ADD):
  - decision diagrams $B^n \to \mathbb{R}$ with real-valued leafs
  - numeric matrix multiplication with generalized $\exists$ abstraction
  - Markov decision process (MDP) value iteration with very large state spaces
- Affine ADDs
  - Generalization of ADDs
  - Leaf value determined by arithmetic operations on the path from root to leaf
  - More compact than ADDs, as not every leaf value needs a separate leaf node
- Multi-Valued Decision Diagrams (MVDD)
  - Functions $D^n \to D$ over non-Boolean domains D

Formulas as Sets of Valuations

Formulas Representing Sets

Formula $\phi \sim$ Set of valuations v such that $v \models \phi$

We visualize valuations $v : X \to \{0, 1\}$ as bit-vectors of length $|X|$.

Example

Let $X = \{A, B, C, D\}$.
The valuation that assigns 1 to A and C, and 0 to B and D, corresponds to vector $\overset{ABCD}{1010}$.
The valuation assigning 1 only to B corresponds to $\overset{ABCD}{0100}$.

Formulas as Sets of Valuations

Example

The formula B represents all bit-vectors of the form ?1??,
The formula A represents all bit-vectors 1???.
$B \sim \{0\mathbf{1}00, 0\mathbf{1}01, 0\mathbf{1}10, 0\mathbf{1}11, 1\mathbf{1}00, 1\mathbf{1}01, 1\mathbf{1}10, 1\mathbf{1}11\}$
$A \sim \{\mathbf{1}000, \mathbf{1}001, \mathbf{1}010, \mathbf{1}011, \mathbf{1}100, \mathbf{1}101, \mathbf{1}110, \mathbf{1}111\}$.

Example

$\neg B$ represents all bit-vectors of the form ?0??, which is the set

$$\{0\mathbf{0}00, 0\mathbf{0}01, 0\mathbf{0}10, 0\mathbf{0}11, 1\mathbf{0}00, 1\mathbf{0}01, 1\mathbf{0}10, 1\mathbf{0}11\}.$$

This is the complement of the set represented by B.

Set-Theoretic Operations

If ϕ_1 and ϕ_2 respectively represent sets S_1 and S_2, then

1. $\phi_1 \wedge \phi_2$ represents the set $S_1 \cap S_2$,
2. $\phi_1 \vee \phi_2$ represents the set $S_1 \cup S_2$, and
3. $\neg\phi_1$ represents the set $\overline{S_1}$.

Example

$A \wedge B$ represents the set $\{1100, 1101, 1110, 1111\}$ and
$A \vee B$ represents the set
$\{0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$.

Properties of Sets

Questions about sets reducible to questions about formulas.

1. Is ϕ satisfiable? $=$ Is the set represented by ϕ non-empty?
2. $\phi \models \alpha$? $=$ Are sets represented by ϕ and α in the subset relation $\subseteq$?
3. Is ϕ valid? $=$ Is the set represented by ϕ the universal set?

Formulas as Sets: Summary

sets	**formulas over X**
those $\frac{2^{\lvert X\rvert}}{2}$ bit-vectors where x is true	$x \in X$
$\overline{E}$ (complement)	$\neg E$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \backslash F$ (set difference)	$E \wedge \neg F$
the empty set $\emptyset$	$\bot$ (constant *false*)
the universal set	$\top$ (constant *true*)
question about sets	**question about formulas**
$E \subseteq F$?	$E \models F$?
$E \subset F$?	$E \models F$ and $F \not\models E$?
$E = F$?	$E \models F$ and $F \models E$?

Roadmap for Next Lecture

- Relations as formulas
- Relational operations as operations on formulas ($\exists x$, $\wedge$, ...)
- Symbolic breadth-first search through formula/BDD operations