Artificial Intelligence

Propositional logic

E. Giunchiglia, F. Leofante, A. Tacchella

Computer Engineering
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Last update: February 20, 2023

Agenda - Propositional logic

1. Motivations
2. Syntax, Semantics and Complexity
3. Representing formulas
4. Reasoning techniques
    - Truth Tables
    - Procedures for formulas in CNF
        - Procedures based on Variable elimination
        - Davis-Putnam-Logemann-Loveland procedure (DPLL)
    - Ordered Binary Decision Diagrams (OBDDs)
    - Incomplete techniques
5. Beyond propositional logic

Why **logic** in AI?

Humans, it seems, know things; and what they know helps them do things. . . . They make strong claims about how the intelligence of humans is achieved — not by purely reflex mechanisms but by processes of ***reasoning*** *that operate on internal* ***representations*** *of knowledge.* *[Stuart Russell, Peter Norvig. AI a Modern Approach. 3rd edition.]*

Logic is a well-known formalism, characterized by a (formal) language which is used to **represent** the known fact, equipped with a (formal) deduction mechanism which allows to **reason** about the known facts and decide.

Why **propositional** logic?

Knowledge representation

Any finite description (i.e., with finitely many variables, each with finitely many values) can be formalized in propositional logic.

Reasoning about knowledge

Very very efficient procedure exists (a competition is run each year).

Why **propositional** logic?

Knowledge representation

- Facts being **true** or **false**
  (E.g., *“It is raining”*)
- Logical combination of facts through **conjunction**, **disjunction**, **negation**, ...
  (E.g., *“It is raining AND I have my umbrella”*

Reasoning about knowledge

- Facts being **(in)consistent**
  (E.g., *“To be AND not to be”* is inconsistent)
- Facts being **consequence** of other facts
  (E.g., facts *“It is raining”* and *“IF it is raining THEN I take my umbrella”* have *“I take my umbrella”* as a consequence)

Syntax

Given a set of (propositional) variables/atoms $A, A_1, \ldots, B, B_1, \ldots$, the set of (well formed) formulas (wffs) is inductively defined as:

- ▷ variables are wffs;
- ▷ if φ_1 and φ_2 are wffs, then $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$ are wffs;
- ▷ nothing else is a wff.

Syntax

Given a set of (propositional) variables/atoms $A, A_1, \ldots, B, B_1, \ldots$, the set of (well formed) formulas (wffs) is inductively defined as:

- ▷ variables are wffs;
- ▷ if φ_1 and φ_2 are wffs, then $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$ are wffs;
- ▷ nothing else is a wff.

1. Literal: an atom A_i (*positive* l.) or its negation $\neg A_i$ (*negative* l.)
2. *Atoms*(φ): the set $\{A_1, ..., A_N\}$ of atoms occurring in φ.

Note: Sometimes,

1. "$\overline{A}$" is used in place of "$\neg A$",
2. "$(w_1 \supset w_2)$" is used in place of "$(w_1 \rightarrow w_2)$",
3. "$(w_1 \equiv w_2)$" is used in place of "$(w_1 \leftrightarrow w_2)$".

Some jargon...

- The term **Boolean** is often used instead of “propositional”
- “$\neg$” is called **negation** (NOT)
- “$\wedge$” is called **conjunction** (AND)
- “$\vee$” is (inclusive) **disjunction** (OR)
- “$\rightarrow$” is (material) **implication**
- “$\leftrightarrow$” is **equivalence**
- $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ are called **(logical) connectives**
- Circuit analog: a formula is like a **combinatorial digital circuit**:
  - variables are the **inputs**
  - connectives are the **gates**

Exercise

1. Which of the following are well-formed propositional formulas?
   1. $\vee AB$
   2. $(\neg(A \rightarrow (A \wedge B)))$
   3. $A \neg B$
   4. $((\neg(A \rightarrow (B = C))))$
   5. $(A \wedge \neg B) \vee (B \rightarrow C)$
   6. $A \rightarrow B \wedge C$
   7. $\neg A \rightarrow B \wedge C \vee D$
   8. $A \wedge B \wedge C$
   9. $A \rightarrow B \rightarrow C$

Exercise

1. Which of the following are well-formed propositional formulas?
   1. $\vee AB$
   2. $(\neg(A \rightarrow (A \wedge B)))$
   3. $A\neg B$
   4. $((\neg(A \rightarrow (B = C))))$
   5. $(A \wedge \neg B) \vee (B \rightarrow C)$
   6. $A \rightarrow B \wedge C$
   7. $\neg A \rightarrow B \wedge C \vee D$
   8. $A \wedge B \wedge C$
   9. $A \rightarrow B \rightarrow C$
2. Assume the following priority among connectives (first, top priority): $\neg, \wedge, \vee, \rightarrow$. For which of the above, do we need parentheses to have “unique readability” and is "unique readability" important?

Exercise

1. Which of the following are well-formed propositional formulas?
   1. $\vee AB$
   2. $(\neg(A \rightarrow (A \wedge B)))$
   3. $A\neg B$
   4. $((\neg(A \rightarrow (B = C))))$
   5. $(A \wedge \neg B) \vee (B \rightarrow C)$
   6. $A \rightarrow B \wedge C$
   7. $\neg A \rightarrow B \wedge C \vee D$
   8. $A \wedge B \wedge C$
   9. $A \rightarrow B \rightarrow C$
2. Assume the following priority among connectives (first, top priority): $\neg, \wedge, \vee, \rightarrow$. For which of the above, do we need parentheses to have “unique readability” and is "unique readability" important?

Convention: When parentheses are omitted, the above priorities among connectives are assumed and it is assumed connectives to be right associative.

Semantics

▷ Total truth assignment μ for φ:
$\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$.

▷ Partial Truth assignment μ for φ:
$\mu : \mathcal{A} \longmapsto \{\top, \bot\}$, $\mathcal{A} \subset Atoms(\varphi)$.

▷ Set and formula representation of an assignment:

- μ can be represented as a set of literals:

  ex: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies \{A_1, \neg A_2\}$

- μ can be represented as a formula:

  ex: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies A_1 \wedge \neg A_2$

Semantics

- It is convenient to extend the language with the symbols "$\top$" (called **true**) and "$\perp$" (called **false**), having the fixed natural interpretation.
- Truth and falsity of variables are determined by the assignment μ.
- Truth and falsity of a formula are determined thanks to the truth tables for $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

Semantics

Truth tables:

φ_1	φ_2	$\neg\varphi_1$	$(\varphi_1 \wedge \varphi_2)$	$(\varphi_1 \vee \varphi_2)$	$(\varphi_1 \rightarrow \varphi_2)$	$(\varphi_1 \leftrightarrow \varphi_2)$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

Circuit analog: if we think about a formula as a circuit,

1. the variables are the inputs,
2. $\top$ and $\bot$ corresponds to the circuit values 1 and 0 respectively,
3. the assignment is the input signal to the circuit,
4. the truth value of the formula is the output of the circuit.

Semantics

Formal definition:

▷ $\mu \models \varphi$ (μ satisfies φ):

- if φ is a variable A, $\mu \models \varphi \Longleftrightarrow \mu(A) = \top$,
- if φ is $\neg\varphi_1$, $\mu \models \varphi \Longleftrightarrow$ *not* $\mu \models \varphi_1$ (or, $\mu \not\models \varphi_1$),
- if φ is $(\varphi_1 \wedge \varphi_2)$, $\mu \models \varphi \Longleftrightarrow \mu \models \varphi_1$ *and* $\mu \models \varphi_2$,
- if φ is $(\varphi_1 \vee \varphi_2)$, $\mu \models \varphi \Longleftrightarrow \mu \models \varphi_1$ *or* $\mu \models \varphi_2$,
- if φ is $(\varphi_1 \rightarrow \varphi_2)$, $\mu \models \varphi \Longleftrightarrow \mu \not\models \varphi_1$ *or* $\mu \models \varphi_2$,
- if φ is $(\varphi_1 \leftrightarrow \varphi_2)$, $\mu \models \varphi \Longleftrightarrow$ *either* $\mu \models \varphi_1$ *and* $\mu \models \varphi_2$ or $\mu \not\models \varphi_1$ *and* $\mu \not\models \varphi_2$,

Semantics

Formal definition:

- ▷ $\mu \models \varphi$ (μ satisfies φ):
  - if φ is a variable *A*, $\mu \models \varphi \iff \mu(A) = \top$,
  - if φ is $\neg\varphi_1$, $\mu \models \varphi \iff$ *not* $\mu \models \varphi_1$ (or, $\mu \not\models \varphi_1$),
  - if φ is $(\varphi_1 \wedge \varphi_2)$, $\mu \models \varphi \iff \mu \models \varphi_1$ *and* $\mu \models \varphi_2$,
  - if φ is $(\varphi_1 \vee \varphi_2)$, $\mu \models \varphi \iff \mu \models \varphi_1$ *or* $\mu \models \varphi_2$,
  - if φ is $(\varphi_1 \rightarrow \varphi_2)$, $\mu \models \varphi \iff \mu \not\models \varphi_1$ *or* $\mu \models \varphi_2$,
  - if φ is $(\varphi_1 \leftrightarrow \varphi_2)$, $\mu \models \varphi \iff$ *either* $\mu \models \varphi_1$ *and* $\mu \models \varphi_2$ or $\mu \not\models \varphi_1$ *and* $\mu \not\models \varphi_2$,
- ▷ If $\mu \models \varphi$ then μ is a model of φ.

Satisfiability, unsatisfiability, entailment, tautology

- φ is satisfiable iff for some μ, $\mu \models \varphi$,

Satisfiability, unsatisfiability, entailment, tautology

- ▷ φ is satisfiable iff for some μ, $\mu \models \varphi$,
- ▷ a set of wffs Γ is satisfiable iff for some μ, for every $\varphi \in \Gamma$, $\mu \models \varphi$,

Satisfiability, unsatisfiability, entailment, tautology

- ▷ φ is satisfiable iff for some μ, $\mu \models \varphi$,
- ▷ a set of wffs Γ is satisfiable iff for some μ, for every $\varphi \in \Gamma$, $\mu \models \varphi$,
- ▷ φ is unsatisfiable or contradictory iff for evey assignment μ, $\mu \not\models \varphi$,

Satisfiability, unsatisfiability, entailment, tautology

- φ is satisfiable iff for some μ, $\mu \models \varphi$,
- a set of wffs Γ is satisfiable iff for some μ, for every $\varphi \in \Gamma$, $\mu \models \varphi$,
- φ is unsatisfiable or contradictory iff for evey assignment μ, $\mu \not\models \varphi$,
- if Γ is a (possibily infinite) set of formulas, $\Gamma \models \varphi$ (Γ entails φ), or, φ (logically) follows from Γ iff for every model μ of the formulas in Γ, $\mu \models \varphi$,

Satisfiability, unsatisfiability, entailment, tautology

- φ is satisfiable iff for some μ, $\mu \models \varphi$,
- a set of wffs Γ is satisfiable iff for some μ, for every $\varphi \in \Gamma$, $\mu \models \varphi$,
- φ is unsatisfiable or contradictory iff for evey assignment μ, $\mu \not\models \varphi$,
- if Γ is a (possibily infinite) set of formulas, $\Gamma \models \varphi$ (Γ entails φ), or, φ (logically) follows from Γ iff for every model μ of the formulas in Γ, $\mu \models \varphi$,
- $\models \varphi$ (φ is valid or a tautology) iff for every μ, $\mu \models \varphi$.

Satisfiability, unsatisfiability, entailment, tautology

Fact: If Γ and Γ' are two sets of wffs with $\Gamma' \subseteq \Gamma$, then

1. If Γ is satisfiable then also Γ' is satisfiable,
2. if φ follows from Γ' then φ follows also from Γ (monotonicity).

Satisfiability, unsatisfiability, entailment, tautology

Fact: If Γ and Γ' are two sets of wffs with $\Gamma' \subseteq \Gamma$, then

1. If Γ is satisfiable then also Γ' is satisfiable,
2. if φ follows from Γ' then φ follows also from Γ (monotonicity).

Question: What is the shortest tautology?

Satisfiability, unsatisfiability, entailment, tautology

Fact: If Γ and Γ' are two sets of wffs with $\Gamma' \subseteq \Gamma$, then

1. If Γ is satisfiable then also Γ' is satisfiable,
2. if φ follows from Γ' then φ follows also from Γ (monotonicity).

Question: What is the shortest tautology?

Question: Does there exist a set of formulas entailing $\perp$?

Exercise

Let $Atoms(\varphi) = \{A, B, C\}$.

Consider an assignment μ where:

$\mu(A) = \bot$, $\mu(B) = \top$ and $\mu(C) = \top$.

Does μ satisfy the following formulas?

1. $(A \rightarrow \neg B) \vee \neg(C \wedge B)$
2. $\neg(\neg A \rightarrow \neg B) \wedge C$
3. $(\neg A \wedge \neg B) \rightarrow (A \wedge \neg C)$
4. $C \vee B$

Equivalence and equi-satisfiability

- φ_1 and φ_2 are (logically) equivalent iff, for every μ, $\mu \models \varphi_1$ iff $\mu \models \varphi_2$
- φ_1 and φ_2 are equi-satisfiable iff exists μ_1 s.t. $\mu_1 \models \varphi_1$ iff exists μ_2 s.t. $\mu_2 \models \varphi_2$

Fact: If φ_1, φ_2 are equivalent then they are also equi-satisfiable, while the viceversa does not necessarily hold.

Fact: If two formulas φ_1 and φ_2 are logically equivalent, you can replace φ_1 for φ_2 in φ and get a logically equivalent formula.

Example

Given that A is equivalent to $A \wedge (A \vee B)$, then $(A \wedge (A \vee B) \wedge \varphi)$ is equivalent to $(A \wedge \varphi)$.

Equivalent formulas

$$\begin{array}{lcl}
(\varphi_1 \rightarrow \varphi_2) & \Longleftrightarrow & (\neg\varphi_1 \vee \varphi_2) \\
\neg(\varphi_1 \rightarrow \varphi_2) & \Longleftrightarrow & (\varphi_1 \wedge \neg\varphi_2) \\
(\varphi_1 \leftrightarrow \varphi_2) & \Longleftrightarrow & ((\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)) \\
 & \Longleftrightarrow & ((\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2)) \\
\neg(\varphi_1 \leftrightarrow \varphi_2) & \Longleftrightarrow & (\neg\varphi_1 \leftrightarrow \varphi_2) \\
 & \Longleftrightarrow & (\varphi_1 \leftrightarrow \neg\varphi_2) \\
 & \Longleftrightarrow & ((\varphi_1 \vee \varphi_2) \wedge (\neg\varphi_1 \vee \neg\varphi_2)) \\
 & \Longleftrightarrow & ((\varphi_1 \wedge \neg\varphi_2) \vee (\neg\varphi_1 \wedge \varphi_2))
\end{array}$$

For any wff, there is an equivalent one using only $\{\neg, \wedge, \vee\}$.

Equivalent formulas

$$
\begin{array}{lcl}
\neg\neg\varphi_1 & \iff & \varphi_1 \\[1ex]
(\varphi_1 \vee \varphi_2) & \iff & \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\
\neg(\varphi_1 \vee \varphi_2) & \iff & (\neg\varphi_1 \wedge \neg\varphi_2) \\[1ex]
(\varphi_1 \wedge \varphi_2) & \iff & \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\neg(\varphi_1 \wedge \varphi_2) & \iff & (\neg\varphi_1 \vee \neg\varphi_2)
\end{array}
$$

For any wff, there is an equivalent one using only

1. $\{\neg, \wedge\}$, or
2. $\{\neg, \vee\}$, or
3. $\{\wedge, \vee\}$ applied to literals.

Equivalence and satisfiability

In propositional logic, the following statements are equivalent:

1. φ_1 and φ_2 are equivalent,
2. φ_1 follows from φ_2 and viceversa,
3. $(\varphi_1 \leftrightarrow \varphi_2)$ is a tautology,
4. $\neg(\varphi_1 \leftrightarrow \varphi_2)$ is not satisfiable.

Also the following statements are equivalent:

1. φ follows from $\varphi_1, \ldots, \varphi_n$,
2. φ follows from $(\varphi_1 \wedge \ldots \wedge \varphi_n)$,
3. $(\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \varphi$ is a tautology,
4. $(\varphi_1 \wedge \ldots \wedge \varphi_n) \wedge \neg\varphi$ is not satisfiable.

(Logical) equivalence and entailment can be reduced to satisfiability!

Reasoning in propositional logic

Problem SAT

Given a propositional formula φ decide whether φ is satisfiable.

- The SAT problem is the first problem to be proved **NP-complete**
- We do not know an algorithm to solve SAT which takes polynomial time in $|Atoms(\varphi)|$...
- ... but we have not been able to show that all algorithms are bound to take time exponential in $|Atoms(\varphi)|$.
- P vs. NP problem
  `https://www.claymath.org/millennium-problems/p-vs-np-problem`

Normal forms - Why?

1. Formulas can be big (several GBs)
2. Representing and manipulating arbitrary formulas is difficult
3. In order to simplify their management, it is convenient to simplify their representation to some normal form:
    1. Negative Normal Form (NNF)
    2. Conjunctive Normal Form (CNF)

Negative Normal Form (NNF)

Negative Normal Form

A formula is in NNF if negation is allowed only over atoms and $\land$, $\lor$ and $\neg$ are the only allowed boolean connectives.

Example:

$\neg(A \lor B)$ ✓ $(A \lor B) \land \neg C$

Negative Normal Form (NNF)

Every φ can be reduced into NNF:

1. by rewriting $\leftrightarrow$: $$\begin{aligned}\varphi_1 \leftrightarrow \varphi_2 &\equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \\ &\equiv (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)\end{aligned}$$
2. by rewriting $\rightarrow$: $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
3. and applying De Morgan's rules: $$\begin{aligned}\neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) &\equiv \neg\varphi_1 \wedge \neg\varphi_2\end{aligned}$$

Negative Normal Form (NNF)

Every φ can be reduced into NNF:

1. by rewriting $\leftrightarrow$: $$\begin{aligned} \varphi_1 \leftrightarrow \varphi_2 &\equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \\ &\equiv (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2) \end{aligned}$$
2. by rewriting $\rightarrow$: $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
3. and applying De Morgan's rules: $$\begin{aligned} \neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) &\equiv \neg\varphi_1 \wedge \neg\varphi_2 \end{aligned}$$

▷ Preserves the equivalence of formulas.

Exercise

Convert the following formula to NNF:

$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$

Exercise

Convert the following formula to NNF:

$$
\begin{gathered}
(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4) \\
\Downarrow \\
((((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \wedge \\
(((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow ((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1))))
\end{gathered}
$$

Exercise

Convert the following formula to NNF:

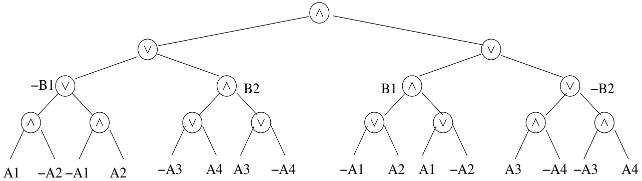
$$
\begin{gathered}
(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4) \\
\Downarrow \\
((((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \wedge \\
(((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow ((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)))) \\
\Downarrow \\
((\neg((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee ((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4))) \wedge \\
(((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee \neg((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4))))
\end{gathered}
$$

Exercise

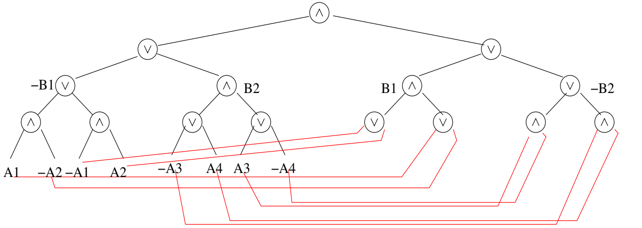
Convert the following formula to NNF:

$$
\begin{gathered}
(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4) \\
\Downarrow \\
((((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3))\wedge \\
(((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow ((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)))) \\
\Downarrow \\
((\neg((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee ((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4)))\wedge \\
(((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee \neg((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4)))) \\
\Downarrow \\
((((A_1 \wedge \neg A_2) \vee (\neg A_1 \wedge A_2)) \vee ((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4)))\wedge \\
(((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee ((A_3 \wedge \neg A_4) \vee (\neg A_3 \wedge A_4))))
\end{gathered}
$$

Exercise (cont.d)



Tree Representation



DAG Representation

Note For each non-literal subformula φ, φ and $\neg\varphi$ have different representations $\Longrightarrow$they are not shared.

Conjunctive Normal Form (CNF)

Conjuctive Normal Form

A formula is in Conjunctive Normal Form if it is a conjunction of disjunctions of literals:

$$\bigwedge_i \left(\bigvee_j l_{ij}\right)$$

- ▷ l_{ij} is the j-th literal of the i-th **clause**.
- ▷ A CNF can be represented as a set of sets of literals.

Example: ✓ $(A \vee B) \wedge C$ $\quad$ $(A \wedge B) \vee C$

Example: ✓ $\{\{A, B\}, \{C\}\}$

Questions: Assume a CNF φ is represented as a set of sets of literals.

1. What if the empty clause is in φ, i.e., if $\emptyset \in \varphi$?
2. What if φ is the empty set of clauses, i.e., if $\varphi = \emptyset$?

CNF Conversion

Every formula φ can be reduced into CNF by:

1. converting it into NNF;
2. applying recursively DeMorgan's Rule.

Example:

$$\begin{array}{cc}
A \rightarrow B \wedge C & \\
\Downarrow & \text{NNF conversion} \\
\neg A \vee (B \wedge C) & \\
\Downarrow & \text{Distributivity} \\
(\neg A \vee B) \wedge (\neg A \vee C) & \\
\Downarrow & \text{Represented as a set of sets} \\
\{\{\neg A, B\}, \{\neg A, C\}\} &
\end{array}$$

- ▷ Worst-case exponential, rarely used in practice.
- ▷ $Atoms(CNF(\varphi)) = Atoms(\varphi)$.
- ▷ $CNF(\varphi)$ is equivalent to φ.

Improving CNF conversion: Tseitin's encoding

Every φ can also be transformed into an equi-satisfiable CNF formula $T(\varphi)$ with only a linear increase in the size of the formula.

How? Tseitin's encoding

- Add one new variable for every logical gate in φ
- Constrain new variables to be equal to the gate they represent

Start from φ, apply recursively the following rules:

$$\begin{array}{lcll} \varphi & \Longrightarrow & \varphi[B/(l_i \vee l_j)] & \wedge \quad CNF(B \leftrightarrow (l_i \vee l_j)) \\ \varphi & \Longrightarrow & \varphi[B/(l_i \wedge l_j)] & \wedge \quad CNF(B \leftrightarrow (l_i \wedge l_j)) \\ \varphi & \Longrightarrow & \varphi[B/(l_i \leftrightarrow l_j)] & \wedge \quad CNF(B \leftrightarrow (l_i \leftrightarrow l_j)) \end{array}$$

l_i, l_j being literals and B being a "new" variable.

Tseitin's encoding (cont.d)

What is CNF(...)? Your turn:

$$
\begin{aligned}
CNF(B \leftrightarrow (l_i \vee l_j)) &\quad \leftrightarrow \\
CNF(B \leftrightarrow (l_i \wedge l_j)) &\quad \leftrightarrow \\
CNF(B \leftrightarrow (l_i \leftrightarrow l_j)) &\quad \leftrightarrow
\end{aligned}
$$

Tseitin's encoding (cont.d)

What is CNF(...)? Your turn:

$$CNF(B \leftrightarrow (l_i \vee l_j)) \quad \leftrightarrow$$

$$CNF(B \leftrightarrow (l_i \wedge l_j)) \quad \leftrightarrow$$

$$CNF(B \leftrightarrow (l_i \leftrightarrow l_j)) \quad \leftrightarrow$$

Some interesting facts...

- ▷ Worst-case linear.
- ▷ $Atoms(\varphi) \subseteq Atoms(T(\varphi))$.
- ▷ $T(\varphi)$ is equi-satisfiable w.r.t. φ.
- ▷ Non-canonical.
- ▷ More used in practice.

Exercise

Using Tseitin's encoding, convert the following formula to CNF:

$$A \rightarrow B \wedge C$$

Exercise

Using Tseitin's encoding, convert the following formula to CNF:

$$A \rightarrow B \wedge C$$

- assign B_2 to the AND gate ($B \wedge C$)
- assign B_1 to the IMPLICATION gate ($A \rightarrow B_2$)

Exercise

Using Tseitin's encoding, convert the following formula to CNF:

$$A \rightarrow B \land C$$

- assign B_2 to the AND gate ($B \land C$)
- assign B_1 to the IMPLICATION gate ($A \rightarrow B_2$)
- $B_1 \leftrightarrow (\neg A \lor B_2)$
- $B_2 \leftrightarrow (B \land C)$

Exercise

Using Tseitin's encoding, convert the following formula to CNF:

$$A \rightarrow B \land C$$

- assign B_2 to the AND gate ($B \land C$)
- assign B_1 to the IMPLICATION gate ($A \rightarrow B_2$)
- $B_1 \leftrightarrow (\neg A \lor B_2)$
- $B_2 \leftrightarrow (B \land C)$
- convert equivalences to CNF

Exercise

Using Tseitin's encoding, convert the following formula to CNF:

$$A \to B \wedge C$$

- assign B_2 to the AND gate $(B \wedge C)$
- assign B_1 to the IMPLICATION gate $(A \to B_2)$
- $B_1 \leftrightarrow (\neg A \vee B_2)$
- $B_2 \leftrightarrow (B \wedge C)$
- convert equivalences to CNF

$$(B_1) \wedge (\neg B_1 \vee \neg A \vee B_2) \wedge (B_1 \vee A) \wedge (B_1 \vee \neg B_2) \wedge$$
$$(\neg B_2 \vee B) \wedge (\neg B_2 \vee C) \wedge (B_2 \vee \neg B \vee \neg C)$$
$$\Downarrow \text{ as a set of sets}$$
$$\{\{B_1\}, \{\neg B_1, \neg A, B_2\}, \{B_1, A\}, \{B_1, \neg B_2\},$$
$$\{\neg B_2, B\}, \{\neg B_2, C\}, \{B_2, \neg B, \neg C\}\}$$

Optimizations

Can we reduce the size of the resulting formula and the number of additional variables?

- ▷ As in the previous case, applying instead the rules:

$$\begin{array}{lllll}
\varphi & \Longrightarrow & \varphi[B/(l_i \vee l_j)] & \wedge\, CNF(B \rightarrow (l_i \vee l_j)) & \text{if } (l_i \vee l_j) \text{ pos.} \\
\varphi & \Longrightarrow & \varphi[B/(l_i \vee l_j)] & \wedge\, CNF((l_i \vee l_j) \rightarrow B) & \text{if } (l_i \vee l_j) \text{ neg.} \\
\varphi & \Longrightarrow & \varphi[B/(l_i \wedge l_j)] & \wedge\, CNF(B \rightarrow (l_i \wedge l_j)) & \text{if } (l_i \wedge l_j) \text{ pos.} \\
\varphi & \Longrightarrow & \varphi[B/(l_i \wedge l_j)] & \wedge\, CNF((l_i \wedge l_j) \rightarrow B) & \text{if } (l_i \wedge l_j) \text{ neg.} \\
\varphi & \Longrightarrow & \varphi[B/(l_i \leftrightarrow l_j)] & \wedge\, CNF(B \rightarrow (l_i \leftrightarrow l_j)) & \text{if } (l_i \leftrightarrow l_j) \text{ pos.} \\
\varphi & \Longrightarrow & \varphi[B/(l_i \leftrightarrow l_j)] & \wedge\, CNF((l_i \leftrightarrow l_j) \rightarrow B) & \text{if } (l_i \leftrightarrow l_j) \text{ neg.}
\end{array}$$

- ▷ Smaller in size:

$$\begin{array}{ll}
CNF(B \rightarrow (l_i \vee l_j)) & = (\neg B \vee l_i \vee l_j) \\
CNF(((l_i \vee l_j) \rightarrow B)) & = (\neg l_i \vee B) \wedge (\neg l_j \vee B)
\end{array}$$

Exercise

Consider the formula

$$A \to B \wedge C$$

and its CNF conversion of the previous exercise.

Repeat the conversion using the optimized rules. Comment on the resulting formula (size, additional variables...)

Further optimizations

- Do not apply T when not necessary:
  - e.g. $T(\varphi_1 \wedge \varphi_2) \Longrightarrow T(\varphi_1) \wedge \varphi_2$, if φ_2 already in CNF
- Apply Demorgan's rules where it is more effective:
  - $T(\varphi_1 \wedge (A \rightarrow (B \wedge C))) \Longrightarrow T(\varphi_1) \wedge (\neg A \vee B) \wedge (\neg A \vee C)$
- Exploit the associativity of $\wedge$'s and $\vee$'s
  :
  - $... \underbrace{(A_1 \vee (A_2 \vee A_3))}_{B} ... \Longrightarrow ... CNF(B \leftrightarrow (A_1 \vee A_2 \vee A_3))...$
- before applying CNF_{label}, rewrite the initial formula so that to maximize the sharing of subformulas
- ...

Agenda - Propositional Logic

Truth Tables

▷ Exhaustive evaluation of all assignments till a satisfying one is found:

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_1 \leftrightarrow \varphi_2$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

- It is possible to generate one assignement after the other
- Given an assignment, evaluating a formula can be done in polynomial time

▷ Does not require any normal form

▷ Requires polynomial space

▷ If the formula is unsatisfiable, requires exponential time

▷ Never used in practice.

Exercise

Use the truth tables method to determine whether

$$\varphi := (\neg A \lor B) \land (B \to \neg C \land \neg A) \land (A \lor C)$$

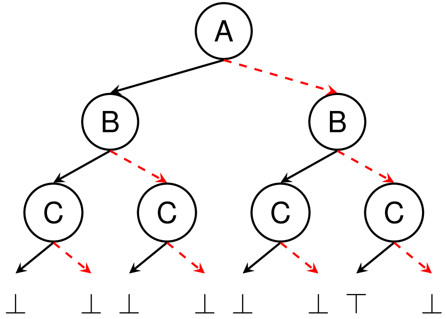
is satisfiable.

A	B	C	$\neg A \lor B$	$\neg C \land \neg A$	$B \to \neg C \land \neg A$	$(A \lor C)$	φ
$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$

There exists an interpretation satisfying φ, thus φ is satisfiable.
The time taken to determine its satisfiabilty critically depends on the way assignments are generated and evaluated.

Exercise, cont.d

Tree representation of the truth table (solid, black arrow: variable assigned $\top$; dashed, red arrow: variable assigned $\perp$).



Truth Tables: alternative

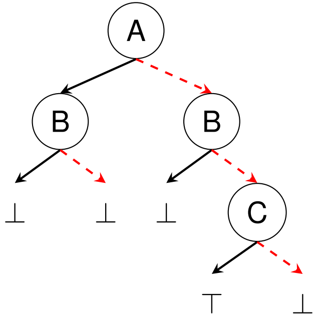
- Instead of evaluating the formula at the leaves,
- Simplify the formula at each node substituting the variable with its truth value and then using truth tables.

$$\varphi := (\neg A \vee B) \wedge (B \rightarrow \neg C \wedge \neg A) \wedge (A \vee C)$$

Truth Tables: alternative

- Instead of evaluating the formula at the leaves,
- Simplify the formula at each node substituting the variable with its truth value and then using truth tables.

$$\varphi := (\neg A \vee B) \wedge (B \rightarrow \neg C \wedge \neg A) \wedge (A \vee C)$$



Agenda - Propositional Logic

Simplification rules for formulas in CNF

Assume φ is in CNF, i.e.,

$$\varphi := \bigwedge_{i=1}^{n} CL_i \quad \text{with} \quad CL_i := (\bigvee_{j=1}^{m_j} l_{ij})$$

or

$$\varphi := \{CL_i : 1 \leq i \leq n\} \quad \text{with} \quad CL_i := \{l_{ij} : 1 \leq j \leq m_j\}$$

Facts:

1. If for some variable V, $\{V, \neg V\} \subseteq CL$, we can remove CL from φ and get an equivalent formula.
2. If CL_1 and CL_2 are two clauses in φ with $CL_1 \subseteq CL_2$, we can remove CL_2 from φ and get an equivalent formula (clause subsumption).

Simplification rules for formulas in CNF

Consider a set of clauses φ and a variable V.

- φ_V^+ is the set of clauses $CL \in \varphi$ such that $V \in CL$.
- φ_V^- is the set of clauses $CL \in \varphi$ such that $\neg V \in CL$.
- *assign*(V, φ) is the set of clauses obtained by
  1. removing all the clauses in φ_V^+, and
  2. replacing every clause $CL \in \varphi_V^-$ with $CL \setminus \{\neg V\}$

  Analogously for *assign*$(\neg V, \varphi)$.

Simplification rules for formulas in CNF

Facts: For any variable V, and set of clauses φ

1. If $\{V\} \in \varphi_V^+$, we say that $\{V\}$ is a unit clause and
   1. φ and *assign*(V, φ) are equisatisfiable,
   2. φ and *assign*$(V, \varphi) \cup \{V\}$ are equivalent.

   Analogouly if $\{\neg V\} \in \varphi_V^-$.
2. If either $\varphi_V^+ = \emptyset$ or $\varphi_V^- = \emptyset$ then we can remove $\varphi_V^+ \cup \varphi_V^-$ from φ and get an equisatisfiable formula (pure literal propagation).

Resolution rule

Idea

To satisfy two clauses CL_1 with $A \in CL_1$ and CL_2 with $\neg A \in CL_2$ either the “rest” of CL_1 or the “rest” of CL_2 must be satisfied!

Resolution rule

Idea

To satisfy two clauses CL_1 with $A \in CL_1$ and CL_2 with $\neg A \in CL_2$ either the “rest” of CL_1 or the “rest” of CL_2 must be satisfied!

Why?

If A is true, then a literal other than $\neg A$ in CL_2 must be true (symmetric for A false).

Resolution rule

Resolution of a pair of clauses with exactly one incompatible variable:

$$\frac{(\overbrace{A}^{\text{resolution variable}} \vee \overbrace{l_1 \vee \ldots \vee l_m}^{\textit{resolving clause}}) \qquad (\overbrace{\neg A}^{\text{resolution variable}} \vee \overbrace{l'_1 \vee \ldots \vee l'_n}^{\textit{resolving clause}})}{(\underbrace{l_1 \vee \ldots \vee l_m \vee l'_1 \vee \ldots \vee l'_n}_{\textit{resolvent}})}$$

Example:

$$\frac{(A \vee B \vee C \vee D \vee E) \qquad (A \vee B \vee \neg C \vee F)}{(A \vee B \vee D \vee E \vee F)}$$

Many standard inference rules are particular cases of resolution:

$$\frac{A \to B \quad B \to C}{A \to C}\ (\textit{Transit.}) \qquad \frac{A \quad A \to B}{B}\ (\textit{M.Ponens}) \qquad \frac{\neg B \quad A \to B}{\neg A}\ (\textit{M.Tollens})$$

Question: What is the result of the resolution rule applied to two clauses CL_1 and CL_2 with $\{A, B\} \subseteq CL_1$ and $\{\neg A, \neg B\} \subseteq CL_2$?

Variable Elimination

For two clauses $CL_1 \in \varphi_V^+$ and $CL_2 \in \varphi_V^-$,

$$Res(CL_1, CL_2) = CL_1 \cup CL_2 \setminus \{V, \neg V\}$$

i.e., $Res(CL_1, CL_2)$ is the resolution between CL_1 and CL_2.

$$Resolve(\varphi_V^+, \varphi_V^-) = \{Res(CL_1, CL_2) : CL_1 \in \varphi_V^+, CL_2 \in \varphi_V^-\}$$

i.e., $Resolve(\varphi_V^+, \varphi_V^-)$ is the set of clauses obtained by resolving all the clauses in φ_V^+ with all the clauses in φ_V^-.

$$\varphi_V = \varphi \cup Resolve(\varphi_V^+, \varphi_V^-) \setminus (\varphi_V^+ \cup \varphi_V^-)$$

Variable Elimination

Facts: For any variable V, and set of clauses φ

1. φ and φ_V are equi-satisfiable.
2. φ_V contains one less variable and possibly quadratically more clauses than φ .

Resolution algorithm: pseudo-code

```
function RESOLUTION($\varphi$):
    if $\emptyset \in \varphi$:                                  /* unsat */
        then return *False*;
    if $\varphi = \emptyset$:                                    /* sat */
        then return *True*;
    if a unit clause $\{l\}$ is in $\varphi$:                    /* unit */
        then $\varphi := assign(l, \varphi)$);
        return RESOLUTION($\varphi$)
    if either $\varphi_V^+ = \emptyset$ or $\varphi_V^- = \emptyset$:    /* pure */
        then $\varphi := \varphi \setminus (\varphi_V^+ \cup \varphi_V^-)$
        return RESOLUTION($\varphi$)
    *V := select-variable*($\varphi$);                           /* resolve */
    $\varphi = \varphi_V$;
    return RESOLUTION($\varphi$).
```

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$$

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$$

Solution

$$\{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$$
$$\Downarrow$$
$$\{\{B\}, \cancel{\{B, \neg B\}}, \cancel{\{\neg B, B\}}, \{\neg B\}\}$$
$$\Downarrow$$
$$\{\{\}\}$$

φ is unsat

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{B, C, D\}, \{B, \neg C, \neg D\}, \{\neg B, D\}\}$$

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{B, C, D\}, \{B, \neg C, \neg D\}, \{\neg B, D\}\}$$

Solution

$$\{\{B, C, D\}, \{B, \neg C, \neg D\}, \{\neg B, D\}\}$$
$$\Downarrow$$
$$\{\{C, D\}, \cancel{\{\neg C, \neg D, D\}}\}$$
$$\Downarrow$$
$$\{\}$$

φ is SAT

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{A, B\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\}$$

Exercise

Use RESOLUTION to determine if the following set of clauses is satisfiable

$$\varphi = \{\{A, B\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\}$$

Solution

$$\begin{gathered}
\{\{A, B\}, \{A, \neg B\}, \{\neg A, C\}, \{\neg A, \neg C\}\} \\
\Downarrow \\
\{\{A\}, \{\neg A, C\}, \{\neg A, \neg C\}\} \\
\Downarrow \\
\{\{C\}, \{\neg C\}\} \\
\Downarrow \\
\{\{\}\}
\end{gathered}$$

φ is UNSAT

Resolution – summary

- Requires CNF
- May require exponential space
- Not very much used in Boolean reasoning
- Often used together with other procedures (e.g., DPLL).

Davis-Putnam-Logemann-Loveland (DPLL)

- Operates on Boolean formulas in CNF;
- Tries to build an interpretation μ satisfying φ;
- If μ cannot be constructed, reports unsatisfiability

DPLL algorithm: pseudo-code

```
function DPLL($\varphi$):
    if $\emptyset \in \varphi$:                              /* unsat */
        then return False;
    if $\varphi = \emptyset$:                                /* sat */
        then return True;
    if a unit clause $\{l\}$ is in $\varphi$:                /* unit */
        then $\varphi := assign(l, \varphi)$);
        return DPLL($\varphi$)
    if either $\varphi_V^+ = \emptyset$ or $\varphi_V^- = \emptyset$:    /* pure */
        then $\varphi := \varphi \setminus (\varphi_V^+ \cup \varphi_V^-)$
        return DPLL($\varphi$)
    V := select-variable($\varphi$);                         /* branch */
    return DPLL($assign(V, \varphi)$) or
           DPLL($assign(\neg V, \varphi)$).
```

DPLL – example

Consider the formula

$$\varphi := (\neg A \lor B \lor C) \land (\neg B \lor C) \land (\neg B \lor \neg C) \land (A \lor \neg B \lor \neg C)$$

On the first level of recursion, DPLL must branch.

Assume B is the variable selected for branching.

Branch on B:

- $\varphi[B \mapsto \top] : (C) \land (\neg C) \land (A \lor \neg C)$
  - unit propagation closes this branch.
- $\varphi[B \mapsto \bot] : (\neg A \lor C)$
  - $\varphi^-_C = \emptyset$ and then $\varphi[C \mapsto \top]$ which generates the empty set of clauses.

Satisfying assignment: $\mu : \{A \mapsto \bot, B \mapsto \bot, C \mapsto \top\}$.

DPLL – example (cont.d)

$$\begin{tikzcd}
 & \varphi \arrow[dl, dash, "B \mapsto \top"'] \arrow[dr, dash, "B \mapsto \bot"] & \\
\{\{C), \{\neg C), \{A, \neg C\}\} \arrow[d, dash, "C \mapsto \top"'] & & \{\{\neg A, C\}\} \arrow[d, dash, "C \mapsto \top"] \\
\{\{\}\} & & \{\} \\
 & & \mu : \{B \mapsto \bot, C \mapsto \top\}
\end{tikzcd}$$

DPLL – example (cont.d)

$$\begin{tikzcd}
 & \varphi \arrow[dl, dash, "B \mapsto \top"'] \arrow[dr, dash, "B \mapsto \bot"] & \\
\{\{C), \{\neg C), \{A, \neg C\}\} \arrow[d, dash, "C \mapsto \top"'] & & \{\{\neg A, C\}\} \arrow[d, dash, "C \mapsto \top"] \\
\{\{\}\} & & \{\}
\end{tikzcd}$$

$$\mu : \{B \mapsto \bot, C \mapsto \top\}$$

Question: What about the truth value of A in μ?

DPLL – summary

- Handles CNF formulas (non-CNF variant known);
- Branches on truth values of variables
- Postpones branching as much as possible;
- Requires polynomial space;
- Currently at the basis of the most efficient SAT algorithms;
- Very efficient implementations exist.

Agenda - Propositional Logic

Ordered Binary Decision Diagrams (OBDDs)

Graph-based data structure for manipulating Boolean formulas:

- “If-then-else” binary DAGs with two leaves: 1 and 0
- Paths leading to 1 represent models
- Paths leading to 0 represent counter-models

Canonicity

The OBDD representation is canonical (unique) for a particular function and variable order.

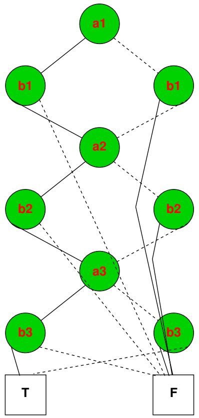
Checking for satisfiability can be done in constant time for a given OBDD...

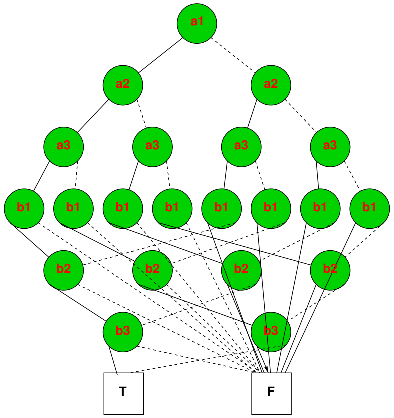
... no free lunch: building an OBDD can take exponential space and time!

Exponentiality of OBDDs

- The size of OBDDs may grow exponentially wrt. the number of variables in worst-case
- Consequence of the canonicity of OBDDs (unless P = co-NP)
- Example: there exist no polynomial-size OBDD representing the electronic circuit of a bitwise multiplier
- the size of intermediate OBDDs may be bigger than that of the final one (e.g., inconsistent formula)

OBDD -





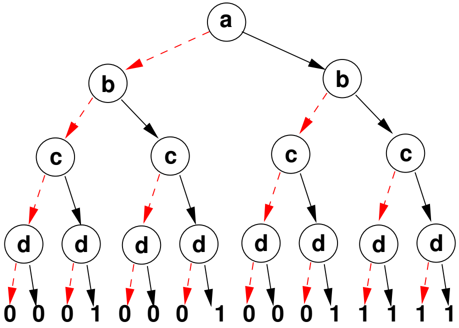
OBDDs of $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$ with different variable orderings

$\triangleright$

Ordered Decision Trees

- represent a Boolean formula;
- variable order is maintained along each path of the tree;
- every path in the tree corresponds to an assignment.

Example: $\varphi = (a \wedge b) \vee (c \wedge d)$



From Ordered Decision Trees to OBDDs: reductions

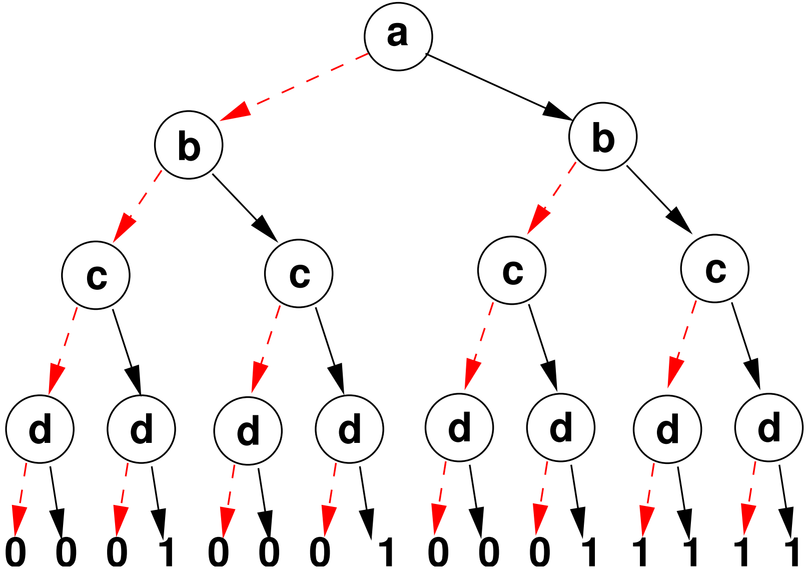
Binary decision tree is not any better than an explicit truth table when it comes to space consumption...

Can we improve?

Recursive applications of the following reductions:

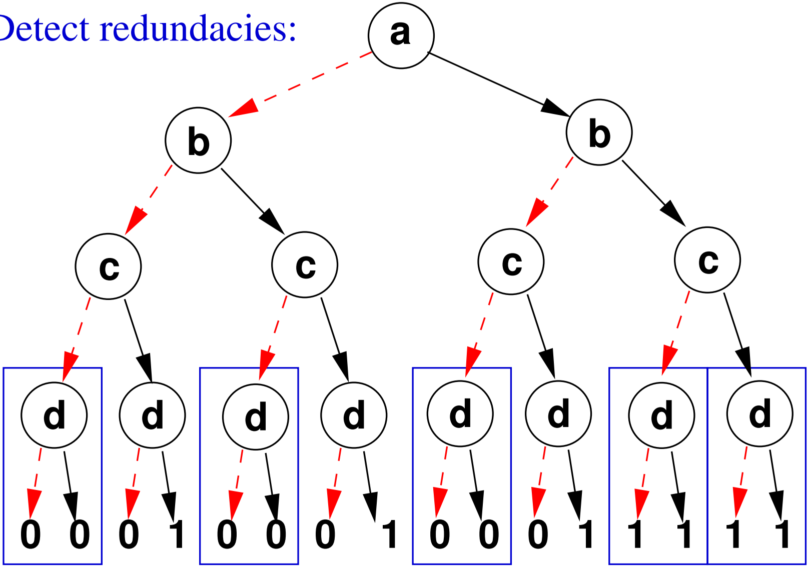
- share subnodes: point to the same occurrence of a subtree
- remove redundancies: nodes with same left and right children can be eliminated

Reduction: example



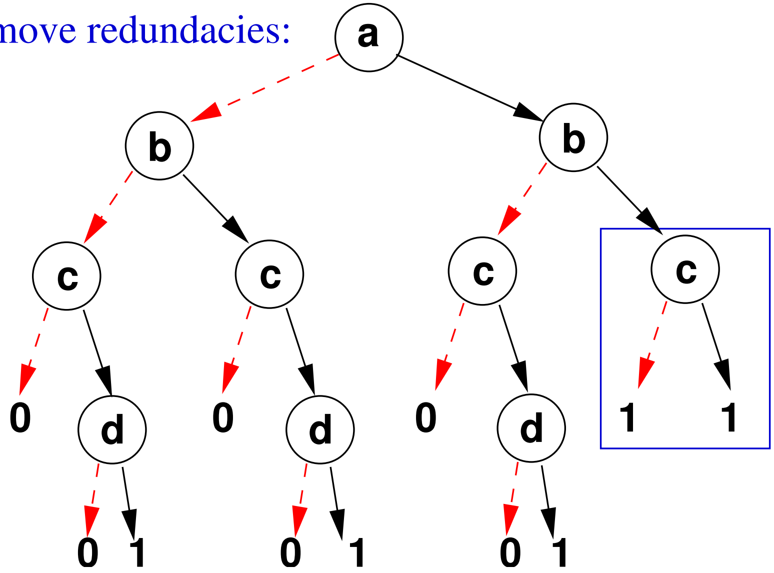
Reduction: example (cont.d)

Detect redundacies:



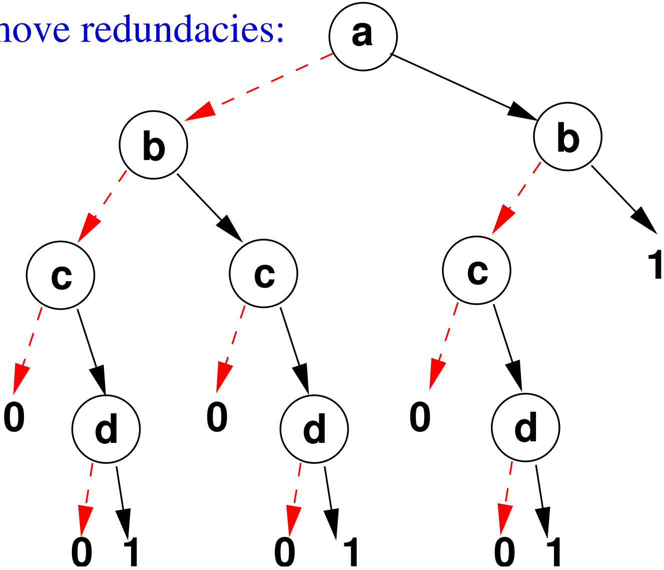
Reduction: example (cont.d)

Remove redundacies:



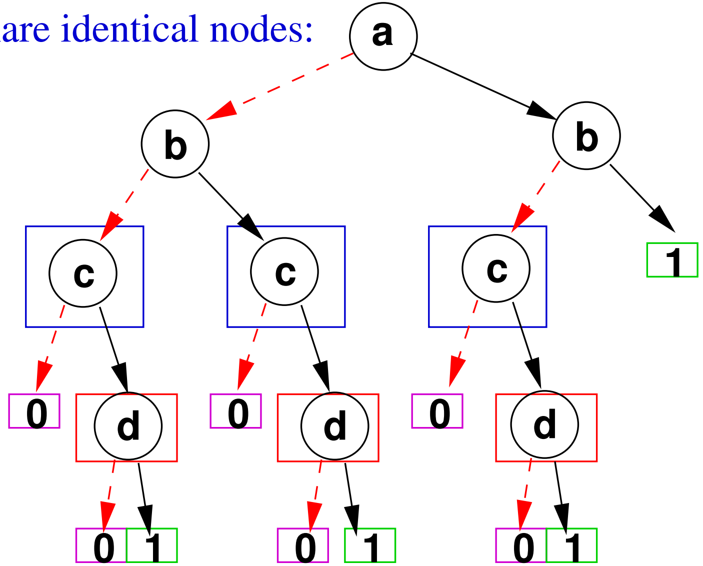
Reduction: example (cont.d)

Remove redundacies:



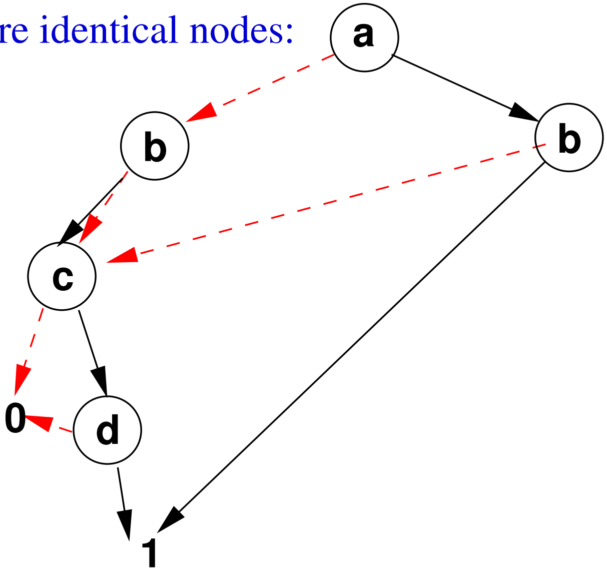
Reduction: example (cont.d)

Share identical nodes:



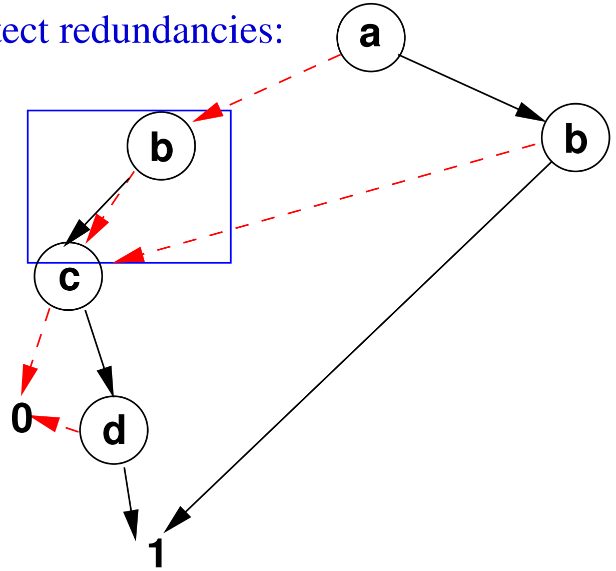
Reduction: example (cont.d)

Share identical nodes:



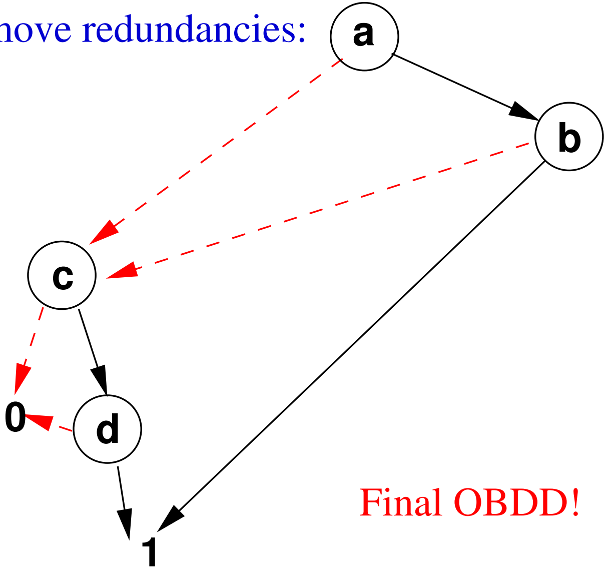
Reduction: example (cont.d)

Detect redundancies:



Reduction: example (cont.)

Remove redundancies:



Final OBDD!

Recursive structure of an OBDD

Turning a binary decision tree into an OBDD helps understanding the reduction rules...

... but we don't want to build the tree (exponential size)!

Recursive structure of an OBDD

Turning a binary decision tree into an OBDD helps understanding the reduction rules...

... but we don't want to build the tree (exponential size)!

Given a formula φ, we build its OBDD recursively and incrementally from the OBDD of its subexpressions:

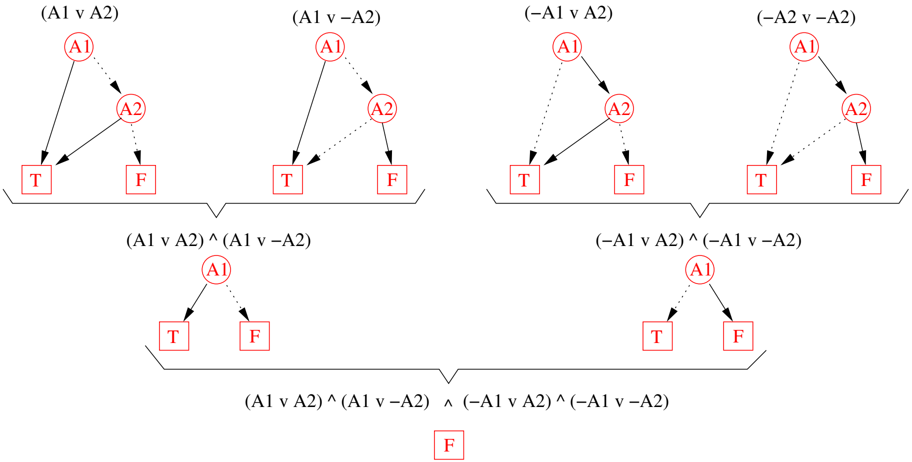
- $\mathit{obdd\_build}(\top, \{...\}) := 1$,
- $\mathit{obdd\_build}(\bot, \{...\}) := 0$,
- $\mathit{obdd\_build}((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $\mathit{obdd\_apply}(\neg, \mathit{obdd\_build}(\varphi, \{A_1, ..., A_n\}))$, // swap the 0-1 leaves
- $\mathit{obdd\_build}((\varphi_1\ op\ \varphi_2), \{A_1, ..., A_n\}) :=$

```
reduce(
  obdd_merge(   op,
                obdd_build(φ1, {A1, ..., An}),
                obdd_build(φ2, {A1, ..., An}),
                {A1, ..., An}
))
```

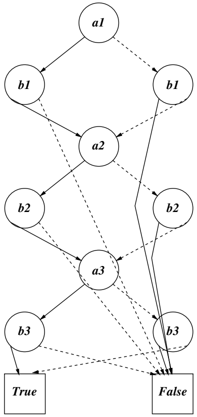
$op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

OBBD incremental building – example

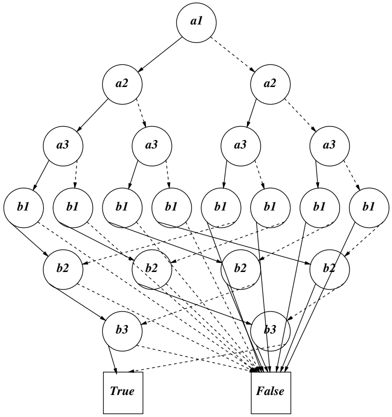
$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$



Variable ordering is critical!



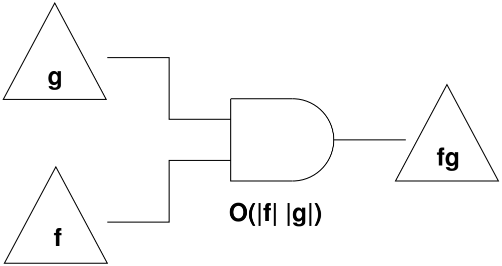
Linear size



Exponential size

Useful Operations over OBDDs

- the equivalence check between two OBDDs is simple
  - are they the same OBDD? ($\Longrightarrow$constant time)
- the size of a Boolean composition is up to the product of the size of the operands: $|f\ op\ g| = O(|f| \cdot |g|)$



(but typically much smaller on average).

Boolean quantification

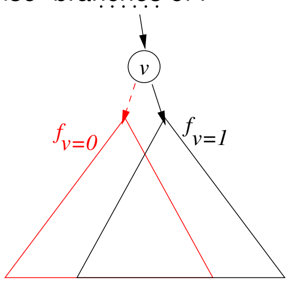
- If v is a Boolean variable, then

$$\begin{aligned} \exists v.f &:= f|_{v=0} \vee f|_{v=1} \\ \forall v.f &:= f|_{v=0} \wedge f|_{v=1} \end{aligned}$$

- Multi-variable quantification: $\exists(w_1, \ldots, w_n).f \ := \ \exists w_1 \ldots \exists w_n.f$
- Example: $\exists(b, c).((a \wedge b) \vee (c \wedge d)) \ = \ a \vee d$
- naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulas
- OBDDs handle very efficiently quantification operations

OBDDs and Boolean quantification

- OBDDs handle quantification operations rather efficiently
  - if f is a sub-OBDD labeled by variable v, then $f|_{v=1}$ and $f|_{v=0}$ are the "then" and "else" branches of f



$\Longrightarrow$lots of sharing of subformulas!

OBDDs – summary

- Factorize common parts of the search tree (DAG);
- Require setting a variable ordering a priori (critical!);
- Canonical representation of a Boolean formula;
- Once built, logical operations (satisfiability, validity, equivalence) immediate;
- Represents all models and counter-models of the formula;
- Require exponential space in worst-case;
- Very efficient for some practical problems (circuits, symbolic model checking).

Agenda - Propositional Logic

Incomplete SAT techniques: GSAT, WSAT

- Hill-Climbing techniques: GSAT, WSAT
- looks for a complete assignment;
- starts from a random assignment;
- Greedy search: looks for a better “neighbor” assignment
- Avoid local minima: restart & random walk

The GSAT algorithm

```
function GSAT($\varphi$)
    for $i := 1$ to Max-tries do:
        $\mu :=$ rand-assign($\varphi$);
        for $j := 1$ to Max-flips do:
            if ($score(\varphi, \mu) = 0$):
                then return True;
                else Best-flips := hill-climb($\varphi, \mu$);
                    $A_i$ := rand-pick(Best-flips);
                    $\mu$ := flip($A_i, \mu$);
        end
    end
    return “no satisfying assignment found”.
```

The WalkSAT algorithm

```
WALKSAT($\varphi$,MAX-STEPS,MAX-TRIES, select())
1  for $i \leftarrow 1$ to MAX-TRIES
2  do $\mu \leftarrow$ a randomly generated truth assignment;
3      for $j \leftarrow 1$ to MAX-STEPS
4      do if $\mu$ satisfies $\varphi$
5            then return $\mu$;
6            else $C \leftarrow$ randomly selected clause unsatisfied under $\mu$;
7                 $x \leftarrow$ variable selected from $C$ according to heuristic
8                 $\mu \leftarrow \mu$ with $x$ flipped;
9  return error "no solution found"
```

GSAT & WSAT– summary

- Handle only CNF formulas;
- Incomplete;
- Extremely efficient for some (satisfiable) problems;
- Require polynomial space;
- Used in Artificial Intelligence (e.g., planning).

Beyond propositional logic

- Logics:
  1. Many-Valued Logic
  2. Quantified Boolean Formulas
  3. Modal and Temporal Logics
  4. Quantifier free predicate logic
- Reasoning techniques:
  1. SAT-based (or SMT) solvers
- Reasoning Tasks:
  1. Prime Implicants computation
  2. Unsatisfiable Core computation
  3. Maximum Satisfiability problems
  4. MAX-ONE problems