Satisfiability Checking
Propositional Logic

Prof. Dr. Erika Ábrahám

Theory of Hybrid Systems
Informatik 2

WS 11/12

Propositional logic

The slides are partly taken from:

www.decision-procedures.org/slides/

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Abstract syntax of propositional logic

Propositional logic is the quantifier-free fragment of the first-order theory with $\Sigma = \{\}$ without axioms.

Abstract grammar of well-formed propositional formulae:

$$\varphi \; := \; a \;\mid\; (\neg\varphi) \;\mid\; (\varphi \wedge \varphi)$$

with $a \in \texttt{Prop}$ and $\texttt{Prop}$ a set of propositions (Boolean variables).

Syntactic sugar:

$$\begin{array}{ccccccl}
 & & \bot & & & := & (a \wedge \neg a) \\
 & & \top & & & := & (a \vee \neg a) \\
(& \varphi_1 & \vee & \varphi_2 &) & := & \neg((\neg\varphi_1) \wedge (\neg\varphi_2)) \\
(& \varphi_1 & \rightarrow & \varphi_2 &) & := & ((\neg\varphi_1) \vee \varphi_2) \\
(& \varphi_1 & \leftrightarrow & \varphi_2 &) & := & ((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)) \\
(& \varphi_1 & \bigoplus & \varphi_2 &) & := & (\varphi_1 \leftrightarrow (\neg\varphi_2))
\end{array}$$

Formulae

- Examples of well-formed formulae:
  - $(\neg a)$
  - $(\neg(\neg a))$
  - $(a \wedge (b \wedge c))$
  - $(a \rightarrow (b \rightarrow c))$
- Remember: we omit parenthesis whenever we may restore them through operator precedence:

binds stronger

$$\longleftarrow$$

$$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Semantics: Assignments

Structures for predicate logic:

- The domain is $\mathbb{B} = \{0, 1\}$.
- Since there are no constants, function or predicate symbols, the interpretation just assigns Boolean values to the variables:

$$\alpha : \texttt{Prop} \rightarrow \{0, 1\}$$

We call these special interpretations assignments and use *Ass* to denote the set of all assignments.
Example: $\texttt{Prop} = \{a, b\}, \alpha(a) = 0, \alpha(b) = 1$

Equivalently, we can see an assignment α as a set of variables ($\alpha \in 2^{\texttt{Prop}}$), defining the variables from the set to be true and the others false.
Example: $\texttt{Prop} = \{a, b\}, \alpha = \{b\}$

An assignment can also be seen as being of type $\alpha \in \{0, 1\}^{\texttt{Prop}}$, if we have an order on the propositions.
Example: $\texttt{Prop} = \{a, b\}, \alpha = \{01\}$

Only the projected assignment matters...

- Let $\alpha_1, \alpha_2 \in$ *Ass* and $\varphi \in$ Formula.
- AP(φ) - the atomic propositions in φ.
- Clearly AP(φ) $\subseteq$ `Prop`.
- Lemma: if $\alpha_1|_{\mathrm{AP}(\varphi)} = \alpha_2|_{\mathrm{AP}(\varphi)}$, then

Projection

$$(\alpha_1 \textit{ satisfies } \varphi) \quad \text{iff} \quad (\alpha_2 \textit{ satisfies } \varphi)$$

- We will assume, for simplicity, that $\texttt{Prop} = \mathrm{AP}(\varphi)$.

Semantics I: Truth tables

- Truth tables define the semantics (=meaning) of the operators. They can be used to define the semantics of formulae inductively over their structure.
- Convention: 0 = false, 1 = true

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \bigoplus q$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	0	0	1
1	1	0	1	1	1	1	0

Each possible assignment is covered by a line of the truth table.
α is a model for φ iff in the line for α and the column for φ the entry is 1.

Q: How many binary operators can we define that have different semantics?
A: 16

Example

- Let φ be defined as $(a \vee (b \rightarrow c))$.
- Let $\alpha : \{a, b, c\} \rightarrow \{0, 1\}$ be an assignment with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.

- Q: Does α satisfy φ?
- A1: Compute with truth table:

a	b	c	$b \rightarrow c$	$a \vee (b \rightarrow c)$
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

Semantics II: Satisfaction relation

Satisfaction relation: $\models \subseteq Ass \times \text{Formula}$

Instead of $(\alpha, \varphi) \in \models$ we write $\alpha \models \varphi$ and say that

- α satisfies φ or
- φ holds for α or
- α is a model of φ.

$\models$ is defined recursively:

$$
\begin{array}{lll}
\alpha & \models p & \text{iff } \alpha(p) = \text{true} \\
\alpha & \models \neg\varphi & \text{iff } \alpha \not\models \varphi \\
\alpha & \models \varphi_1 \wedge \varphi_2 & \text{iff } \alpha \models \varphi_1 \text{ and } \alpha \models \varphi_2 \\
\alpha & \models \varphi_1 \vee \varphi_2 & \text{iff } \alpha \models \varphi_1 \text{ or } \alpha \models \varphi_2 \\
\alpha & \models \varphi_1 \rightarrow \varphi_2 & \text{iff } \alpha \models \varphi_1 \text{ implies } \alpha \models \varphi_2 \\
\alpha & \models \varphi_1 \leftrightarrow \varphi_2 & \text{iff } \alpha \models \varphi_1 \text{ iff } \alpha \models \varphi_2
\end{array}
$$

Example

- Let φ be defined as $(a \vee (b \to c))$.
- Let $\alpha : \{a, b, c\} \to \{0, 1\}$ be an assignment with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.
- Q: Does α satisfy φ?

A2: Compute with the satisfaction relation:

$$\begin{array}{ll} & \alpha \models (a \vee (b \to c)) \\ \textit{iff} & \alpha \models a \text{ or } \alpha \models (b \to c) \\ \textit{iff} & \alpha \models a \text{ or } (\alpha \models b \textit{ implies } \alpha \models c) \\ \textit{iff} & 0 \textit{ or } (0 \textit{ implies } 1) \\ \textit{iff} & 0 \textit{ or } 1 \\ \textit{iff} & 1 \end{array}$$

Semantics III: The algorithmic view

- Using the satisfaction relation we can define an algorithm for the problem to decide if an assignment $\alpha : AP \rightarrow \{0,1\}$ is a model of a propositional logic formula φ with variables from AP:

```
Eval($\varphi$, $\alpha$) {
   if $\varphi \equiv a$ return $\alpha(a)$;
   if $\varphi \equiv (\neg\varphi_1)$ return not Eval($\varphi_1, \alpha$);
   if $\varphi \equiv (\varphi_1$ op $\varphi_2)$
         return Eval($\varphi_1, \alpha$) $[\![$op$]\!]$ Eval($\varphi_2, \alpha$);
}
```

- Complexity? Eval uses polynomial time and space.

Example

- Recall our example
  - $\varphi = (a \vee (b \rightarrow c))$
  - $\alpha : \{a, b, c\} \rightarrow \{0, 1\}$ with $\alpha(a) = 0$, $\alpha(b) = 0$, and $\alpha(c) = 1$.

- $$\begin{aligned} \mathrm{Eval}(\varphi, \alpha) \;=\;& \mathrm{Eval}(a, \alpha) \textit{ or } \mathrm{Eval}(b \rightarrow c, \alpha) = \\ & 0 \textit{ or } (\mathrm{Eval}(b, \alpha) \textit{ implies } \mathrm{Eval}(c, \alpha)) = \\ & 0 \textit{ or } (0 \textit{ implies } 1) = \\ & 0 \textit{ or } 1 = \\ & 1 \end{aligned}$$

- Hence, $\alpha \models \varphi$.

Set of assignments

- Intuition: a formula specifies a set of truth assignments.
- Remember: *Ass* denotes the set of all assignments.
- Function models : Formula $\rightarrow 2^{Ass}$

  (a formula $\rightarrow$ set of satisfying assignments)
- Recursive definition:
  - $\text{models}(a) = \{\alpha \mid \alpha(a) = 1\}$, $a \in \texttt{Prop}$
  - $\text{models}(\neg\varphi_1) = Ass \setminus \text{models}(\varphi_1)$
  - $\text{models}(\varphi_1 \wedge \varphi_2) = \text{models}(\varphi_1) \cap \text{models}(\varphi_2)$
  - $\text{models}(\varphi_1 \vee \varphi_2) = \text{models}(\varphi_1) \cup \text{models}(\varphi_2)$
  - $\text{models}(\varphi_1 \rightarrow \varphi_2) = (Ass \setminus \text{models}(\varphi_1)) \cup \text{models}(\varphi_2)$

Example

- models$(a \vee b) = \{\alpha \in Ass \mid \alpha(a) = 1$ or $\alpha(b) = 1\}$

- This is compatible with the recursive definition:

  $$\text{models}(a \vee b) = models(a) \cup models(b) = \\ \{\alpha \in Ass \mid \alpha(a) = 1\} \cup \{\alpha \in Ass \mid \alpha(b) = 1\}$$

Theorem

- Let $\varphi \in$ Formula and $\alpha \in \mathit{Ass}$, then the following statements are equivalent:
    1. $\alpha \models \varphi$
    2. $\alpha \in \text{models}(\varphi)$

Extension of $\models$ to sets of assignments

- Let $\varphi \in$ Formula.
- Let T be a set of assignments, i.e., $T \subseteq 2^{Ass}$
- Definition: $\models \subseteq 2^{Ass} \times$ Formula with

$$T \models \varphi \text{ iff } T \subseteq \text{models}(\varphi)$$

Extension of $\models$ to formulae

- $\models \subseteq 2^{\text{Formula}} \times 2^{\text{Formula}}$

- Definition. Let φ_1, φ_2 be propositional formulae.

  $\varphi_1 \models \varphi_2$

  iff models(φ_1) $\subseteq$ models(φ_2), or equivalently
  iff for all $\alpha \in Ass$

  if $\alpha \models \varphi_1$ then $\alpha \models \varphi_2$

  Examples:

  $x_1 \wedge x_2 \models x_1 \vee x_2$

  $x_1 \wedge x_2 \models x_2 \vee x_3$

Short summary for propositional logic

- Syntax: $\varphi \;:=\; \texttt{prop} \mid (\neg\varphi) \mid (\varphi \wedge \varphi)$
- Semantics:
    - Assignments:

$$\begin{array}{l} \alpha : \texttt{Prop} \rightarrow \{0,1\} \\ \alpha \in 2^{\texttt{Prop}} \\ \alpha \in \{0,1\}^{\texttt{Prop}} \end{array}$$

    - Satisfiability relation:

$$\begin{array}{rlcll} \models & \subseteq \mathit{Ass} \times \text{Formula} & , & (\text{e.g., } \alpha & \models \varphi\;) \\ \models & \subseteq 2^{\mathit{Ass}} \times \text{Formula} & , & (\text{e.g., } \{\alpha_1, \ldots, \alpha_n\} & \models \varphi\;) \\ \models & \subseteq \text{Formula} \times \text{Formula} & , & (\text{e.g., } \varphi_1 & \models \varphi_2) \\ \text{models} : & \text{Formula} \rightarrow 2^{\mathit{Ass}} & , & (\text{e.g., } \;\text{models}(\varphi) & \;) \end{array}$$

Propositional logic - Outline

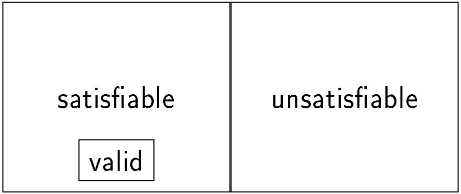
- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Semantic classification of formulae

- A formula φ is called valid if $\text{models}(\varphi) = \mathit{Ass}$.
  (Also called a tautology).

- A formula φ is called satisfiable if $\text{models}(\varphi) \neq \emptyset$.

- A formula φ is called unsatisfiable if $\text{models}(\varphi) = \emptyset$.
  (Also called a contradiction).



Some notations

- We can write:
    - $\models \varphi$ when φ is valid
    - $\not\models \varphi$ when φ is not valid
    - $\not\models \neg\varphi$ when φ is satisfiable
    - $\models \neg\varphi$ when φ is unsatisfiable

Examples

- $(x_1 \land x_2) \to (x_1 \lor x_2)$ is valid
- $(x_1 \lor x_2) \to x_1$ is satisfiable
- $(x_1 \land x_2) \land \neg x_1$ is unsatisfiable

Examples

- Here are some valid formulae:
  - $\models a \wedge 1 \leftrightarrow a$
  - $\models a \wedge 0 \leftrightarrow 0$
  - $\models \neg\neg a \leftrightarrow a$ // The double-negation rule
  - $\models a \wedge (b \vee c) \leftrightarrow (a \wedge b) \vee (a \wedge c)$

- Some more (De Morgan rules):
  - $\models \neg(a \wedge b) \leftrightarrow (\neg a \vee \neg b)$
  - $\models \neg(a \vee b) \leftrightarrow (\neg a \wedge \neg b)$

The decision problem of formulae

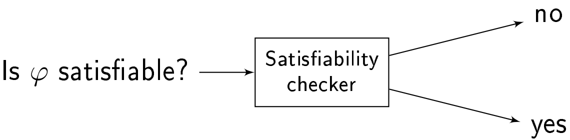
- The decision problem:

  Given a propositional formula φ, is φ satisfiable?

- An algorithm that always terminates with a correct answer to this problem is called a decision procedure for propositional logic.

Characteristics of formulae

Goal: Design a satisfiability checker



Lemma:

- A formula φ is valid iff $\neg\varphi$ is unsatisfiable.

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Before we solve this problem...

- Suppose we can solve the satisfiability problem... how can this help us?
- There are numerous problems in the industry that are solved via the satisfiability problem of propositional logic
  - Logistics
  - Planning
  - Electronic Design Automation industry
  - Cryptography
  - . . .

Example 1: Placement of wedding guests

- Three chairs in a row: $1, 2, 3$
- We need to place Aunt, Sister and Father.
- Constraints:
  - Aunt doesn't want to sit near Father
  - Aunt doesn't want to sit in the left chair
  - Sister doesn't want to sit to the right of Father

- Q: Can we satisfy these constraints?

Example 1 (continued)

- Denote: Aunt = 1, Sister = 2, Father = 3
- Introduce a propositional variable for each pair (person, place).
- x_{ij} = "person i is sited in place j, for $1 \leq i, j \leq 3$"
- Constraints:
  - Aunt doesn't want to sit near Father:
    $((x_{1,1} \vee x_{1,3}) \rightarrow \neg x_{3,2}) \wedge (x_{1,2} \rightarrow (\neg x_{3,1} \wedge \neg x_{3,3}))$
  - Aunt doesn't want to sit in the left chair
    $\neg x_{1,1}$
  - Sister doesn't want to sit to the right of Father
    $x_{3,1} \rightarrow \neg x_{2,2} \wedge x_{3,2} \rightarrow \neg x_{2,3}$

Example 1 (continued)

- More constraints:
- Each person is placed:

  $(x_{1,1} \vee x_{1,2} \vee x_{1,3}) \wedge (x_{2,1} \vee x_{2,2} \vee x_{2,3}) \wedge (x_{3,1} \vee x_{3,2} \vee x_{3,3})$
- Or, more concisely:

$$\bigwedge_{i=1}^{3} \bigvee_{j=1}^{3} x_{i,j}$$

- No person is placed in more than one place:

$$\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{2} \bigwedge_{k=j+l}^{3} (\neg x_{i,j} \vee \neg x_{i,k})$$

- Overall 9 variables, 26 conjoined constraints.

Example 2: Assignment of frequencies

- n radio stations
- For each assign one of k transmission frequencies, $k < n$.
- E – set of pairs of stations, that are too close to have the same frequency.

- Q: Can we assign to each station a frequency, such that no statin pairs from E have the same frequency?

Example 2 (continued)

- $x_{i,j}$: station i is assigned frequency j, for $1 \leq i \leq n$, $1 \leq j \leq k$.
  - Every station is assigned at least one frequency:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} x_{i,j}$$

  - Every station is assigned not more than one frequency:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k-1} (x_{i,j} \rightarrow \bigwedge_{j<t\leq k} \neg x_{i,t})$$

  - Close stations are not assigned the same frequency:
    For each $(i,j) \in E$,

$$\bigwedge_{t=1}^{k} (x_{i,t} \rightarrow \neg x_{j,t})$$

Two classes of algorithms for validity

- Q: Is φ satisfiable? (Is $\neg\varphi$ valid?)
- Complexity: NP-Complete (Cook's theorem)
- Two classes of algorithms for finding out:
  - Enumeration of possible solutions (Truth tables etc.)
  - Deduction
- More generally (beyond propositional logic):
  - Enumeration is possible only in some logics.
  - Deduction cannot necessarily be fully automated.

The satisfiability problem

- Given a formula φ, is φ satisfiable?

Enumeration the first:

```
Boolean SAT($\varphi$){
        result:=false;
        for all $\alpha \in Ass$
            result = result $\vee$ Eval($\varphi, \alpha$);
        return result;
}
```

Enumeration the second:

Use substitution to eliminate all variables one by one:

$$\varphi \quad \text{iff} \quad \varphi[0/a] \vee \varphi[1/a]$$

- What is the difference?
- There must be a better way to do that in practice.

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Definitions

- Definition: A literal is either a variable or a negation of a variable.
- Let $\varphi = \neg(a \vee \neg b)$. Then:
- Variables: $\mathrm{AP}(\varphi) = \{a, b\}$
- Literals: $\mathrm{lit}(\varphi) = \{a, \neg b\}$
- Equivalent formulae can have different literals
- $\varphi' = \neg a \wedge b$
- Now $\mathrm{lit}(\varphi') = \{\neg a, b\}$

Definitions

- Definition: a term is a conjunction of literals
  - Example: $(a \wedge \neg b \wedge c)$

- Definition: a clause is a disjunction of literals
  - Example: $(a \vee \neg b \vee c)$

Negation Normal Form (NNF)

- Definition: A formula is in Negation Normal Form (NNF) iff
  (1) it contains only $\neg$, $\wedge$ and $\vee$ as connectives and
  (2) only variables are negated.

- Examples:
- $\varphi_1 = \neg(a \vee \neg b)$ is not in NNF
- $\varphi_2 = \neg a \wedge b$ is in NNF

Converting to NNF

- Every formula can be converted to NNF in linear time:
  - Eliminate all connectives other than $\wedge$, $\vee$, $\neg$
  - Use De Morgan and double-negation rules to push negations to the right

- Example: $\varphi = \neg(a \rightarrow \neg b)$
  - Eliminate '$\rightarrow$' : $\varphi = \neg(\neg a \vee \neg b)$
  - Push negation using De Morgan: $\varphi = (\neg\neg a \wedge \neg\neg b)$
  - Use double-negation rule: $\varphi = (a \wedge b)$

Disjunctive Normal Form (DNF)

- Definition: A formula is said to be in Disjunctive Normal Form (DNF) iff it is a disjunction of terms.
  - In other words, it is a formula of the form

    $$\bigvee_i (\bigwedge_j l_{i,j})$$

    where $l_{i,j}$ is the j-th literal in the i-th term.

- Example:

  $$\varphi = (a \land \neg b \land c) \lor (\neg a \land d) \lor (b) \text{ is in DNF}$$

- DNF is a special case of NNF

Converting to DNF

- Every formula can be converted to DNF in exponential time and space:
  1. Convert to NNF
  2. Distribute disjunctions following the rule: $\models a \land (b \lor c) \leftrightarrow ((a \land b) \lor (a \land c))$
- Example:

$$
\begin{aligned}
\varphi \quad & = (a \lor b) \land (\neg c \lor d) \\
& = ((a \lor b) \land (\neg c)) \lor ((a \lor b) \land d) \\
& = (a \land \neg c) \lor (b \land \neg c) \lor (a \land d) \lor (b \land d)
\end{aligned}
$$

- Q: How many clauses would the DNF have had if we started from a conjunction of n binary clauses (i.e., clauses with 2 literals)?

Satisfiability of DNF

- Is the following DNF formula satisfiable?
  $(a_1 \land a_2 \land \neg a_1) \lor (a_2 \land a_1) \lor (a_2 \land \neg a_3 \land a_3)$

- Q: What is the complexity of the satisfiability check of DNF formulae?

Conjunctive Normal Form (CNF)

- Definition: A formula is said to be in Conjunctive Normal Form (CNF) iff it is a conjunction of clauses.

  In other words, it is a formula of the form

  $$\bigwedge_i (\bigvee_j l_{i,j})$$

  where $l_{i,j}$ is the j-th literal in the i-th clause.

- Example:

  $$\varphi = (a \vee \neg b \vee c) \wedge (\neg a \vee d) \wedge (b) \ \text{ is in CNF}$$

- CNF is a special case of NNF

Converting to CNF

- Every formula can be converted to CNF:
  - in exponential time and space with the same set of variables, or
  - in linear time and space if new variables are added.
- For the latter—the so-called Tseitin's encoding—the original and the converted formulae are equi-satisfiable, but not equivalent.

- Q: Can there be any such linear transformation into DNF?
- A: No. Linear DNF transformation and linear DNF solution would violate the NP-completeness of the problem.

Converting to CNF: The exponential way

```
CNF($\varphi$){
case
    $\varphi$ is a literal: return $\varphi$
    $\varphi$ is $\varphi_1 \wedge \varphi_2$: return CNF($\varphi_1$) $\wedge$ CNF($\varphi_2$)
    $\varphi$ is $\varphi_1 \vee \varphi_2$: return Dist(CNF($\varphi_1$),CNF($\varphi_2$))
}
```

```
Dist($\varphi_1$,$\varphi_2$) {
case
    $\varphi_1$ is $\varphi_{11} \wedge \varphi_{12}$: return Dist($\varphi_{11}$,$\varphi_2$) $\wedge$ Dist($\varphi_{12}$,$\varphi_2$)
    $\varphi_2$ is $\varphi_{21} \wedge \varphi_{22}$: return Dist($\varphi_1$,$\varphi_{21}$) $\wedge$ Dist($\varphi_1$,$\varphi_{22}$)
    else: return $\varphi_1 \vee \varphi_2$
}
```

Converting to CNF: The exponential way

- Consider the formula
- $\varphi = (a_1 \wedge b_1) \vee (a_2 \wedge b_2)$
- $\text{CNF}(\varphi) = (a_1 \vee a_2) \wedge (a_1 \vee b_2) \wedge (b_1 \vee a_2) \wedge (b_1 \vee b_2)$

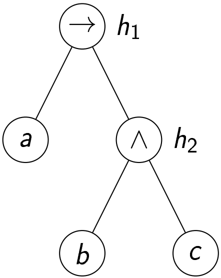
- Now consider: $\varphi_n = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee \ldots \vee (a_n \wedge b_n)$
- Q: How many clauses does CNF(φ) return?
- A: 2^n

Converting to CNF: Tseitin's encoding

- Consider the formula

  $\varphi = (a \rightarrow (b \wedge c))$

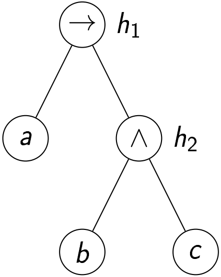
The Parse Tree:



- Associate a new auxiliary variable with each gate.
- Add constraints that define these new variables.
- Finally, enforce the root node.

Converting to CNF: Tseitin's encoding

- Need to satisfy:
  $(h_1 \leftrightarrow (a \ \rightarrow \ h_2))\wedge$
  $(h_2 \leftrightarrow (b \wedge c))\wedge$
  (h_1)



- Each gate encoding has a CNF representation with 3 or 4 clauses.

Converting to CNF: Tseitin's encoding

- Need to satisfy:

  $(h_1 \leftrightarrow (a \rightarrow h_2)) \wedge (h_2 \leftrightarrow (b \wedge c)) \wedge (h_1)$

- First: $(h_1 \vee a) \wedge (h_1 \vee \neg h_2) \wedge (\neg h_1 \vee \neg a \vee h_2)$
- Second: $(\neg h_2 \vee b) \wedge (\neg h_2 \vee c) \wedge (h_2 \vee \neg b \vee \neg c)$

Converting to CNF: Tseitin's encoding

- Let's go back to
  $\varphi_n = (x_1 \land y_1) \lor (x_2 \land y_2) \lor \dots \lor (x_n \land y_n)$

- With Tseitin's encoding we need:
  - n auxiliary variables $a_1, \dots, a_n$.
  - Each adds 3 constraints.
  - Top clause: $(a_1 \lor \dots \lor a_n)$

- Hence, we have
  - $3n + 1$ clauses, instead of 2^n.
  - $3n$ variables rather than $2n$.

Propositional logic - Outline

- Abstract grammar of propositional logic
- Semantics of propositional logic
- Satisfiability and validity
- Modeling with propositional logic
- Normal forms
- Deductive proofs and resolution

Deduction requires axioms and inference rules

- Inference rules:

$$\frac{\text{Antecedents}}{\text{Consequents}} \qquad \text{(rule-name)}$$

  Meaning: If all antecedents hold then at least one of the consequents can be derived.

- Examples:

$$\frac{a \rightarrow b \qquad b \rightarrow c}{a \rightarrow c} \qquad \text{(Trans)}$$

$$\frac{a \rightarrow b \qquad a}{b} \qquad \text{(M.P.)}$$

Axioms

- Axioms are inference rules with no antecedents, e.g.,

$$\frac{}{a \rightarrow (b \rightarrow a)} \quad \text{(H1)}$$

- We can turn an inference rule into an axiom if we have '$\rightarrow$' in the logic.
- So the difference between them is not sharp.

Proofs

- A proof uses a given set of axioms and inference rules.
- This is called the proof system.
- Let $\mathcal{H}$ be a proof system.
- $\Gamma \vdash_{\mathcal{H}} \varphi$ means: There is a proof of φ in system $\mathcal{H}$ whose premises are included in Γ
- $\vdash_{\mathcal{H}}$ is called the provability relation.

Example

- Let $\mathcal{H}$ be the proof system comprised of the rules Trans and M.P. that we saw earlier:

$$\frac{a \rightarrow b \quad b \rightarrow c}{a \rightarrow c} \qquad \text{(Trans)}$$

$$\frac{a \rightarrow b \quad a}{b} \qquad \text{(M.P.)}$$

- Does the following relation hold?

$$a \rightarrow b,\ b \rightarrow c,\ c \rightarrow d,\ d \rightarrow e,\ a \quad \vdash_{\mathcal{H}} \quad e$$

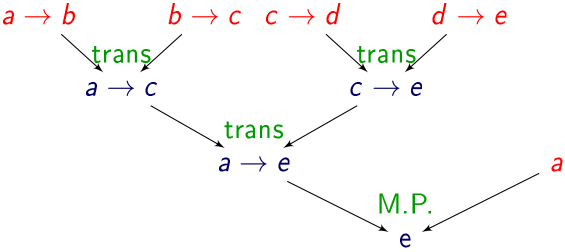
Deductive proof: Example

$$\frac{a \rightarrow b \quad b \rightarrow c}{a \rightarrow c} \quad \text{(Trans)} \qquad \frac{a \rightarrow b \quad a}{b} \quad \text{(M.P.)}$$

$$a \rightarrow b,\ b \rightarrow c,\ c \rightarrow d,\ d \rightarrow e,\ a \quad \vdash_{\mathcal{H}} \quad e$$

1.	$a \rightarrow b$	premise
2.	$b \rightarrow c$	premise
3.	$a \rightarrow c$	1, 2, Trans
4.	$c \rightarrow d$	premise
5.	$d \rightarrow e$	premise
6.	$c \rightarrow e$	4, 5, Trans
7.	$a \rightarrow e$	3, 6, Trans
8.	a	premise
9.	e	7, 8, M.P.

Proof graph (DAG)



$a \rightarrow b$ $b \rightarrow c$ $c \rightarrow d$ $d \rightarrow e$

trans trans

$a \rightarrow c$ $c \rightarrow e$

trans

$a \rightarrow e$ a

M.P.

e

Correctness and Completeness

- $\vdash$ is a relation defined by syntactic transformations of the underlying proof system.
- For a given proof system $\mathcal{H}$,
  - Correctness: Does $\vdash$ conclude "correct" conclusions from premises?
  - Completeness: Can we conclude all true statements with $\mathcal{H}$?
- Correct with respect to what?
- With respect to the semantic definition of the logic. In the case of propositional logic truth tables give us this.

Soundness and completeness

- Let $\mathcal{H}$ be a proof system

$$\begin{array}{llllll}
\text{Soundness of } \mathcal{H}: & \text{if} & \vdash_{\mathcal{H}} & \varphi & \text{then} & \models \quad \varphi \\
\text{Completeness of } \mathcal{H}: & \text{if} & \models & \varphi & \text{then} & \vdash_{\mathcal{H}} \quad \varphi
\end{array}$$

- How to prove soundness and completeness?

Example: Hilbert axiom system (H)

- Let H be (M.P.) together with the following axiom schemes:

$$\frac{}{a \rightarrow (b \rightarrow a)} \quad (H1)$$

$$\frac{}{((a \rightarrow (b \rightarrow c)) \rightarrow ((a \rightarrow b) \rightarrow (a \rightarrow c)))} \quad (H2)$$

$$\frac{}{(\neg b \rightarrow \neg a) \rightarrow (a \rightarrow b)} \quad (H3)$$

- H is sound and complete for propositional logic.

Soundness and completeness

- To prove soundness of H, prove the soundness of its axioms and inference rules (easy with truth-tables).

  For example:

a	b	$a \rightarrow (b \rightarrow a)$
0	0	1
0	1	1
1	0	1
1	1	1

- Completeness: harder, but possible.

The resolution inference system

- The resolution inference rule for CNF:

$$\frac{(l \lor l_1 \lor l_2 \lor \ldots \lor l_n) \quad (\neg l \lor l'_1 \lor \ldots \lor l'_m)}{(l_1 \lor \ldots \lor l_n \lor l'_1 \lor \ldots \lor l'_m)} \text{ Resolution}$$

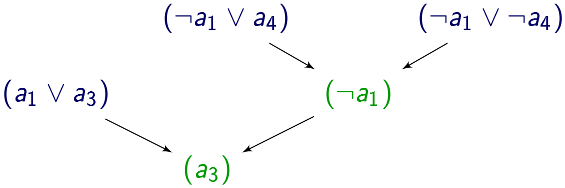
- Example:

$$\frac{(a \lor b) \quad (\neg a \lor c)}{(b \lor c)}$$

- We first see some example proofs, before proving soundness and completeness.

Proof by resolution

- Let $\varphi = (a_1 \vee a_3) \wedge (\neg a_1 \vee a_2 \vee a_5) \wedge (\neg a_1 \vee a_4) \wedge (\neg a_1 \vee \neg a_4)$
- We'll try to prove $\varphi \;\rightarrow\; (a_3)$



$(\neg a_1 \vee a_4)$

$(\neg a_1 \vee \neg a_4)$

$(a_1 \vee a_3)$

$(\neg a_1)$

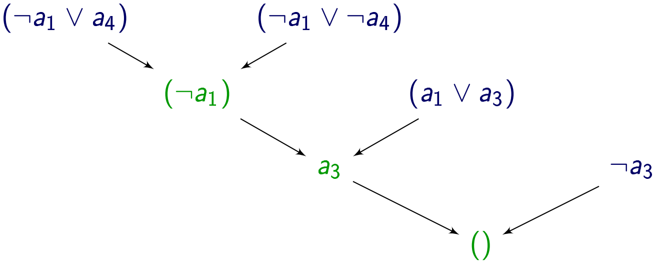
(a_3)

Resolution

- Resolution is a sound and complete inference system for CNF.
- If the input formula is unsatisfiable, there exists a proof of the empty clause.

Example

Let $\varphi = (a_1 \vee a_3) \wedge (\neg a_1 \vee a_2) \wedge (\neg a_1 \vee a_4) \wedge (\neg a_1 \vee \neg a_4) \wedge (\neg a_3)$.



$(\neg a_1 \vee a_4)$

$(\neg a_1 \vee \neg a_4)$

$(\neg a_1)$

$(a_1 \vee a_3)$

a_3

$\neg a_3$

$()$

Soundness and completeness of resolution

- Soundness is straightforward. Just prove by truth table that

$$\models ((\varphi_1 \vee a) \wedge (\varphi_2 \vee \neg a)) \rightarrow (\varphi_1 \vee \varphi_2).$$

- Completeness is a bit more involved.
  Basic idea: Use resolution for variable elimination .

$$\begin{gathered}
(a \vee \varphi_1) \wedge \ldots \wedge (a \vee \varphi_n) \wedge \\
(\neg a \vee \psi_1) \wedge \ldots (\neg a \vee \psi_m) \wedge \\
R \\
\Leftrightarrow \\
(\varphi_1 \vee \psi_1) \wedge \ldots \wedge (\varphi_1 \vee \psi_m) \wedge \\
\ldots \\
(\varphi_n \vee \psi_1) \wedge \ldots (\varphi_n \vee \psi_m) \wedge \\
R
\end{gathered}$$

  where φ_i $(i = 1, \ldots, n)$, ψ_j $(j = 1, \ldots, m)$, and R contains neither a nor $\neg a$.