Lecture 2

Propositional logic

syntax and semantics, the satisfiability problem, constraint problems

Dr Christoph Haase
University of Oxford
(with small changes by Javier Esparza)

Agenda

1. Propositional Logic
2. Syntax and semantics of propositional logic
3. Encoding constraint problems into satisfiability problems

Propositional logic

- Formulas of propositional logic (aka propositions) are built of smaller formulas using *connectives* like *and*, *or*, *not*, *implies*, and others.
- The smallest formulas are *propositional variables*, aka *atomic propositions* or *atoms*, which can be instantiated with statements which are either true or false.
- A prime concern: *given a compound formula, determine which truth values of its atoms make it true*.

An example

- Atomic propositions:

Atoms	Instance
a	“Alice is an architect”
b	“Bob is a builder”
c	“Charlie is a cook”

An example

- Atomic propositions:

Atoms	Instance
a	“Alice is an architect”
b	“Bob is a builder”
c	“Charlie is a cook”

- Compound formulas:

$\neg c$	“Charlie is not a cook”
$a \vee b$	“Alice is an architect or Bob is a builder”
$b \rightarrow c$	“If Bob is a builder then Charlie is a cook”

An example

- Atomic propositions:

Atoms	Instance
a	“Alice is an architect”
b	“Bob is a builder”
c	“Charlie is a cook”

- Compound formulas:

$\neg c$	“Charlie is not a cook”
$a \vee b$	“Alice is an architect or Bob is a builder”
$b \rightarrow c$	“If Bob is a builder then Charlie is a cook”

- These entail that Alice is an architect: if the above three propositions are all true then a *must* also be true ($\{\neg c,\ a \vee b,\ b \rightarrow c\} \models a$).

An example

- Atomic propositions:

Atoms	Instance
a	"Alice is an architect"
b	"Bob is a builder"
c	"Charlie is a cook"

- Compound formulas:

$\neg c$	"Charlie is not a cook"
$a \vee b$	"Alice is an architect or Bob is a builder"
$b \to c$	"If Bob is a builder then Charlie is a cook"

- These entail that Alice is an architect: if the above three propositions are all true then a *must* also be true ($\{\neg c, a \vee b, b \to c\} \models a$).
- The correctness of this entailment is *independent* of the instantiation of the atomic propositions!

1. Propositional Logic
2. Syntax and semantics of propositional logic
3. Encoding constraint problems into satisfiability problems

Syntax of propositional logic

Definition (Syntax of propositional logic)

Let $X = \{x_1, x_2, x_3, \ldots\}$ be a countably infinite set of **propositional variables**. **Formulas** of propositional logic are inductively defined as follows:

1. *true* and *false* are formulas.
2. Every propositional variable x_i is a formula.
3. If F is a formula, then $\neg F$ is a formula.
4. If F and G are formulas, then $(F \wedge G)$ and $(F \vee G)$ are formulas.

Additional notation

- We often write x, y, z or p to denote propositional variables.
- We call $\neg F$ the **negation** of F.
- Given formulas F and G, $(F \wedge G)$ is the **conjunction** of F and G, and $(F \vee G)$ is the **disjunction** of F and G.
- We call $\neg$, $\wedge$ and $\vee$ **logical connectives**.
- We denote by $\mathcal{F}(X)$ the **set of all formulas** built from propositional variables in X.

Derived connectives

- **Implication**: $(F_1 \rightarrow F_2) := (\neg F_1 \vee F_2)$
- **Bi-implication**: $(F_1 \leftrightarrow F_2) := (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$
- **Exclusive Or**: $(F_1 \oplus F_2) := (F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$
- **Indexed Conjunction**: $\bigwedge_{i=1}^{n} F_i := (\cdots((F_1 \wedge F_2) \wedge F_3) \wedge \cdots \wedge F_n)$
- **Indexed Disjunction**: $\bigvee_{i=1}^{n} F_i := (\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
- Note on bracketing:
  - We usually drop outer brackets
  - Operator precedences: $\leftrightarrow$ and $\rightarrow$ bind weaker than $\wedge$ and $\vee$, which bind weaker than $\neg$.
  - Example: $\neg x \wedge y \rightarrow z$ means $((\neg x \wedge y) \rightarrow z)$

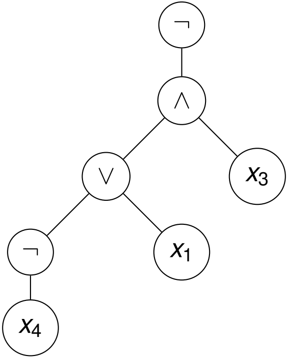
Syntax trees

- Every formula F can be represented by a *syntax tree* whose nodes are labelled either by connectives or by propositional variables.
- **Subformulas** of F correspond to all subtrees of F

Syntax trees

- Every formula F can be represented by a *syntax tree* whose nodes are labelled either by connectives or by propositional variables.
- **Subformulas** of F correspond to all subtrees of F

Example: syntax tree of $\neg((\neg x_4 \vee x_1) \wedge x_3)$:



Inductive definitions

Inductive definition of formulas allows us to define functions on formulas by **structural induction**, by defining the function

- For the base cases *true*, *false* and x_i, and
- For the induction steps $\neg F$, $F \wedge G$ and $F \vee G$.

Inductive definitions

Inductive definition of formulas allows us to define functions on formulas by **structural induction**, by defining the function

- For the base cases *true*, *false* and x_i, and
- For the induction steps $\neg F$, $F \wedge G$ and $F \vee G$.

Example

The function $sub\colon \mathcal{F}(X) \to 2^{\mathcal{F}(X)}$ returning the set of all subformulas of a given formula can be defined by:

- $sub(true) = \{true\}$, $sub(false) = \{false\}$
- $sub(x) = \{x\}$ for all $x \in X$
- $sub(\neg F) = \{\neg F\} \cup sub(F)$
- $sub(F \wedge G) = \{F \wedge G\} \cup sub(F) \cup sub(G)$
- $sub(F \vee G) = \{F \vee G\} \cup sub(F) \cup sub(G)$

Syntax vs semantics

The *syntax* tells us how we write something down, the *semantics* what it means:

- syntax: some formal *language*
- semantics: some mathematical *object*
- our syntax: propositional formulas
- our semantics: **truth tables**

Semantics of propositional logic

Definition

An **assignment** is a function $\mathcal{A}\colon X \to \{0,1\}$ that induces an assignment $\hat{\mathcal{A}}\colon \mathcal{F}(X) \to \{0,1\}$ by structural induction as follows:

1. $\hat{\mathcal{A}}(\mathit{false}) = 0$, $\hat{\mathcal{A}}(\mathit{true}) = 1$
2. For every $x \in X$, $\hat{\mathcal{A}}(x) := \mathcal{A}(x)$
3. $\hat{\mathcal{A}}(\neg F) := \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 0 \\ 0 & \text{otherwise} \end{cases}$
4. $\hat{\mathcal{A}}((F \wedge G)) := \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ and } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}$
5. $\hat{\mathcal{A}}((F \vee G)) := \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise} \end{cases}$

Semantics of propositional logic

- The semantics of a formula F is the function that maps each assignment $\mathcal{A}\colon X \to \{0,1\}$ to the truth value $\hat{\mathcal{A}}(F)$.
- Let $Y \subseteq X$ be the set of variables occurring in F. $\hat{\mathcal{A}}(F)$ is completely determined by the values assigned by $\mathcal{A}$ to the variables of Y.
- With a slight abuse of language, we also say that the semantics of F is the function that maps each restricted assignment $\mathcal{A}'\colon Y \to \{0,1\}$ to the truth value $\widehat{\mathcal{A}'}(F)$.
- Observe that X is infinite, but Y is finite. So there is an uncountable infinity of assignments, but only $2^{|Y|}$ restricted assignments.

Semantics of propositional logic

Example

Let $F = (x \wedge \neg y) \vee z$ and $\mathcal{A}$ be an assignment such that $\mathcal{A}(x) = 1$ and $\mathcal{A}(y) = \mathcal{A}(z) = 0$. Then F evaluates to true under $\mathcal{A}$, since

$$\begin{aligned}
\hat{\mathcal{A}}(F) &= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}((x \wedge \neg y)) = 1 \text{ or } \hat{\mathcal{A}}(z) = 1 \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}((x \wedge \neg y)) = 1 \text{ (since } \mathcal{A}(z) = 0) \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(x) = 1 \text{ and } \hat{\mathcal{A}}(\neg y) = 1 \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(y) = 0 \text{ (since } \mathcal{A}(x) = 1) \\ 0 & \text{otherwise} \end{cases} \\
&= 1 \text{ (since } \mathcal{A}(y) = 0).
\end{aligned}$$

Semantics of propositional logic

Example

Let $F = (x \wedge \neg y) \vee z$ and $\mathcal{A}$ be an assignment such that $\mathcal{A}(x) = 1$ and $\mathcal{A}(y) = \mathcal{A}(z) = 0$. Then F evaluates to true under $\mathcal{A}$, since

$$\begin{aligned}
\hat{\mathcal{A}}(F) &= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}((x \wedge \neg y)) = 1 \text{ or } \hat{\mathcal{A}}(z) = 1 \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}((x \wedge \neg y)) = 1 \text{ (since } \mathcal{A}(z) = 0) \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(x) = 1 \text{ and } \hat{\mathcal{A}}(\neg y) = 1 \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(y) = 0 \text{ (since } \mathcal{A}(x) = 1) \\ 0 & \text{otherwise} \end{cases} \\
&= 1 \text{ (since } \mathcal{A}(y) = 0).
\end{aligned}$$

Subsequently we will not write the hat on top of $\mathcal{A}$.

Semantics via truth tables

Example

The semantics of logical connectives via **truth tables**:

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
1	0	0
0	1	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
1	0	1
0	1	1
1	1	1

Semantics via truth tables

Example

The semantics of logical connectives via **truth tables**:

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
1	0	0
0	1	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
1	0	1
0	1	1
1	1	1

Semantics via truth tables

Example

The semantics of logical connectives via **truth tables**:

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
1	0	0
0	1	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
1	0	1
0	1	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \rightarrow G)$
0	0	1
1	0	0
0	1	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \oplus G)$
0	0	0
1	0	1
0	1	1
1	1	0

Semantics via truth tables

Example

The semantics of logical connectives via **truth tables**:

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \wedge G)$
0	0	0
1	0	0
0	1	0
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \vee G)$
0	0	0
1	0	1
0	1	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \rightarrow G)$
0	0	1
1	0	0
0	1	1
1	1	1

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \oplus G)$
0	0	0
1	0	1
0	1	1
1	1	0

Formalising natural language: an example

A device consists of a thermostat, a pump, and a warning light.
Suppose we are told the following four facts about the pump:

- The thermostat or the pump (or both) are broken.
- If the thermostat is broken then the pump is also broken.
- If the pump is broken and the warning light is on then the thermostat is not broken.
- The warning light is on.

Is it possible for all four to be true at the same time?

Formalising natural language: an example

A device consists of a thermostat, a pump, and a warning light.
Suppose we are told the following four facts about the pump:

- The thermostat or the pump (or both) are broken.
- If the thermostat is broken then the pump is also broken.
- If the pump is broken and the warning light is on then the thermostat is not broken.
- The warning light is on.

Is it possible for all four to be true at the same time?
In a propositional formula:

$$F := (t \vee p) \wedge (t \rightarrow p) \wedge ((p \wedge w) \rightarrow \neg t) \wedge w$$

Truth table

$$F := (t \lor p) \land (t \to p) \land ((p \land w) \to \neg t) \land w$$

t	p	w	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Truth table

$$F := (t \vee p) \wedge (t \rightarrow p) \wedge ((p \wedge w) \rightarrow \neg t) \wedge w$$

t	p	w	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

There is a unique assignment that makes F true. We can think of each assignment as describing a *possible world*, and there is only one world in which F is true.

Models, satisfiability and validity

Definition

Let $F \in \mathcal{F}(X)$ and $\mathcal{A}\colon X \to \{0, 1\}$ be an assignment.

1. If $\mathcal{A}(F) = 1$ then we write $\mathcal{A} \models F$ ("F *holds under* $\mathcal{A}$", or "$\mathcal{A}$ is a **model** of F".)
2. If F has at least one model then F is **satisfiable**, otherwise F is **unsatisfiable**.
3. If F holds under any assignment $\mathcal{A}\colon X \to \{0, 1\}$ then F is called **valid** or a **tautology**, written $\models F$.

Models, satisfiability and validity

Definition

Let $F \in \mathcal{F}(X)$ and $\mathcal{A}\colon X \to \{0,1\}$ be an assignment.

1. If $\mathcal{A}(F) = 1$ then we write $\mathcal{A} \models F$ ("*F holds under $\mathcal{A}$*", or "$\mathcal{A}$ is a **model** of F".)
2. If F has at least one model then F is **satisfiable**, otherwise F is **unsatisfiable**.
3. If F holds under any assignment $\mathcal{A}\colon X \to \{0,1\}$ then F is called **valid** or a **tautology**, written $\models F$.

Definition

Given $F \in \mathcal{F}(X)$, the **Boolean satisfiability problem (SAT)** is to decide whether F is satisfiable.

Models, satisfiability and validity

Example

The subsequent first two tautologies are known as the *distributive laws*, the last two as *De Morgan's laws*:

$$\models (F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H))$$
$$\models (F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H))$$
$$\models \neg(F \wedge G) \leftrightarrow \neg F \vee \neg G$$
$$\models \neg(F \vee G) \leftrightarrow \neg F \wedge \neg G.$$

Entailment and equivalence

Definition (Entailment)

A formula G is a **consequence** of (or is **entailed** by) a set of formulas $\mathcal{S}$ if every assignment that satisfies each formula in $\mathcal{S}$ also satisfies G. In this case we write $\mathcal{S} \models G$.

Entailment and equivalence

Definition (Entailment)

A formula G is a **consequence** of (or is **entailed** by) a set of formulas $\mathcal{S}$ if every assignment that satisfies each formula in $\mathcal{S}$ also satisfies G. In this case we write $\mathcal{S} \models G$.

Definition (Equivalence)

Two formulas F and G are said to be **logically equivalent** if $\mathcal{A}(F) = \mathcal{A}(G)$ for every assignment $\mathcal{A}$. We write $F \equiv G$ to denote that F and G are equivalent.

1. Propositional Logic
2. Syntax and semantics of propositional logic
3. **Encoding constraint problems into satisfiability problems**

Sudoku

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$ we have a proposition $x_{i,j,k}$ expressing that *grid position* i, j *contains number* k. Build formula F as the conjunction of the following *constraints*:

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$ we have a proposition $x_{i,j,k}$ expressing that *grid position* i, j *contains number* k. Build formula F as the conjunction of the following *constraints*:

- Each number appears in each row and in each column:

$$F_1 := \bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{j=1}^{9} x_{i,j,k} \qquad F_2 := \bigwedge_{j=1}^{9} \bigwedge_{k=1}^{9} \bigvee_{i=1}^{9} x_{i,j,k}$$

- Each number appears in each 3×3 block:

$$F_3 := \bigwedge_{k=1}^{9} \bigwedge_{u=0}^{2} \bigwedge_{v=0}^{2} \bigvee_{i=1}^{3} \bigvee_{j=1}^{3} x_{3u+i,3v+j,k}$$

- No square contains two numbers:

$$F_4 := \bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{1 \le k < k' \le 9} \neg(x_{i,j,k} \wedge x_{i,j,k'}) \,.$$

Sudoku

- Certain numbers appear in certain positions: we assert

$$F_5 := x_{2,1,2} \wedge x_{1,2,8} \wedge x_{2,3,3} \wedge \ldots \wedge x_{8,9,6} .$$

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Sudoku

- Missing constraints? What about: no number appears twice in the same row?

$$F_6 := \bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{1 \leq j < j' < 9} \neg(x_{i,j,k} \wedge x_{i,j',k})$$

- Entailed by the existing formulas: adding F_6 as an extra constraint would not change the set of satisfying assignments.

Sudoku

- Missing constraints? What about: no number appears twice in the same row?

$$F_6 := \bigwedge_{i=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{1 \le j < j' < 9} \neg(x_{i,j,k} \wedge x_{i,j',k})$$

- Entailed by the existing formulas: adding F_6 as an extra constraint would not change the set of satisfying assignments.

- But adding logically redundant constraints may help a computer search for a satisfying assignment.

- The number of variables $x_{i,j,k}$ is $9^3 = 729$. Thus a truth table for the corresponding formula would have $2^{729} > 10^{200}$ lines! Nevertheless a modern SAT-solver can find a satisfying assignment in milliseconds.

Hamiltonian path

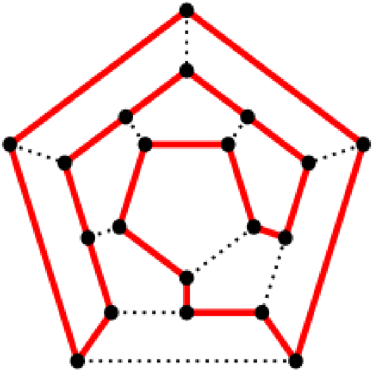


Figure: Example of a Hamiltonian path in an undirected graph.

Hamiltonian path

Given an undirected graph $G = (V, E)$ such that $E \subseteq V \times V$ is symmetric, for each vertex $i, j \in \{1, \ldots, n\}$ we have a proposition $x_{i,j}$ expressing that *vertex i is the jth vertex in the Hamiltonian path*. Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once:

$$F_1 := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} x_{i,j} \qquad F_2 := \bigwedge_{i=1}^{n} \bigwedge_{1 \leq j \neq k \leq n} \neg(x_{i,j} \wedge x_{i,k}) \wedge \neg(x_{j,i} \wedge x_{k,i})$$

- The path goes along edges:

$$F_4 := \bigwedge_{i=1}^{n} \bigwedge_{k=1}^{n} \bigwedge_{j=1}^{n-1} x_{i,j} \wedge x_{k,j+1} \rightarrow e_{i,k}$$

$$F_5 := \bigwedge_{(i,j) \in E} e_{i,j} \wedge \bigwedge_{(i,j) \notin E} \neg e_{i,j}$$

Polynomial-time vs exponential-time

Polynomial-time vs exponential-time

- Can solve SAT in time $O(2^n)$ (via truth tables).

Polynomial-time vs exponential-time

- Can solve SAT in time $O(2^n)$ (via truth tables).
- No sub-exponential algorithm is known (e.g., $O(n^{1028})$, $n^{\log(n)}$, $2^{n/log(n)}$,...)

Polynomial-time vs exponential-time

- Can solve SAT in time $O(2^n)$ (via truth tables).
- No sub-exponential algorithm is known (e.g., $O(n^{1028})$, $n^{\log(n)}$, $2^{n/log(n)}$,...)
- Can do better for special formula classes: Horn formulas, 2-CNF formulas, XOR-clauses, ...

Polynomial-time vs exponential-time

- Can solve SAT in time $O(2^n)$ (via truth tables).
- No sub-exponential algorithm is known (e.g., $O(n^{1028})$, $n^{\log(n)}$, $2^{n/log(n)}$,...)
- Can do better for special formula classes: Horn formulas, 2-CNF formulas, XOR-clauses, ...
- Reductions of combinatorial problems to SAT should run in polynomial-time!