Lecture 2

Propositional logic

syntax and semantics, the satisfiability problem, constraint problems

Introduction to Logic for Computer Science

Prof Hongseok Yang
KAIST

These slides are minor variants of those made by Prof Worrell and Prof Haase for their logic course at Oxford.

Agenda

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

Propositional logic

- Informally, a study on a type of boolean expressions in PLs, called sentences, formulas or propositions.
- The most basic kind of sentences are *atomic propositions*, which can be true, or false, or variables.
- Sentences are combined using *logical connectives*.
- Propositional logic analyses how the truth values of compound sentences depend on their constituents.
- A prime concern: *given a compound sentence, determine which truth values of its atoms make it true*.
- Key to formulate the notions of *logical consequence* and *valid argument*.

An example

- Atomic propositions:

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

An example

- Atomic propositions:

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

- Compound sentences:

$\neg c$	“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:

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

- Compound sentences:

$\neg c$	"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"

- Ex: Assume the three propositions. What can you say about a?

An example

- Atomic propositions:

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

- Compound sentences:

$\neg c$	"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"

- Ex: Assume the three propositions. What can you say about a?
- Answer: Alice is an architect. That is, $\{\neg c,\ a \vee b,\ b \rightarrow c\} \models a$.
- The correctness of this entailment is *independent* of the meaning of the atomic propositions!

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**: $(F_1 \rightarrow F_2) \;:=\; (\neg F_1 \vee F_2)$

Derived connectives

- **Implication**: $(F_1 \to F_2) \ := \ (\neg F_1 \vee F_2)$
- **Bi-implication**: $(F_1 \leftrightarrow F_2) \ := \ ??$
- **Exclusive Or**: $(F_1 \oplus F_2) \ := \ ??$
- **Indexed Conjunction**: $\bigwedge_{i=1}^{n} F_i := ??$
- **Indexed Disjunction**: $\bigvee_{i=1}^{n} F_i := ??$
- Ex1: Fill in ??.
- Ex2: Find a connective that can be used to define all the others.
- Ex3: Prove that $\wedge$ is not an answer for Ex2.

Derived connectives

- **Implication**: $(F_1 \to F_2) \ := \ (\neg F_1 \vee F_2)$
- **Bi-implication**: $(F_1 \leftrightarrow F_2) \ := \ (F_1 \to F_2) \wedge (F_2 \to F_1)$
- **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**: $\bigvee_{i=1}^{n} F_i := (\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
- Ex1: Fill in ??.
- Ex2: Find a connective that can be used to define all the others.
- Ex3: Prove that $\wedge$ is not an answer for Ex2.

Derived connectives

- **Implication**: $(F_1 \to F_2) \ := \ (\neg F_1 \vee F_2)$
- **Bi-implication**: $(F_1 \leftrightarrow F_2) \ := \ (F_1 \to F_2) \wedge (F_2 \to F_1)$
- **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**: $\bigvee_{i=1}^{n} F_i := (\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
- Ex1: Fill in ??.
- Ex2: Find a connective that can be used to define all the others.
- Ex3: Prove that $\wedge$ is not an answer for Ex2.

Derived connectives

- **Implication**: $(F_1 \to F_2) \;:=\; (\neg F_1 \vee F_2)$
- **Bi-implication**: $(F_1 \leftrightarrow F_2) \;:=\; (F_1 \to F_2) \wedge (F_2 \to F_1)$
- **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**: $\bigvee_{i=1}^{n} F_i := (\cdots((F_1 \vee F_2) \vee F_3) \vee \cdots \vee F_n)$
- Ex1: Fill in ??.
- Ex2: Find a connective that can be used to define all the others.
- Ex3: Prove that $\wedge$ is not an answer for Ex2. By Mono.

Convention on bracketing

- We drop brackets, unless doing so causes big confusion.
- No outer brackets usually.
- Use the standard precedence of connectives.
- Example: $\neg x \wedge y \to z$ means $(((\neg x) \wedge y) \to z)$.
- Lecture notes for the detail. Ask me when confused.

Syntax trees

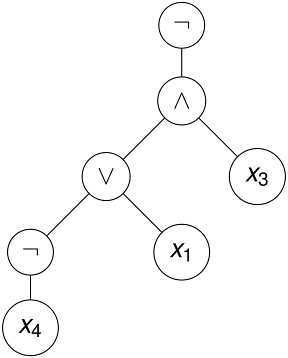
- Every formula F can be represented by a *syntax tree* whose nodes are labelled with connectives or propositional variables.

- **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 with connectives or 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 $size : \mathcal{F}(X) \to \mathbb{N}$ returning the number of symbols in a given formula can be defined by:

- $size(true) = size(false) = size(x) = 1$;
- $size(\neg F) = 1 + size(F)$;
- $size(F \wedge G) = size(F \vee G) = size(F) + size(G) + 1$.

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 $size : \mathcal{F}(X) \to \mathbb{N}$ returning the number of symbols in a given formula can be defined by:

- $size(true) = size(false) = size(x) = 1$;
- $size(\neg F) = 1 + size(F)$;
- $size(F \wedge G) = size(F \vee G) = size(F) + size(G) + 1$.

- Ex1: What is $size(\neg((\neg x_4 \vee x_1) \wedge x_3))$?
- Ex2: Define a function $sub : \mathcal{F}(X) \to 2^{\mathcal{F}(X)}$ that returns the set of all subformulas of a given formula.

Syntax vs semantics

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

- syntax: some formal *language*.
- semantics: some mathematical *model*.
- semantics should capture the ‘essence’ of what’s going on.
- have to have semantics to prove anything *about* syntax.

Syntax vs semantics

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

- syntax: some formal *language*.
- semantics: some mathematical *model*.
- semantics should capture the ‘essence’ of what’s going on.
- have to have semantics to prove anything *about* syntax.
- our syntax: propositional formulas.
- our semantics: **truth values** $\{0, 1\}$.

Semantics of propositional logic

Definition

An **assignment** is a function $\mathcal{A} \colon X \to \{0, 1\}$. It induces a function $\hat{\mathcal{A}} \colon \mathcal{F}(X) \to \{0, 1\}$, called **assignment** again, by structural induction:

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

Definition

An **assignment** is a function $\mathcal{A}\colon X \to \{0,1\}$. It induces a function $\hat{\mathcal{A}}\colon \mathcal{F}(X) \to \{0,1\}$, called **assignment** again, by structural induction:

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. $\hat{\mathcal{A}}(F \vee G) := \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ or } \hat{\mathcal{A}}(G) = 1, \\ 0 & \text{otherwise.} \end{cases}$

Ex: Let $\mathcal{A}$ be an assignment s.t. $\mathcal{A}(x) = 1$ and $\mathcal{A}(y) = \mathcal{A}(z) = 0$.
What are $\hat{\mathcal{A}}((x \wedge \neg y) \vee z)$ and $\hat{\mathcal{A}}(x \wedge (x \vee y) \wedge (y \vee \neg z))$?

Semantics of propositional logic

Definition

An **assignment** is a function $\mathcal{A}\colon X \to \{0,1\}$. It induces a function $\hat{\mathcal{A}}\colon \mathcal{F}(X) \to \{0,1\}$, called **assignment** again, by structural induction:

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. $\hat{\mathcal{A}}(F \vee G) := \begin{cases} 1 & \text{if } \hat{\mathcal{A}}(F) = 1 \text{ or } \hat{\mathcal{A}}(G) = 1 \\ 0 & \text{otherwise.} \end{cases}$

From now on 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

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(F \to G)$
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 device:

- 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.

Ex: Is it possible for all four to be true at the same time? Express this question using a formula, and answer it using semantics.

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 device:

- 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.

Ex: Is it possible for all four to be true at the same time? Express this question using a formula, and answer it using semantics.

Answer:

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

So, yes under $\mathcal{A}$ with $\mathcal{A}(t) = 0$ and $\mathcal{A}(p) = \mathcal{A}(w) = 1$.

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

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", or "$\mathcal{A}$ **satisfies** 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", or "$\mathcal{A}$ **satisfies** 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

The **Boolean satisfiability problem (SAT)** is to decide whether a given formula $F \in \mathcal{F}(X)$ is satisfiable.

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", or "$\mathcal{A}$ **satisfies** 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

The **Boolean satisfiability problem (SAT)** is to decide whether a given formula $F \in \mathcal{F}(X)$ is satisfiable.

Ex: Suppose that we have a program for solving SAT. How to convert it to a checker for validity?

Example

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

$$\begin{aligned}
&\models (F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H)) \\
&\models (F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H)) \\
&\models \neg(F \wedge G) \leftrightarrow \neg F \vee \neg G \\
&\models \neg(F \vee G) \leftrightarrow \neg F \wedge \neg G.
\end{aligned}$$

Ex: Prove the last two.

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 all formulas 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 all formulas 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.

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 all formulas 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.

Ex: Suppose that we have a program for solving SAT. Convert it to checkers for entailment and equivalence.

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	

How to encode an instance of Sudoku into the satisfiability of a propositional formula?

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$, we have a propositional variable $x_{i,j,k}$ expressing that *grid position* i, j *contains number* k.

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$, we have a propositional variable $x_{i,j,k}$ 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:

- Each number appears in each 3×3 block:

- No square contains two numbers:

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$, we have a propositional variable $x_{i,j,k}$ 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:

- No square contains two numbers:

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$, we have a propositional variable $x_{i,j,k}$ 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:

- No square contains two numbers:

Sudoku

For each $i, j, k \in \{1, \ldots, 9\}$, we have a propositional variable $x_{i,j,k}$ 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:

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

- 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

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

	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?

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

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?

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

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

- 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

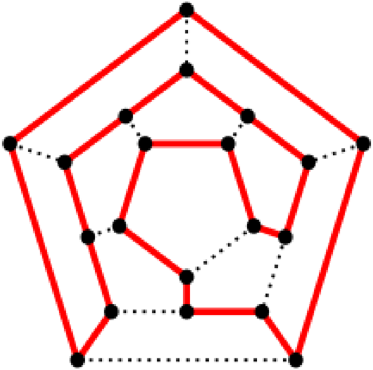


Figure: Example of a Hamiltonian path in an undirected graph.

How to encode an instance of the Hamiltonian path problem into the satisfiability of a propositional formula?

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \ldots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that *i is the jth vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \ldots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that *i is the jth vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once,
- The path goes along edges:
- $e_{i,j}$ encodes E:

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \ldots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that *i is the jth vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once,

$$F_0 := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} x_{i,j}, \quad F_1 := \bigwedge_{i=1}^{n} \bigwedge_{1 \leq j \neq k \leq n} \neg(x_{i,j} \wedge x_{i,k}),$$

- The path goes along edges:

- $e_{i,j}$ encodes E:

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \dots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that *i is the jth vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once,

$$F_0 := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} x_{i,j}, \quad F_1 := \bigwedge_{i=1}^{n} \bigwedge_{1 \leq j \neq k \leq n} \neg(x_{i,j} \wedge x_{i,k}),$$

- The path goes along edges:

- $e_{i,j}$ encodes E:

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \ldots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that *i is the jth vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once,

$$F_0 := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} x_{i,j}, \quad F_1 := \bigwedge_{i=1}^{n} \bigwedge_{1 \le j \ne k \le n} \neg(x_{i,j} \wedge x_{i,k}),$$

- The path goes along edges:

$$F_3 := \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=1}^{n-1} x_{i,k} \wedge x_{j,k+1} \to e_{i,j}.$$

- $e_{i,j}$ encodes E:

$$F_4 := \bigwedge_{(i,j) \in E} e_{i,j} \wedge \bigwedge_{(i,j) \notin E} \neg e_{i,j}.$$

Hamiltonian path for an undirected graph $G = (V, E)$

For each vertex $i, j \in \{1, \ldots, n\}$, we have propositional variables

- $x_{i,j}$ expressing that i *is the* j*th vertex in the Hamiltonian path*;
- $e_{i,j}$ expressing that there is an edge from vertex i to vertex j.

Build formula F as the conjunction of the following *constraints*:

- Each vertex is visited precisely once, and some vertex is visited at each step of the path:

$$F_0 := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} x_{i,j}, \quad F_1 := \bigwedge_{i=1}^{n} \bigwedge_{1 \leq j \neq k \leq n} \neg(x_{i,j} \wedge x_{i,k}), \quad F_2 := \bigwedge_{j=1}^{n} \bigvee_{i=1}^{n} x_{i,j}.$$

- The path goes along edges:

$$F_3 := \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=1}^{n-1} x_{i,k} \wedge x_{j,k+1} \rightarrow e_{i,j}.$$

- $e_{i,j}$ encodes E:

$$F_4 := \bigwedge_{(i,j) \in E} e_{i,j} \wedge \bigwedge_{(i,j) \notin E} \neg e_{i,j}.$$

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., $O(n^{1028})$, $n^{\log(n)}$, $2^{n/log(n)}$,...).
- In fact, the existence of a polynomial time algorithm is equivalent to $\mathbf{P} = \mathbf{NP}$.

Polynomial-time vs exponential-time

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