Yes, No, Maybe, I Don't Know

A Journey into Automated Reasoning

Romain Pascual

MICS, CentraleSupélec, Université Paris-Saclay

April, 17, 2025

Yes, No, Maybe, I Don't Know

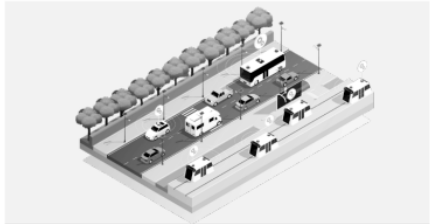
Yes, No, Maybe, I Don't Know

Introduction

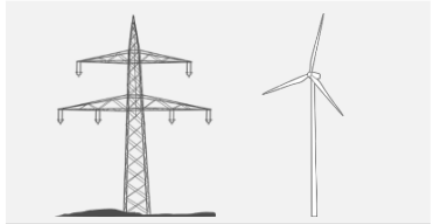
Syllogisms: First Steps in Reasoning

Every man is mortal (H). Socrates is a man (H). Therefore, Socrates is mortal (C).

Everything that is rare is expensive (H). A cheap horse is rare (H). Therefore, a cheap horse is expensive (C).



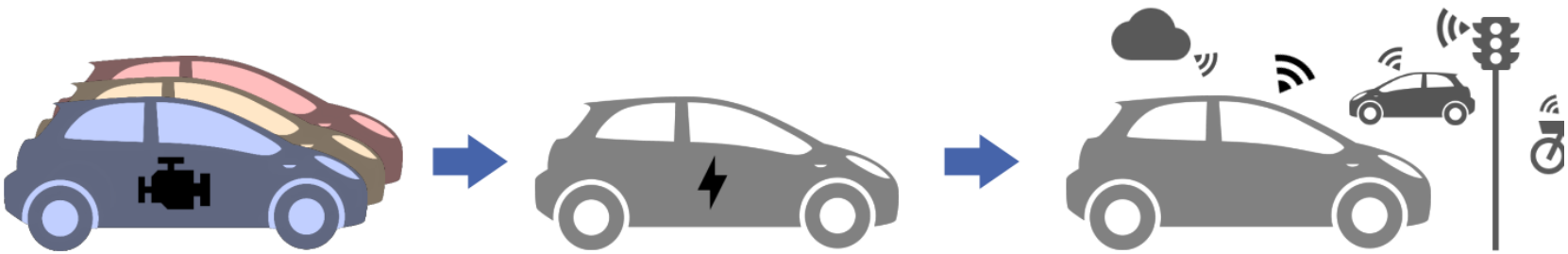
Mobility



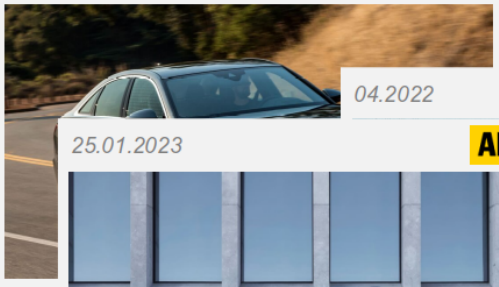
Energy

Production

...



01.12.2022



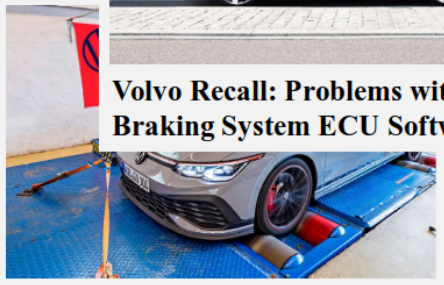
US I

04.2022

25.01.2023 ADAC

Volvo Recall: Problems with the Braking System ECU Software

16.12.20



Service Action: 150.000 Golfs Need New Software

Recall of BMW i4/i7/iX: Proble with Battery Management

02.11.2022 t-or Nachrichten

Camera Software Failure: Recall Mercedes

14.10.2022

Opel recalls 194.000 Ins Possibly Extended Bral

01.02.2023 auto motor sport

Recall of BMW i4/iX: Pedestrian Warning System Could Fail

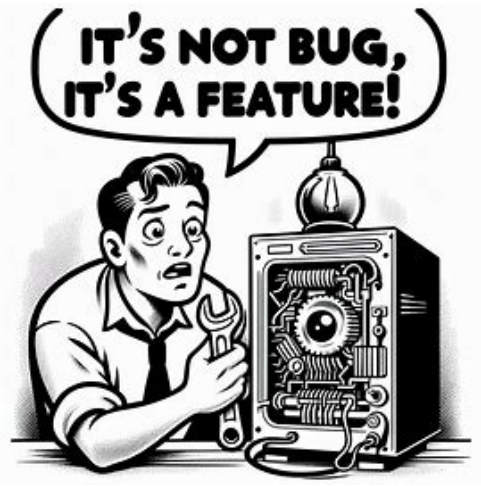
16.02.2023 tagesschau

Driver Assistance Software: Tesla Recalls 360.000 Cars in the US


HAVE YOU TRIED
TURNING IT OFF AND ON AGAIN?
quickmeme.com

Formal Methods?

A solution to improve the overall quality of software and systems



Testing

- "How do you know if a system is working?"

Testing

- "How do you know if a system is working?"
- "Let's test it!"

Testing

- "How do you know if a system is working?"

- "Let's test it!"

- "How many inputs exists? What about the inputs you did not test?"

Testing

- "How do you know if a system is working?"

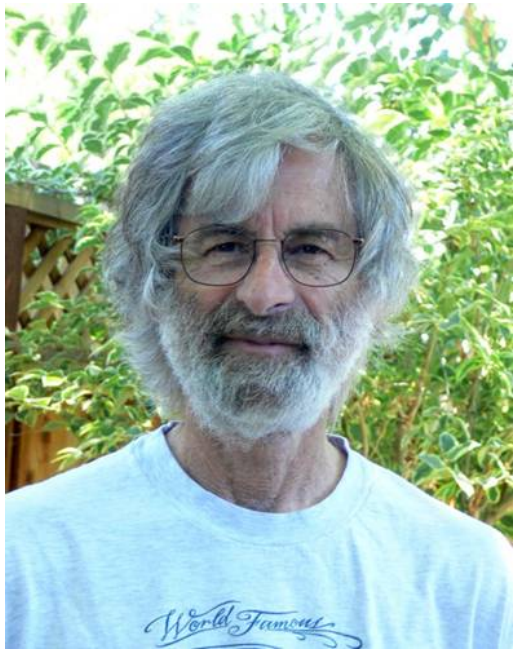
- "Let's test it!"

- "How many inputs exists? What about the inputs you did not test?"

Testing shows the *presence of errors*, in general not their absence!

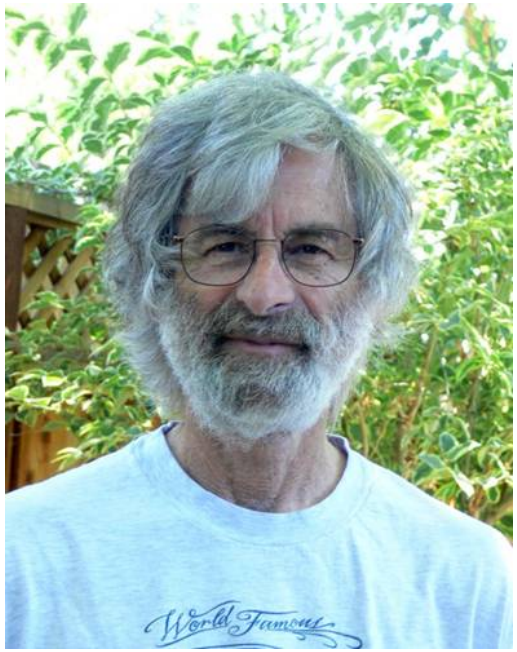
Not tests

Specifying a system helps us understand it.



Specifying a system helps us understand it.

It's a good idea to understand a system before building it, so it's a good idea to write a specification of a system before implementing it.
– Lamport 2002



What are formal methods?

Mathematically founded techniques and tools for the specification, design, realisation or verification of systems.

What are formal methods?

Mathematically founded techniques and tools for the specification, design, realisation or verification of systems.

Two aspects:

- System *specification*
- System *implementation*

Explicit formal statements for both and use tools to *mechanically* prove that *formal implementation* satisfies *formal specification*

Properties that can be checked

- Simple properties
  - Safety properties - *Something bad will never happen*
    eg: mutual exclusion, no buffer overflow
  - Liveness properties - *Something good will happen eventually*
- General properties of concurrent/distributed systems
  - deadlock-free, no starvation, fairness
- Security properties
  - non-interference, information flow
  - access control
  - availability
- Non-functional properties
  - Runtime, memory, usability

Automating Reasoning

Why Automate?

Instead of proving results manually, we can leverage algorithmic techniques to automate the process.

Automating Reasoning

Why Automate?

Instead of proving results manually, we can leverage algorithmic techniques to automate the process.

What Are Decision Procedures?

- Algorithms designed to automatically reason about logical formulae.
- Given a formula, they:
  - Prove its validity, or
  - Find a counterexample.

Propositional Logic and SAT Solvers

Starting Simple

We begin with the simplest logic: propositional logic.

Propositional Logic and SAT Solvers

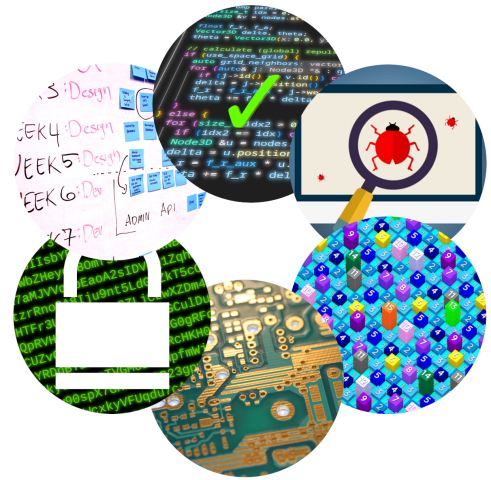
Starting Simple

We begin with the simplest logic: propositional logic.

SAT Solvers

- Decision procedures for propositional logic are called SAT solvers.
- They exploit the relationship between:
  - **Validity:** Is the formula always true?
  - **Satisfiability:** Can the formula be true under some interpretation?
- SAT solvers directly solve the satisfiability problem.

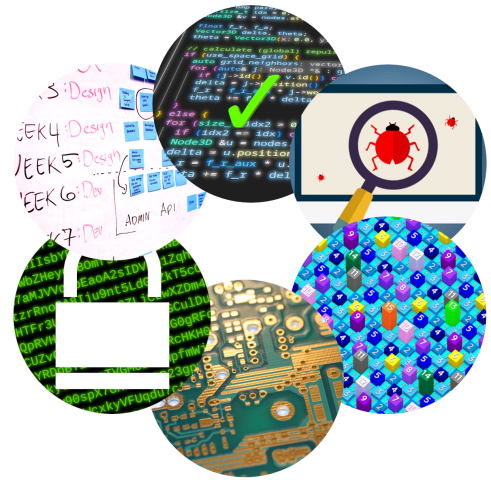
Applications of SAT Solving



Hardware verification and design

- Major hardware companies (Intel, . . .) use SAT to verify chip designs
- Computer Aided Design of electronic circuits

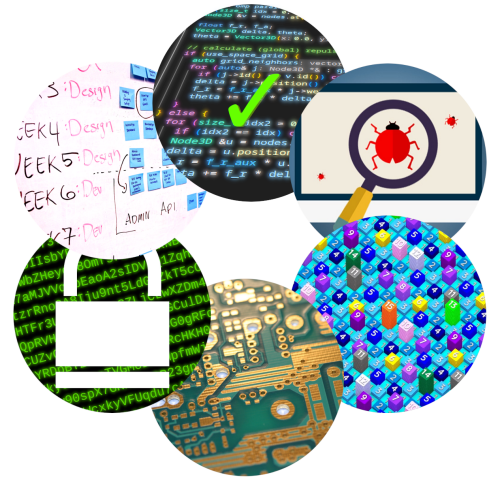
Applications of SAT Solving



Software verification

- SAT-based SMT solvers are used to verify Microsoft software products
  (also great interest at Amazon – AWS software in particular)
- Embedded software in cars, airplanes, refrigerators, . . .
- Unix utilities

Applications of SAT Solving



Automated planning and scheduling in Artificial Intelligence

- Job shop scheduling, train scheduling, multi-agent path finding

Number theoretic problems (Pythagorean triples, grid coloring)

Solving other difficult problems (coloring, clique, . . .)

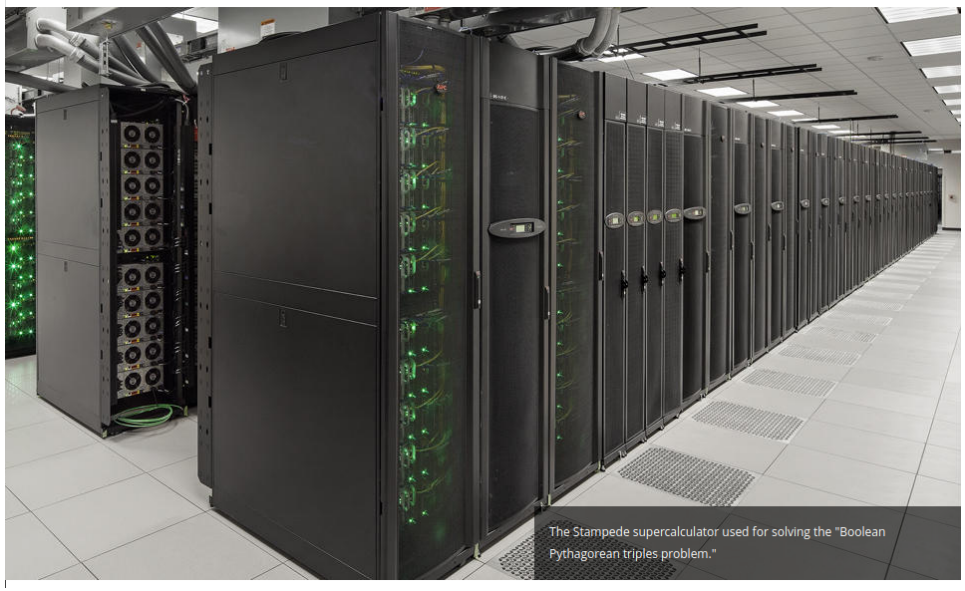
SAT Solving in the News

DIGITAL MATHEMATICS -A +A Print ARTICLE

The Longest Proof in the History of Mathematics

07.20.2016

Reading time: 4 minutes



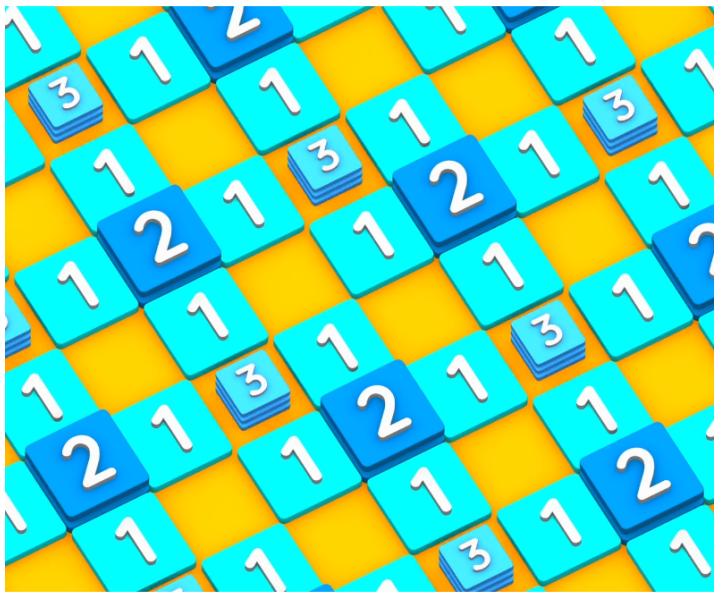
The Stampede supercalculator used for solving the "Boolean Pythagorean triples problem."

COMBINATORICS

The Number 15 Describes the Secret Limit of an Infinite Grid

13

The "packing coloring" problem asks how many numbers are needed to fill an infinite grid so that identical numbers never get too close to one another. A new computer-assisted proof finds a surprisingly straightforward answer.



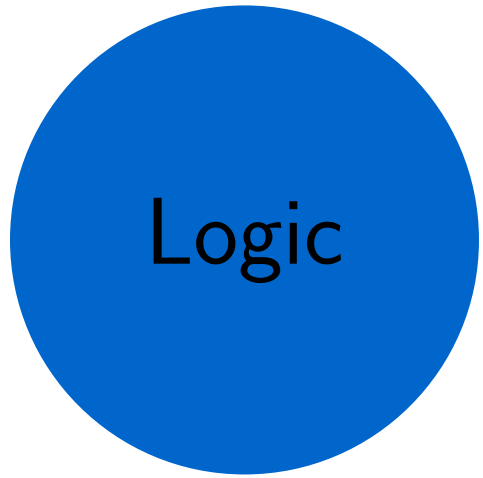
Disclaimer

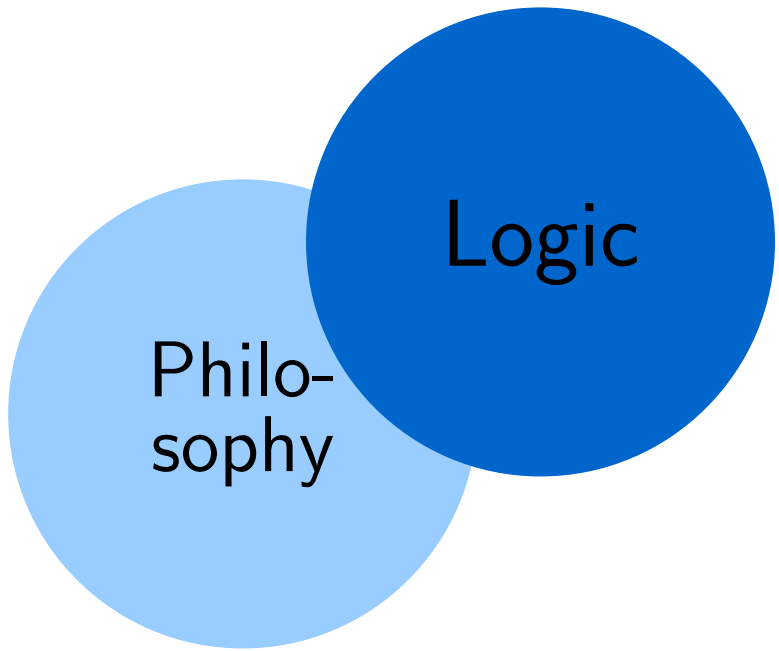
I reused contents from several lectures:

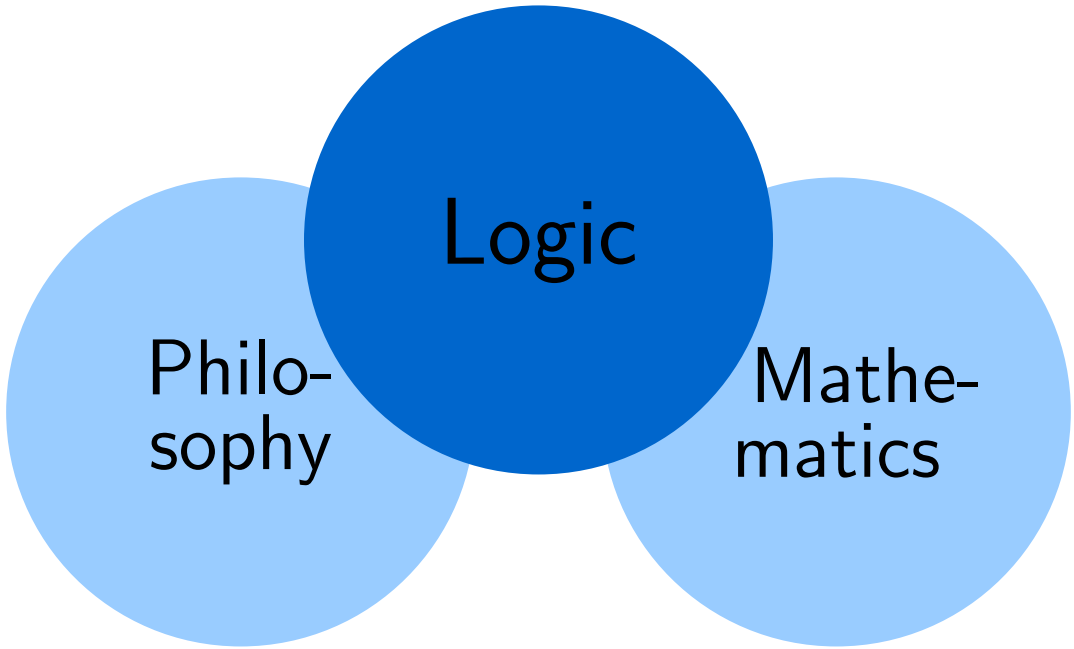
- "Logic" by Pascale Le Gall Pascale Le Gall and Marc Aiguier at CentraleSupélec.
- "Practical SAT Solving" by Markus Iser, Dominik Schreiber, and Tomas Balyo at KIT.
- "Bug Catching: Automated Program Verification" by Ruben Martins at Carnegie Mellon University.

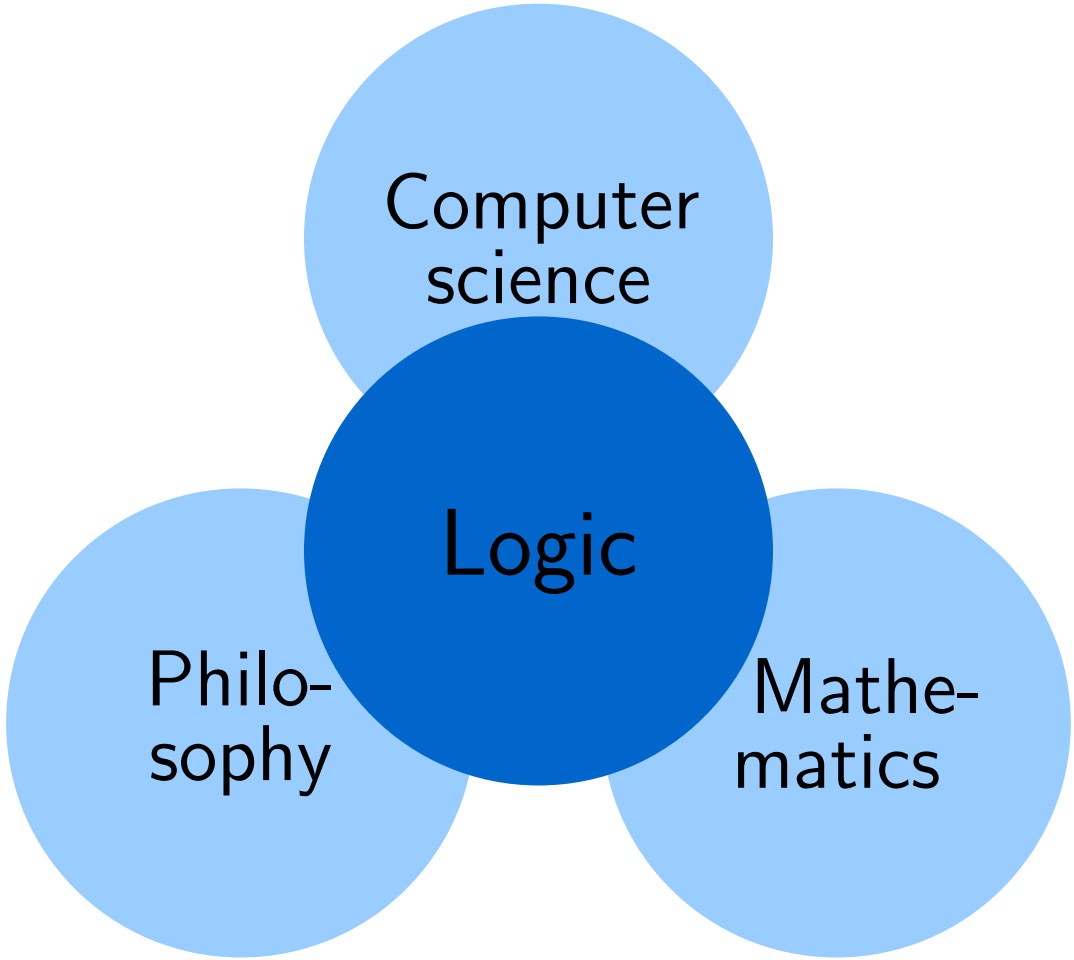
Outline

Logic


Logic


Logic
Philo-
sophy

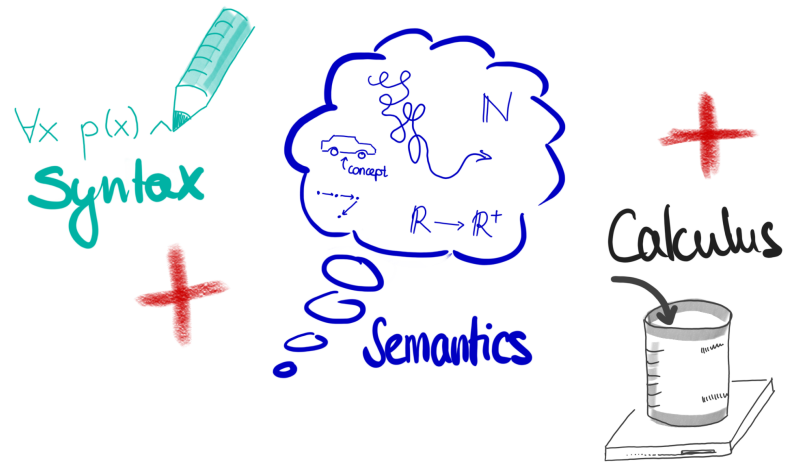

Logic
Philo-
sophy
Mathe-
matics


Computer
science
Logic
Philo-
sophy
Mathe-
matics

Structure of Logics

A logic is a formal system for reasoning about truth content.

- **Syntax**
  Symbols and constructions to assemble the symbols into sentences
- **Semantics**
  Meaning of the symbols and connectives to assert the meaning of a sentence
- **Calculus**
  A set of inference rules to reason about the "truth content" of a sentence by syntactic modifications



SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

1. What is a propositional formula?

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

1. What is a propositional formula?
2. What is a satisfiable formula?

Recap on Propositional Logic

Is this Argument Convincing?

- If the autonomous car caused the accident, then the accident occurred due to a sensor malfunction, or the driver overrode the autonomous driving.
- If the accident occurred due to a sensor malfunction, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- However, the damage was significant.
- Therefore, the autonomous car did not cause the accident.

Syntax

How to represent the sentences in a formal way?

Statements

- *(Sentence 1)* If the autonomous car caused the accident, then the accident occurred due to a sensor malfunction, or the driver overrode the autonomous driving.
- *(Sentence 2)* If the accident occurred due to a sensor malfunction, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- *(Sentence 3)* However, the damage was significant.
- *(Sentence 4)* Therefore, the autonomous car did not cause the accident.

Statements

- *(Sentence 1)* If **a**, then the accident occurred due to a sensor malfunction, or the driver overrode the autonomous driving.
- *(Sentence 2)* If the accident occurred due to a sensor malfunction, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- *(Sentence 3)* However, the damage was significant.
- *(Sentence 4)* Therefore, not **a**.

More elementary statements:

(**a**) The autonomous car caused the accident

Statements

- *(Sentence 1)* If **a**, then **s**, or the driver overrode the autonomous driving.
- *(Sentence 2)* If **s**, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- *(Sentence 3)* However, the damage was significant.
- *(Sentence 4)* Therefore, not **a**.

More elementary statements:

(**a**) The autonomous car caused the accident

(**s**) The accident occurred due to a sensor malfunction

Statements

- *(Sentence 1)* If **a**, then **s**, or **o**.
- *(Sentence 2)* If **s**, then, if **o**, less significant damage would have resulted.
- *(Sentence 3)* However, the damage was significant.
- *(Sentence 4)* Therefore, not **a**.

More elementary statements:

(**a**) The autonomous car caused the accident

(**s**) The accident occurred due to a sensor malfunction

(**o**) The driver overrode the autonomous driving

Statements

- *(Sentence 1)* If **a**, then **s**, or **o**.
- *(Sentence 2)* If **s**, then, if **o**, not **d**.
- *(Sentence 3)* However, **d**.
- *(Sentence 4)* Therefore, not **a**.

More elementary statements:

(**a**) The autonomous car caused the accident

(**s**) The accident occurred due to a sensor malfunction

(**o**) The driver overrode the autonomous driving

(**d**) The damage was significant

Propositional Variables

(**a**) The autonomous car caused the accident

(**s**) The accident occurred due to a sensor malfunction

(**o**) The driver overrode the autonomous driving

(**d**) The damage was significant

Propositional Variables

(**a**) The autonomous car caused the accident

(**s**) The accident occurred due to a sensor malfunction

(**o**) The driver overrode the autonomous driving

(**d**) The damage was significant

Propositional **signature** P: set of atomic statements called propositional **variables**

Example: $P = \{\mathbf{a},\ \mathbf{s},\ \mathbf{o},\ \mathbf{d}\}$

We speak of variables because they take two possible truth values: **True** (T, 1) or **False** (F, 0)

Formulae

We connect the variables to obtain formulae.

A formula over a signature P is a sequence of symbols from $P \cup \{\Rightarrow, \neg, \wedge, \vee\}$ such that

1. Propositional variables are formulae
2. If φ and ψ are formulae, so are $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \Rightarrow \psi$ and $\neg\varphi$

We write $\mathit{Fml}(P)$ for the set of formulae over P

Example:

- *(Sentence 1)* If **a**, then **s**, or **o** $\quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o})$
- *(Sentence 2)* If **s**, then, if **o**, not **d** $\quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg\mathbf{d})$
- *(Sentence 3)* However, **d** $\quad \mathbf{d}$
- *(Sentence 4)* Therefore, not **a** $\quad \neg\mathbf{a}$

Propositional Logic Puzzle – Exercise

A group of friends is trying to decide who ate the last slice of pizza. Here's what they claim:

- Alice: "If Bob didn't eat it, then I did."
- Bob: "If I ate it, then Charlie didn't."
- Charlie: "I didn't eat it, but either Alice or Bob did."

Write their claim in propositional logic with the following variables:

(A) Alice ate the pizza.

(B) Bob ate the pizza.

(C) Charlie ate the pizza.

Propositional Logic Puzzle – Solution

- Alice: "If Bob didn't eat it, then I did."
- Bob: "If I ate it, then Charlie didn't."
- Charlie: "I didn't eat it, but either Alice or Bob did."

Propositional Logic Puzzle – Solution

- Alice: "If Bob didn't eat it, then I did."
  $\neg \mathbf{B} \Rightarrow \mathbf{A}$
- Bob: "If I ate it, then Charlie didn't."
  $\mathbf{B} \Rightarrow \neg \mathbf{C}$
- Charlie: "I didn't eat it, but either Alice or Bob did."
  $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

Propositional Logic Puzzle – Solution

- Alice: "If Bob didn't eat it, then I did."
  $\neg \mathbf{B} \Rightarrow \mathbf{A}$
- Bob: "If I ate it, then Charlie didn't."
  $\mathbf{B} \Rightarrow \neg \mathbf{C}$
- Charlie: "I didn't eat it, but either Alice or Bob did."
  $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
- Someone took the last slice.
  $\mathbf{A} \vee \mathbf{B} \vee \mathbf{C}$
- Only one of them took it.
  $\neg(\mathbf{A} \wedge \mathbf{B}) \wedge \neg(\mathbf{A} \wedge \mathbf{C}) \wedge \neg(\mathbf{B} \wedge \mathbf{C})$

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

1. What is a propositional formula? ✓
2. What is a satisfiable formula?

Is this Argument Convincing?

- If the autonomous car caused the accident, then the accident occurred due to a sensor malfunction, or the driver overrode the autonomous driving.
- If the accident occurred due to a sensor malfunction, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- However, the damage was significant.
- Therefore, the autonomous car did not cause the accident.

Semantics

How to study the truth content of the argument?

Truth Values

An **interpretation** over a signature P is a mapping $I : P \to \mathbb{B} = \{T, F\}$

Truth Values

An **interpretation** over a signature P is a mapping
$I : P \to \mathbb{B} = \{T, F\}$

For an interpretation I, a **valuation** over P is defined by
$I^* : Fml(P) \to \mathbb{B}$ extending I with the following truth tables

$I^*(\varphi)$	$I^*(\psi)$	$I^*(\varphi \wedge \psi)$	$I^*(\varphi \vee \psi)$	$I^*(\varphi \Rightarrow \psi)$	$I^*(\neg\varphi)$
T	T	T	T	T	F
T	F	F	T	F	F
F	T	F	T	T	T
F	F	F	F	T	T

Truth Table: Exercise

Evaluate the truth table for the expression: $\neg\mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

Help:

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \Rightarrow \psi$	$\neg\varphi$
T	T	T	T	T	F
T	F	F	T	F	F
F	T	F	T	T	T
F	F	F	F	T	T

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
T	T	T			

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
T	T	T	T		

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
T	T	T	T	F	

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
T	*T*	*T*	*T*	*F*	*F*

Truth Table: Solution

Evaluate the truth table for the expression: $\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$

A	**B**	**C**	$\mathbf{A} \vee \mathbf{B}$	$\neg \mathbf{C}$	$\neg \mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B})$
T	T	T	T	F	F
T	T	F	T	T	T
T	F	T	T	F	F
T	F	F	T	T	T
F	T	T	T	F	F
F	T	F	T	T	T
F	F	T	F	F	F
F	F	F	F	T	F

Logical Equivalences

Two propositional formulae φ and ψ are logically equivalent, written $\varphi \equiv \psi$, if they have the same truth tables.

Examples:

De Morgan's Laws:

- $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
- $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

Double Negation: $\neg(\neg P) \equiv P$

Implication: $P \Rightarrow Q \equiv \neg P \vee Q$

Distributivity: $(P \vee (Q \wedge R)) \equiv (P \vee Q) \wedge (P \vee R)$

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T					
T	F					
F	T					
F	F					

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T				
T	F	F				
F	T	F				
F	F	F				

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	F			
T	F	F	T			
F	T	F	T			
F	F	F	T			

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	F	F		
T	F	F	T	F		
F	T	F	T	T		
F	F	F	T	T		

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	F	F	F	
T	F	F	T	F	T	
F	T	F	T	T	F	
F	F	F	T	T	T	

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Exercise: Logical Equivalence

Verify that $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ using a truth table.

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
T	T	T	**F**	F	F	**F**
T	F	F	**T**	F	T	**T**
F	T	F	**T**	T	F	**T**
F	F	F	**T**	T	T	**T**

The columns for $\neg(P \wedge Q)$ and $\neg P \vee \neg Q$ are identical, proving the equivalence.

Redifining Connectives

Logical equivalences can also be used as definition.

Consider formulae defined using only $\vee$ (disjunction) and $\neg$ (negation). We can define the other connectives as "syntactic sugar":

- $\wedge$ (conjunction): $P \wedge Q := \neg(\neg P \vee \neg Q)$
- $\Rightarrow$ (implication): $P \Rightarrow Q := \neg P \vee Q$

Redifining Connectives

Logical equivalences can also be used as definition.

Consider formulae defined using only $\vee$ (disjunction) and $\neg$ (negation). We can define the other connectives as "syntactic sugar":

- $\wedge$ (conjunction): $P \wedge Q := \neg(\neg P \vee \neg Q)$
- $\Rightarrow$ (implication): $P \Rightarrow Q := \neg P \vee Q$

Similarly, we can define

- $\top$ (the formula that is always true): $\top := P \vee \neg P$
- $\bot$ (the formula that is always false): $\bot := \neg\top$

Models

Given a formula φ

- A **model** of φ is an interpretation I over P that make φ true: $I^*(\varphi) = T$

Models

Given a formula φ

- A **model** of φ is an interpretation I over P that make φ true: $I^*(\varphi) = T$
- The formula φ is **satisfiable** if it admits a model

Models

Given a formula φ

- A **model** of φ is an interpretation I over P that make φ true: $I^*(\varphi) = T$
- The formula φ is **satisfiable** if it admits a model
- The formula φ is **valid** (or a **tautology**) if every interpretation I over P is a model of φ

Models

Given a formula φ

- A **model** of φ is an interpretation I over P that make φ true: $I^*(\varphi) = T$
- The formula φ is **satisfiable** if it admits a model
- The formula φ is **valid** (or a **tautology**) if every interpretation I over P is a model of φ

Notation: we write $I \models \varphi$ if I is a model of φ.

Inductive Definition of Models

Based on the truth tables, we can define a model inductively:

- $I \models p$, with p a propositional variable if and only if $I(p) = T$

Inductive Definition of Models

Based on the truth tables, we can define a model inductively:

- $I \models p$, with p a propositional variable if and only if $I(p) = T$
- $I \models \neg\varphi$ if and only if $I \not\models \varphi$

Inductive Definition of Models

Based on the truth tables, we can define a model inductively:

- $I \models p$, with p a propositional variable if and only if $I(p) = T$
- $I \models \neg\varphi$ if and only if $I \not\models \varphi$
- $I \models \varphi \wedge \psi$ if and only if $I \models \varphi$ and $I \models \psi$

Inductive Definition of Models

Based on the truth tables, we can define a model inductively:

- $I \models p$, with p a propositional variable if and only if $I(p) = T$
- $I \models \neg\varphi$ if and only if $I \not\models \varphi$
- $I \models \varphi \wedge \psi$ if and only if $I \models \varphi$ and $I \models \psi$
- $I \models \varphi \vee \psi$ if and only if $I \models \varphi$ or $I \models \psi$

Inductive Definition of Models

Based on the truth tables, we can define a model inductively:

- $I \models p$, with p a propositional variable if and only if $I(p) = T$
- $I \models \neg\varphi$ if and only if $I \not\models \varphi$
- $I \models \varphi \wedge \psi$ if and only if $I \models \varphi$ and $I \models \psi$
- $I \models \varphi \vee \psi$ if and only if $I \models \varphi$ or $I \models \psi$

When looking for a model of a formula, we can decompose the formula into smaller parts and check each part separately!

Decidability

Theorem

The three notions – tautology, model, and satisfiability – are **decidable**, i.e., there exists a decision algorithm that answers yes or no for each of these notions.

It suffices to check the truth table of the formula.

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

1. What is a propositional formula? ✓
2. What is a satisfiable formula? ✓

Is this Argument Convincing?

- If the autonomous car caused the accident, then the accident occurred due to a sensor malfunction, or the driver overrode the autonomous driving.
- If the accident occurred due to a sensor malfunction, then, if the driver overrode the autonomous driving, less significant damage would have resulted.
- However, the damage was significant.
- Therefore, the autonomous car did not cause the accident.

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T			

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T		

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T	F	

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T	F	F

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T	F	F
T	T	T	F	T	F	F

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T	F	F
T	T	T	F	T	F	F
T	T	F	T	T	F	F
T	T	F	F	T	F	F
T	F	T	T	F	F	F
T	F	T	F	F	F	F
T	F	F	T	F	F	F
T	F	F	F	F	F	F

Is this Argument Convincing?

$$\varphi \quad := \quad \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \quad \wedge \quad \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}) \quad \wedge \quad \mathbf{d} \quad \wedge \quad \neg \mathbf{a}$$

Truth table for the formula:

a	**s**	**o**	**d**	$(\mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}))$	$(\mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg \mathbf{d}))$	φ
T	T	T	T	T	F	F
T	T	T	F	T	F	F
T	T	F	T	T	F	F
T	T	F	F	T	F	F
T	F	T	T	F	F	F
T	F	T	F	F	F	F
T	F	F	T	F	F	F
T	F	F	F	F	F	F
F	T	T	T	T	T	T

Thus, the argumentation is plausible!

SAT in practice

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Example (Hardness)

Try it yourself: `http://www.cs.utexas.edu/~marijn/game/`

Complexity

Complexity of SAT

Naive Algorithm

1. Enumerate all possible interpretation (entries in the truth table).
2. For each interpretation, compute the truth value (fill the entry in the table).
3. Stop as soon as a model is found.
4. If no model is found, the formula is unsatisfiable.

Complexity of SAT

Naive Algorithm

1. Enumerate all possible interpretation (entries in the truth table).
2. For each interpretation, compute the truth value (fill the entry in the table).
3. Stop as soon as a model is found.
4. If no model is found, the formula is unsatisfiable.

⚠ The truth table of a formula with n variables has 2^n entries, i.e., exponential in the number of propositional variables.

How complex is SAT?

What is the Complexity of a Problem?

What is the Complexity of a Problem?

The difficulty of solving it.

How do we measure that?

What is the Complexity of a Problem?

The difficulty of solving it.

How do we measure that?

- Implement an algorithm and analyze its runtime.
- But are we analyzing the **algorithm** or the **problem** itself?

What is the Complexity of a Problem?

The difficulty of solving it.

How do we measure that?

- Implement an algorithm and analyze its runtime.
- But are we analyzing the **algorithm** or the **problem** itself?

Key Question

Can we define complexity independently of a specific algorithm/implementation?

A Formal Approach

. . . to compare problems objectively and understand the inherent difficulties.

What do we need?

A Formal Approach

. . . to compare problems objectively and understand the inherent difficulties.

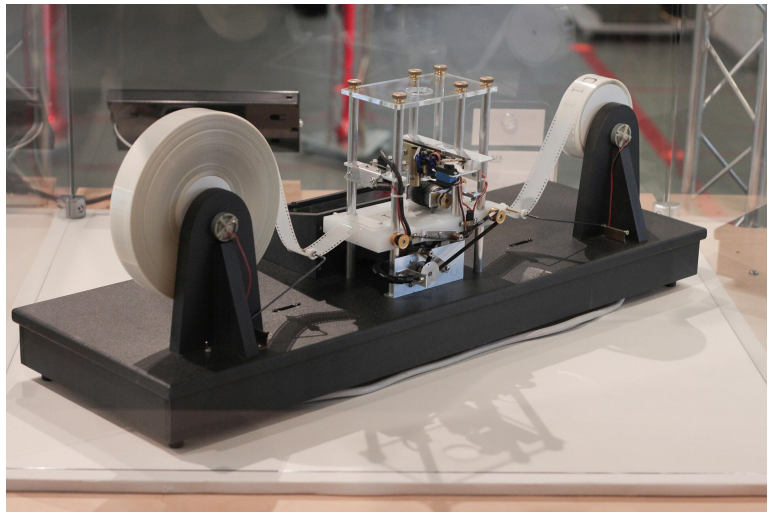
What do we need?

- A mathematical model of computation.
- A framework to classify problems.

Turing Machines

A Turing Machine

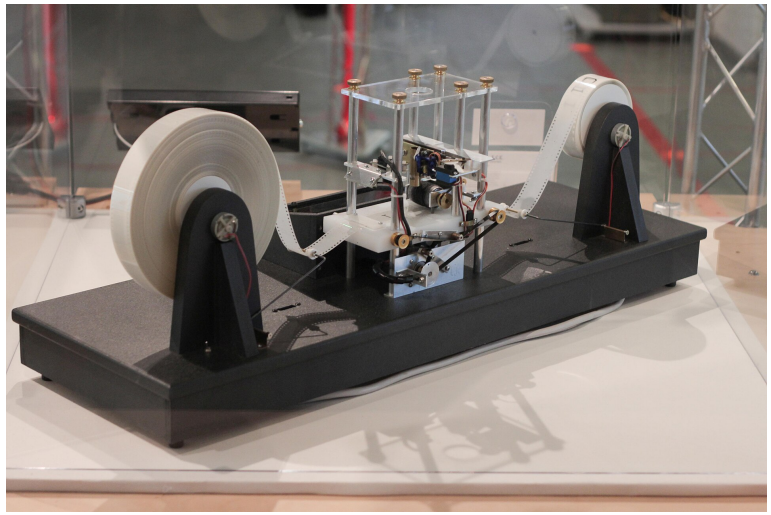
- is a mathematical model of computation (like an abstract computer),
- can simulate any algorithm,
- provides a definition of what it means for a function to be computable.

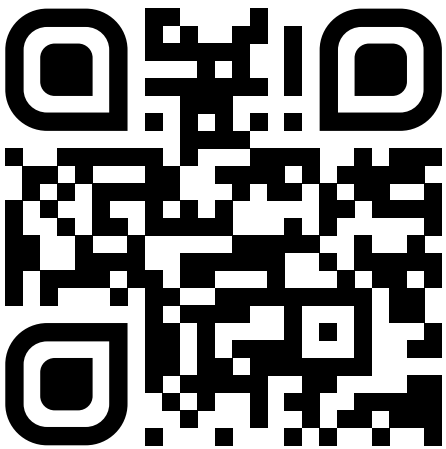


Turing Machines

A Turing Machine

- is a mathematical model of computation (like an abstract computer),
- can simulate any algorithm,
- provides a definition of what it means for a function to be computable.





Example (An Abstract Computer)

Try it yourself:
`https://turingmachine.io/`

P, NP, and NP-Complete Problems

P: Polynomial Time
Problems in P can be solved by a deterministic Turing Machine in polynomial time.
Example: Sorting a list of numbers.

NP: Nondeterministic Polynomial Time
Problems in NP can be verified by a deterministic Turing Machine in polynomial time or solved by a nondeterministic Turing Machine in polynomial time.
Example: Solving a Sudoku.

NP-Complete
A problem is NP-complete if it is in NP and every problem in NP can be reduced to it in polynomial time.
Example: The SAT problem.

SAT is NP-complete (Cook-Levin Theorem)

- SAT is in NP
  Proof: solution can be checked in polynomial time
- Every problem in NP can be reduced to SAT in polynomial time
  Proof: encode the run of a non-deterministic Turing machine as a formula

Consequences of NP-completeness of SAT

Relationship Between P, NP, and NP-Complete

- **P** is a subset of **NP**.
- NP-complete problems are a subset of **NP**.
- It is unknown whether **P = NP**.

Consequences of NP-completeness of SAT

Relationship Between P, NP, and NP-Complete

- **P** is a subset of **NP**.
- NP-complete problems are a subset of **NP**.
- It is unknown whether **P** = **NP**.

We do not have a polynomial algorithm for SAT 😐

Consequences of NP-completeness of SAT

Relationship Between P, NP, and NP-Complete

- **P** is a subset of **NP**.
- NP-complete problems are a subset of **NP**.
- It is unknown whether **P = NP**.

We do not have a polynomial algorithm for SAT 😐

If $P \neq NP$, we will never have a polynomial algorithm for SAT ☹

Consequences of NP-completeness of SAT

Relationship Between P, NP, and NP-Complete

- **P** is a subset of **NP**.
- NP-complete problems are a subset of **NP**.
- It is unknown whether **P = NP**.

We do not have a polynomial algorithm for SAT 😐

If $P \neq NP$, we will never have a polynomial algorithm for SAT ☹

All known NP-complete algorithms have exponential runtime ☹

SAT Solvers

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

1. How can we practically solve the SAT problem for a given formula?

Historic Landmarks

- 1960: DP Algorithm (first SAT solving algorithm)
- 1962: DPLL Algorithm (improving upon DP algorithm)
- 1971: SAT is NP-Complete
- 1992: The First International SAT Competition (followed by 1993, 1996, since 2002 every year)
- 1996: The First International SAT Conference (followed by 1998, since 2000 every year)
- Nowadays, SAT solvers based on deep learning

Advances

From 1992: **100** variables and **200** clauses.
To 2024: **21,000,000** variables and **96,000,000** clauses.

SAT Conference 2024

SAT Solvers

SAT solvers are algorithms or software tools designed to efficiently solve the SAT problem: sat, unsat (or timeout).

Efficiently?

SAT Solvers

SAT solvers are algorithms or software tools designed to efficiently solve the SAT problem: sat, unsat (or timeout).

Efficiently? General Principle

- Transforms the input formula into a formula in a standard form (*normal form*)
- Explores the different candidate models
- Uses search and propagation techniques to reduce the search space

Conjunctive Normal Form

Conjunctive Normal Form (CNF)

- A **CNF formula** is a conjunction ($\wedge$) of clauses.
- A **clause** is a disjunction ($\vee$) of literals.
- A **literal** is a variable x (positive literal) or its negation $\overline{x}$ (negative literal).

$$\bigwedge_i \bigvee_j \ell_{ij}$$

where ℓ_{ij} is the j-th literal in the i-th clause

CNF Example

$$\begin{aligned}
\varphi :=& (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1) \\
\text{variables}(\varphi) =& \{x_1, x_2, x_3\} \\
\text{litterals}(\varphi) =& \{x_1, \overline{x_1}, x_2, \overline{x_2}, x_3\} \\
\text{clauses}(\varphi) =& \{\{\overline{x_1}, x_2\}, \{\overline{x_1}, \overline{x_2}, x_3\}, \{x_1\}\}
\end{aligned}$$

A CNF formula is given as a set of clauses, i.e., clauses(φ), it is the standard input format for SAT solvers.

Satisfiability

A formula φ is **satisfiable** if there exists an interpretation of variables(φ) that satisfies φ.

An interpretation I satisfies

- a CNF formula if it satisfies all of its clauses
- a clause if it satisfies at least one of its literals
- a positive literal x if $I(x) = T$
- a negative literal $\overline{x}$ if $I(x) = F$

Exercise: Satisfiable or Unsatisfiable?

$$\varphi_1 := \{\{X\}\}$$

$$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$$

$$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$$

$$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$$

$$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$$

$$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$$

Exercise: Satisfiable or Unsatisfiable?

$\varphi_1 := \{\{X\}\}$ *sat*

$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$

$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$

$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$

$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$

$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$

Exercise: Satisfiable or Unsatisfiable?

$\varphi_1 := \{\{X\}\}$ *sat*

$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$ *unsat*

$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$

$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$

$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$

$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$

Exercise: Satisfiable or Unsatisfiable?

$\varphi_1 := \{\{X\}\}$ *sat*

$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$ *unsat*

$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$ *sat*

$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$

$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$

$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$

Exercise: Satisfiable or Unsatisfiable?

$\varphi_1 := \{\{X\}\}$ *sat*

$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$ *unsat*

$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$ *sat*

$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$ *unsat*

$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$

$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$

Exercise: Satisfiable or Unsatisfiable?

$\varphi_1 := \{\{X\}\}$ *sat*

$\varphi_2 := \{\{X\}, \{\overline{X}\}\}$ *unsat*

$\varphi_3 := \{\{X, Y, \overline{Z}\}\}$ *sat*

$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$ *unsat*

$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$ *unsat*

$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$

Exercise: Satisfiable or Unsatisfiable?

$$\varphi_1 := \{\{X\}\} \qquad \textit{sat}$$

$$\varphi_2 := \{\{X\}, \{\overline{X}\}\} \qquad \textit{unsat}$$

$$\varphi_3 := \{\{X, Y, \overline{Z}\}\} \qquad \textit{sat}$$

$$\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\} \qquad \textit{unsat}$$

$$\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\} \qquad \textit{unsat}$$

$$\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\} \qquad \textit{sat}$$

Conversion to CNF

Conversion to CNF

Use the logical equivalences to rewrite the formula:

1. Eliminate implications.
   $P \Rightarrow Q \equiv \neg P \vee Q$

Conversion to CNF

Use the logical equivalences to rewrite the formula:

1. Eliminate implications.
   $P \Rightarrow Q \equiv \neg P \vee Q$

2. Move negations inward using De Morgan's laws. (At this point, the formula is in negation normal form (NNF))
   $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$

Conversion to CNF

Use the logical equivalences to rewrite the formula:

1. Eliminate implications.
   $P \Rightarrow Q \equiv \neg P \vee Q$

2. Move negations inward using De Morgan's laws. (At this point, the formula is in negation normal form (NNF))
   $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$

3. Distribute disjunctions over conjunctions to achieve CNF.
   $(P \vee (Q \wedge R)) \equiv (P \vee Q) \wedge (P \vee R)$

Convert to CNF – Exercise

Convert the following formulae into CNF:

1. $(A \vee B) \Rightarrow C$
2. $\neg(P \wedge (Q \vee R))$
3. $\neg(X \Rightarrow Y) \vee \neg(Y \Rightarrow Z)$

Hints:

- Start by eliminating implications.
- Use De Morgan's laws to move negations inward.
- Distribute disjunctions over conjunctions to achieve CNF.

Convert to CNF – Solution 1

Convert $\psi_1 := (A \vee B) \Rightarrow C$ into CNF.

Convert to CNF – Solution 1

Convert $\psi_1 := (A \vee B) \Rightarrow C$ into CNF.

$$
\begin{aligned}
\psi_1 &\equiv \neg(A \vee B) \vee C \\
&\equiv (\neg A \wedge \neg B) \vee C \\
&\equiv (\neg A \vee C) \wedge (\neg B \vee C) \\
&=_{CNF} \{\{\overline{A}, C\}, \{\overline{B}, C\}\}
\end{aligned}
$$

Convert to CNF – Solution 1

Convert $\psi_1 := (A \vee B) \Rightarrow C$ into CNF.

$$
\begin{aligned}
\psi_1 &\equiv \neg(A \vee B) \vee C \\
&\equiv (\neg A \wedge \neg B) \vee C \\
&\equiv (\neg A \vee C) \wedge (\neg B \vee C) \\
&=_{CNF} \{\{\overline{A}, C\}, \{\overline{B}, C\}\} \ \textit{sat}
\end{aligned}
$$

Convert to CNF – Solution 2

Convert $\psi_2 := \neg(P \wedge (Q \vee R))$ into CNF.

Convert to CNF – Solution 2

Convert $\psi_2 := \neg(P \wedge (Q \vee R))$ into CNF.

$$
\begin{aligned}
\psi_2 &\equiv \neg P \vee \neg(Q \vee R) \\
&\equiv \neg P \vee (\neg Q \wedge \neg R) \\
&\equiv (\neg P \vee \neg Q) \wedge (\neg P \vee \neg R) \\
&=_{CNF} \{\{\overline{P}, \overline{Q}\}, \{\overline{P}, \overline{R}\}\}
\end{aligned}
$$

Convert to CNF – Solution 2

Convert $\psi_2 := \neg(P \wedge (Q \vee R))$ into CNF.

$$
\begin{aligned}
\psi_2 &\equiv \neg P \vee \neg(Q \vee R) \\
&\equiv \neg P \vee (\neg Q \wedge \neg R) \\
&\equiv (\neg P \vee \neg Q) \wedge (\neg P \vee \neg R) \\
&=_{CNF} \{\{\overline{P}, \overline{Q}\}, \{\overline{P}, \overline{R}\}\} \ \textit{sat}
\end{aligned}
$$

Convert to CNF – Solution 3

Convert $\psi_3 := \neg(X \Rightarrow Y) \vee \neg(Y \Rightarrow Z)$ into CNF.

Convert to CNF – Solution 3

Convert $\psi_3 := \neg(X \Rightarrow Y) \vee \neg(Y \Rightarrow Z)$ into CNF.

$$
\begin{aligned}
\psi_3 &\equiv \neg(X \vee \neg Y) \vee \neg(Y \vee \neg Z) \\
&\equiv (X \wedge \neg Y) \vee (Y \wedge \neg Z) \\
&\equiv (X \vee Y) \wedge (X \vee \neg Z) \wedge (\neg Y \vee Y) \wedge (\neg Y \vee \neg Z) \\
&\equiv (X \vee Y) \wedge (X \vee \neg Z) \wedge (\neg Y \vee \neg Z) \\
&=_{CNF} \{\{X, Y\}, \{X, \overline{Z}\}, \{\overline{Y}, \overline{Z}\}\}
\end{aligned}
$$

Convert to CNF – Solution 3

Convert $\psi_3 := \neg(X \Rightarrow Y) \vee \neg(Y \Rightarrow Z)$ into CNF.

$$
\begin{aligned}
\psi_3 &\equiv \neg(X \vee \neg Y) \vee \neg(Y \vee \neg Z) \\
&\equiv (X \wedge \neg Y) \vee (Y \wedge \neg Z) \\
&\equiv (X \vee Y) \wedge (X \vee \neg Z) \wedge (\neg Y \vee Y) \wedge (\neg Y \vee \neg Z) \\
&\equiv (X \vee Y) \wedge (X \vee \neg Z) \wedge (\neg Y \vee \neg Z) \\
&=_{CNF} \{\{X, Y\}, \{X, \overline{Z}\}, \{\overline{Y}, \overline{Z}\}\} \ \textit{sat}
\end{aligned}
$$

SAT solvers are algorithms or software tools designed to efficiently solve the SAT problem: sat, unsat (or timeout).

Efficiently? General Principle

- Transforms the input formula into a formula in a standard form (*normal form*) ✓
- Explores the different candidate models
- Uses search and propagation techniques to reduce the search space

Hands-On Exercise

DIMACS Format for CNF

- DIMACS is a standard format for representing CNF formulae.
- **Structure**:
  - Each clause is a line of integers.
  - Positive integers represent variables.
  - Negative integers represent negated variables.
  - Each clause ends with a ‘0‘.
- **Example**:
  - Formula: $(A \vee \neg B) \wedge (B \vee C)$
  - DIMACS:

```
p cnf 3 2
1 -2 0
2 3 0
```

Tools

Install MiniSat:

```
sudo apt install minisat
```

Alternatively, you can use a brower-based SAT solver:

`https://www.msoos.org/cryptominisat/`

Try it out with the following formula:

```
p cnf 3 2
1 -2 0
2 3 0
```

Hands-On: Solve a SAT Problem

Check the various examples we have seen so far:

- $\varphi_1 := \{\{X\}\}$
- $\varphi_2 := \{\{X\}, \{\overline{X}\}\}$
- $\varphi_3 := \{\{X, Y, \overline{Z}\}\}$
- $\varphi_4 := \{\{X\}, \{\overline{Y}\}, \{Y, \overline{X}\}\}$
- $\varphi_5 := \{\{X, Y\}, \{\overline{X}, Y\}, \{X, \overline{Y}\}, \{\overline{X}, \overline{Y}\}\}$
- $\varphi_6 := \{\{\overline{X}, Y\}, \{\overline{X}, \overline{Y}, Z\}, \{X\}\}$
- $\psi_1 := \{\{\overline{A}, C\}, \{\overline{B}, C\}\}$
- $\psi_2 := \{\{\overline{P}, \overline{Q}\}, \{\overline{P}, \overline{R}\}\}$
- $\psi_3 := \{\{X, Y\}, \{X, \overline{Z}\}, \{\overline{Y}, \overline{Z}\}\}$

Hands-On: Solve a SAT Problem (2)

Check the car argumentation:

$$\varphi_{car} := \mathbf{a} \Rightarrow (\mathbf{s} \vee \mathbf{o}) \wedge \mathbf{s} \Rightarrow (\mathbf{o} \Rightarrow \neg\mathbf{d}) \wedge \mathbf{d} \wedge \neg\mathbf{a}$$

Check the pizza example:

$$\begin{aligned}\varphi_{pizza} := \quad & \neg\mathbf{B} \Rightarrow \mathbf{A} \\ \wedge \; & \mathbf{B} \Rightarrow \neg\mathbf{C} \\ \wedge \; & \neg\mathbf{C} \wedge (\mathbf{A} \vee \mathbf{B}) \\ \wedge \; & \mathbf{A} \vee \mathbf{B} \vee \mathbf{C} \\ \wedge \; & \neg(\mathbf{A} \wedge \mathbf{B}) \wedge \neg(\mathbf{A} \wedge \mathbf{C}) \wedge \neg(\mathbf{B} \wedge \mathbf{C})\end{aligned}$$

Hint, you need to convert the formulae into CNF first.

Outline

Recap

What We Have Learned So Far

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Satisfiability means that there exists an interpretation of the variables that makes the formula true.

Logical equivalences can be used to transform formulae into a standard form (for instance CNF).

Conjunctive Normal Form (CNF): a conjunction of clauses, where each clause is a disjunction of literals. This is the standard input format for SAT solvers.

SAT Solving Algorithms

Any idea?

Any idea?

Construct a valuation step by step, considering partial valuations $[p \backslash b]$ (or $[p \leftarrow b]$) with $b \in \mathbb{B} = \{F, T\}$, completed step by step until the satisfiability of the initial formula (or a contradiction) is determined.

Partial Valuations

Let C be a clause $\{l_1, \ldots, l_n\}$ with l_i as literals, let p be a propositional variable, and $b \in \mathbb{B}$ a Boolean value.

$C[p \backslash b]$ is the propositional formula:

- $\top$ if $p \in C$ and $b = 1$,
- $\top$ if $\neg p \in C$ and $b = 0$,
- $C \backslash \neg p$ if $\neg p \in C$ and $b = 1$,
- $C \backslash p$ if $p \in C$ and $b = 0$,
- C if $p \notin C$ and $\neg p \notin C$.

Example and Exercise

Let φ be the formula $(x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$.

With the partial valuation $[x\backslash 1]$ and $[x\backslash 0]$, we obtain
$\varphi[x\backslash 1] = (x \vee y \vee z)[x\backslash 1] \wedge (x \vee \neg y \vee \neg z)[x\backslash 1] = \top \wedge \top \equiv \top$
$\varphi[x\backslash 0] = (y \vee z) \wedge (\neg y \vee \neg z)$

Example and Exercise

Let φ be the formula $(x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$.

With the partial valuation $[x\backslash 1]$ and $[x\backslash 0]$, we obtain
$\varphi[x\backslash 1] = (x \vee y \vee z)[x\backslash 1] \wedge (x \vee \neg y \vee \neg z)[x\backslash 1] = \top \wedge \top \equiv \top$
$\varphi[x\backslash 0] = (y \vee z) \wedge (\neg y \vee \neg z)$

Exercise: what are the partial valuations of φ with $[y\backslash 1]$ and $[y\backslash 0]$?

Example and Exercise

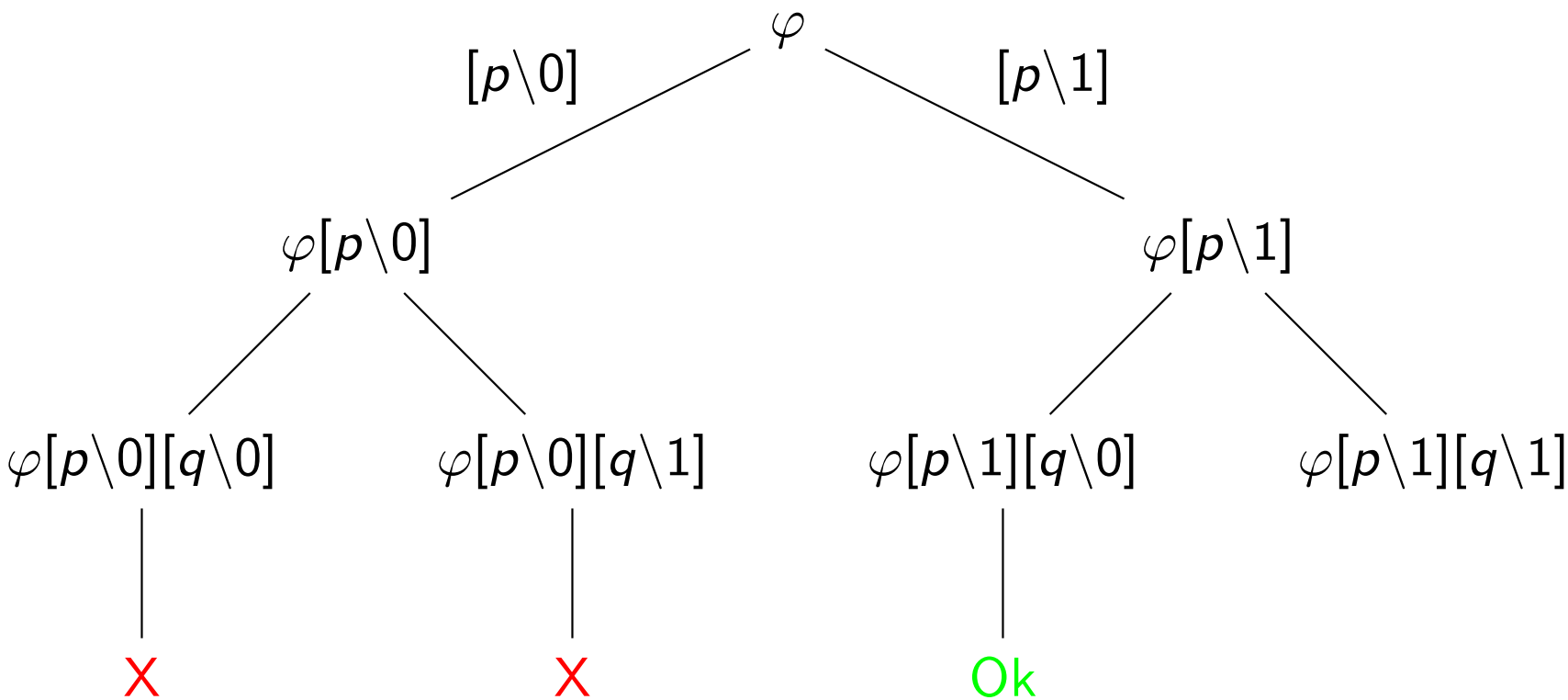
Let φ be the formula $(x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$.

With the partial valuation $[x\backslash 1]$ and $[x\backslash 0]$, we obtain
$\varphi[x\backslash 1] = (x \vee y \vee z)[x\backslash 1] \wedge (x \vee \neg y \vee \neg z)[x\backslash 1] = \top \wedge \top \equiv \top$
$\varphi[x\backslash 0] = (y \vee z) \wedge (\neg y \vee \neg z)$

Exercise: what are the partial valuations of φ with $[y\backslash 1]$ and $[y\backslash 0]$?

$\varphi[y\backslash 1] = (x \vee y \vee z)[y\backslash 1] \wedge (x \vee \neg y \vee \neg z)[y\backslash 1] = (x \vee \neg z)$
$\varphi[y\backslash 0] = (x \vee z)$

Traversal with *backtrack*



φ in CNF

$$\varphi = C_1 \wedge \ldots \wedge C_k$$

What is the value of $\varphi[p \backslash 1]$?

φ in CNF

$$\varphi = C_1 \wedge \ldots \wedge C_k$$

What is the value of $\varphi[p \backslash 1]$?

If $p \in C_i$, then C_i is satisfied, and we can remove it from φ.
If $\neg p \in C_i$, then C_i is simplified to $C_i \backslash \neg p$.

φ in CNF

$$\varphi = C_1 \wedge \ldots \wedge C_k$$

What is the value of $\varphi[p\backslash 1]$?

If $p \in C_i$, then C_i is satisfied, and we can remove it from φ.
If $\neg p \in C_i$, then C_i is simplified to $C_i \backslash \neg p$.

It suffices to

- remove all clauses C_i containing p,
- remove $\neg p$ from all clauses C_i containing $\neg p$.

(Symmetrical situation for $\varphi[p\backslash 0]$)

Pseudocode for backtracking

```
backtrack(φ):
   if φ = ∅ :  return sat
   if ∅ ∈ φ :  return unsat
   p = pickVariable(φ)
   return backtrack(φ[p\0]) or backtrack(φ[p\1])
```

Heuristics for picking a variable:

- Pick the variable that occurs most frequently in the clauses.
- Pick the variable that occurs in the most clauses.
- Pick the variable that occurs in the fewest clauses.

An observation

Consider the following CNF formula:

$$\begin{aligned}\varphi :=&(p_1 \vee \neg p_3 \vee \neg p_5) \wedge (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_3 \vee p_4) \\ &\wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_4 \vee \neg p_2)\end{aligned}$$

Suppose that we start by choosing to assign T to p_1. This leaves us with:

$$\varphi' := (p_2) \wedge (\neg p_3 \vee p_4) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_4 \vee \neg p_2)$$

An observation

$$\varphi' := (p_2) \wedge (\neg p_3 \vee p_4) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_4 \vee \neg p_2)$$

The clause $\neg p_1 \vee p_2$ is now simply p_2.

Any satisfying interpretation must assign T to p_2: there is no choice to make given this formula. We say that p_2 is a **unit clause**: there is no other literal in the clause.

We set p_2 to T, and simplify the formula further:

$$\varphi'' := (\neg p_3 \vee p_4) \wedge (p_3) \wedge (\neg p_4)$$

An observation

$$\varphi'' := (\neg p_3 \vee p_4) \wedge (p_3) \wedge (\neg p_4)$$

We have two unit literals p_3 and $\neg p_4$. We can continue, pick p_3, assigning it T, and simplify:

$$\varphi''' := (p_4) \wedge (\neg p_4)$$

Now all clauses are unit, and there is no solution to obtain a model for φ'''.

An observation

If we assign p_1 to T then the resulting formula is not satisfiable.

Once we assigned p_1 to T, we were able to determine that the resulting formula was unsatisfiable without making any further decisions.

All of the resulting simplifications were a logical consequence of this original choice.

The process of carrying this to its conclusion is called **unit propagation**.

Unit Propagation

A **unit clause** is a clause with only one unassigned literal.

Unit Propagation:

- If a unit clause is found, the solver assigns the value that satisfies the clause and simplifies the formula accordingly.
- This process continues until a fixpoint: no more unit clauses can be found or a contradiction is reached.

Pure Literal

If x occurs only positively (or only negatively), then it is a **pure litteral**. It can be fixed to the respective value.

Why?

Pure Literal

If x occurs only positively (or only negatively), then it is a **pure litteral**. It can be fixed to the respective value.

Why?

If there is a model that where the x is assigned the opposite value, then flipping the assignment of x still yields a model.

Example

$$\varphi := (x \vee \neg y) \wedge (y \vee \neg z) \wedge (\neg z \vee \neg w)$$

x is a positive pure litteral
$\varphi[x \backslash 1] = (y \vee \neg z) \wedge (\neg z \vee \neg w)$

Example

$$\varphi := (x \vee \neg y) \wedge (y \vee \neg z) \wedge (\neg z \vee \neg w)$$

x is a positive pure litteral
$\varphi[x\backslash 1] = (y \vee \neg z) \wedge (\neg z \vee \neg w)$

y is a positive pure litteral
$\varphi[x\backslash 1][y\backslash 1] = (\neg z \vee \neg w)$

Example

$$\varphi := (x \vee \neg y) \wedge (y \vee \neg z) \wedge (\neg z \vee \neg w)$$

x is a positive pure litteral
$\varphi[x\backslash 1] = (y \vee \neg z) \wedge (\neg z \vee \neg w)$

y is a positive pure litteral
$\varphi[x\backslash 1][y\backslash 1] = (\neg z \vee \neg w)$

z is a negative pure litteral
$\varphi[x\backslash 1][y\backslash 1][z\backslash 0] = \top$

Davis Putnam Logemann Loveland (DPLL) Algorithm (Davis et al., 1962)

```
dpll(φ):
   if φ = ∅ :  return sat
   if ∅ ∈ φ :  return unsat
   if {l} ∈ φ :  return dpll(φ[l\1])
   if pure − literal(p, b, φ) :  return dpll(φ[p\b])
   p = pickVariable(φ)
   return dpll(φ[p\0]) or dpll(φ[p\1])
```

DPLL Algorithm

Properties

- DPLL **always terminates**
  - Each recursion eliminates one variable
  - Worst case: **binary tree search** of depth $|V|$
- DPLL is **sound and complete**
  - If clause set S is sat, we eventually find a satisfying valuation.
  - If clause set S is unsat, the entire space of (partial) variable assignments is searched (but **variable selection still matters**!)
- Space complexity: **linear**!

Tseitin's Transformation

A Naive CNF Conversion Algorithm

1. Convert to Negation Normal Form (NNF)
2. Apply distributive laws to get CNF

⚠ Applying the distributive laws may result in an exponential blow-up.

For example, the formula

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

gets converted to

$$(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$$

which contains 2^n clauses.

Equisatisfiability

Two formulas φ and ψ are equisatisfiable, if φ is satisfiable if and only if ψ is.

Example: p and $p \wedge q$ are equisatisfiable, but they are not equivalent.

⚠ The models of two equisatisfiable formulas do not necessarily share the same models.

Tseitin's Transformation

Idea: Introduce new propositional variables, which act as names for subformulae of the original formula.

Equisatisfiability

- $\mathcal{T}(\varphi)$ are φ equisatisfiable
- $\mathcal{T}(\varphi)$ is a CNF formula
- $\mathcal{T}(\varphi)$ has size linear in the size of φ

If the solver finds a satisfying assignment for $\mathcal{T}(\varphi)$, it can be used to find a satisfying assignment for φ by deleting the label variables.

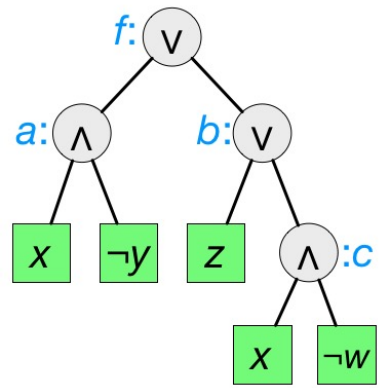
Example Tseitin's Transformation

$$\varphi := (x \wedge \neg y) \vee z \vee (x \wedge \neg w)$$

(Negation Normal Form)

$$\overset{\text{SAT}}{=} (c \leftrightarrow x \wedge \neg w) \wedge \cdots \wedge (f \leftrightarrow a \vee b) \wedge f$$

(Tseitin's Encoding)



- Define new variables:
  $d_a \leftrightarrow x \wedge \overline{y}, \quad d_\varphi \leftrightarrow a \vee b, \quad \ldots$
- Encode definitions in CNF:
  $(\overline{d_\varphi} \vee d_a \vee d_b) \wedge (d_\varphi \vee \overline{d_a}) \wedge (d_\varphi \vee \overline{d_b}) \wedge \ldots$
- One additional clause d_φ to assert that φ must be true

Tseitin's Transformation

The Tseitin's Transformation $\mathcal{T}(\varphi)$ of a propositional formula φ over connectives $\{\wedge, \vee, \neg\}$ is specified as follows.

$$\mathcal{T}(\varphi) = d_\varphi \wedge \mathcal{T}^*(\varphi) \qquad \text{(Root Formula)}$$

$$\mathcal{T}^*(\varphi) = \begin{cases} \mathcal{T}_{\mathsf{def}}(\varphi) \wedge \mathcal{T}^*(\psi) \wedge \mathcal{T}^*(\theta), & \text{if } \varphi = \psi @ \theta \text{ and } @ \in \{\wedge, \vee\} \\ \mathcal{T}_{\mathsf{def}}(\varphi) \wedge \mathcal{T}^*(\psi), & \text{if } \varphi = \neg\psi \\ T, & \text{if } \varphi \in \mathcal{V} \end{cases}$$

(Recursion)

$$\mathcal{T}_{\mathsf{def}}(\varphi) = \begin{cases} (\overline{d_\varphi} \vee d_\psi) \wedge (\overline{d_\varphi} \vee d_\theta) \wedge (d_\varphi \vee \overline{d_\psi} \vee \overline{d_\theta}), & \text{if } \varphi = \psi \wedge \theta \\ (\overline{d_\varphi} \vee d_\psi \vee d_\theta) \vee (d_\varphi \vee \overline{d_\psi}) \wedge (d_\varphi \vee \overline{d_\theta}), & \text{if } \varphi = \psi \vee \theta \\ (\overline{d_\varphi} \vee \overline{d_\psi}) \wedge (d_\varphi \vee d_\psi), & \text{if } \varphi = \neg\psi \end{cases}$$

(Definitions)

Encodings

At-Most-One Constraints

Notation: AtMostOne$(x_1, \ldots, x_n)$ or $\leq 1\,(x_1, \ldots, x_n)$ or $\sum_i^n x_i \leq 1$

Not more than one literal from $x_1, \ldots, x_n$ is set to T.

Pairwise Encoding:

$\mathcal{E}\left[\leq 1\,(x_1, \ldots, x_n)\right] = \left\{\{\overline{x_i}, \overline{x_j}\} \mid 1 \leq i < j \leq n\right\}$

Size: $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ clauses

Cardinality Constraints

Notation: $\leq k\,(x_1, \ldots, x_n)$ or $\sum_i^n x_i \leq k$

Not more than k literals from $x_1, \ldots, x_n$ are set to T.

Direct Encoding:

$\mathcal{E}\left[\leq k\,(x_1, \ldots, x_n)\right] = \left\{\{\overline{x_{i_1}}, \ldots, \overline{x_{i_{k+1}}}\} \mid 1 \leq i_1 < \cdots < i_{k+1} \leq n\right\}$

Size: $\binom{n}{k+1}$ clauses[1]

[1] $\approx 2^n/\sqrt{n}$ by Stirling's Approx. for the worst case $k = \lceil n/2 \rceil$

Finite-Domain Variables

Common in combinatorial problems. Discrete, finite value domains: $x \in \{v_1, \ldots, v_n\}$

Relationships between them expressed as equality-formulas, e.g.: $x = v_3 \Rightarrow y \neq v_2$.

One-hot encoding:

- Boolean variables x_v: "x takes value v"
- Must encode that each variable takes exactly one value from its domain
  (by using at-least-one/at-most-one constraints)

Tradeoffs in Encodings

These encodings were simple and straightforward.

More complex encodings exist, which often rely on introducing auxiliary variables.

The use of auxiliary variables helps reduce the number of clauses, making the encoding more efficient for SAT solvers.

Some examples

Pythagorean Triples

Problem Definition

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then a, b and c do not all have the same color.

- Solution: Nope
- for $n = 7825$ it is not possible
- proof obtained by a SAT solver has 200 Terabytes – back then the largest Math proof yet

Pythagorean Triples

Problem Definition

Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then a, b and c do not all have the same color.

- Solution: Nope
- for $n = 7825$ it is not possible
- proof obtained by a SAT solver has 200 Terabytes – back then the largest Math proof yet

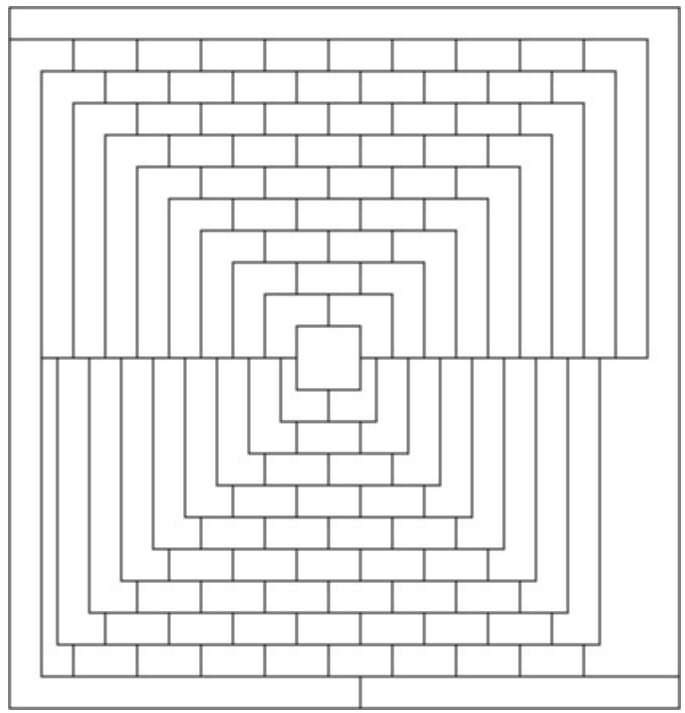
How to encode this?

- for each integer i we have a Boolean variable x_i, $x_i = 1$ if color of i is 1, $x_i = 0$ otherwise.
- for each a, b, c such that $a^2 + b^2 = c^2$ we have two clauses: $(x_a \vee x_b \vee x_c)$ and $(\overline{x_a} \vee \overline{x_b} \vee \overline{x_c})$

Graph Coloring

Example (McGregor Graph, 110 nodes, planar)

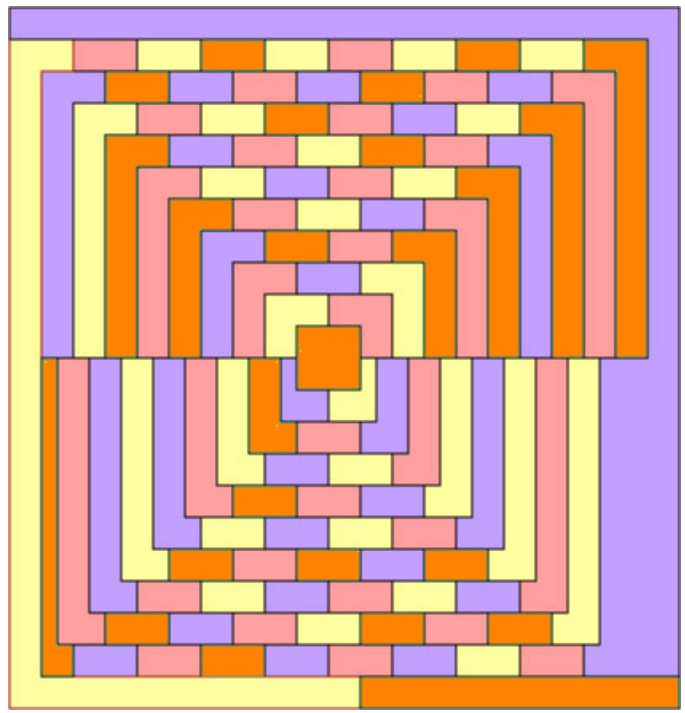
Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner's column "Mathematical Games")



Graph Coloring

Example (McGregor Graph, 110 nodes, planar)

Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner's column "Mathematical Games")



Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:

    $(\overline{u_j} \vee \overline{v_j})$ for $u, v \in E, 1 \leq j \leq k$

Graph Coloring: SAT Encoding

A k-coloring of a graph assigns one of k colors to each node, such that all adjacent nodes have a different color.

Graph Coloring Problem (GCP):

Input: An undirected graph $G = (V, E)$ and a number k.

Question: Is there a k-coloring for G?

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:

    $(\overline{u_j} \vee \overline{v_j})$ for $u, v \in E, 1 \leq j \leq k$
  - Suppress multiple colors for a node: At-most-one constraints

Graph Coloring: Example

- $V = \{u, v, w, x, y\}$
- Colors: red (=1), green (=2), blue (=3)
- Clauses:

  "every node gets a color"

  $(u_1 \vee u_2 \vee u_3)$

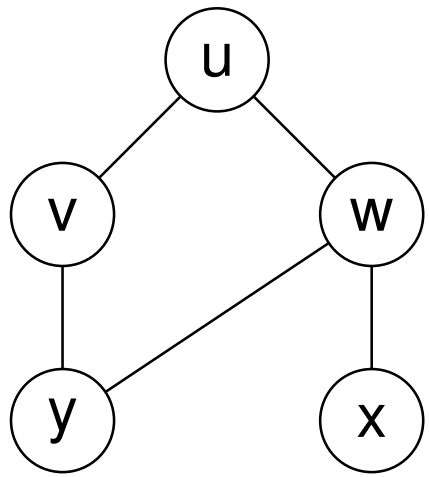
  $\vdots$

  $(y_1 \vee y_2 \vee y_3)$

  "adjacent nodes have different colors" $(\overline{u_1} \vee \overline{v_1}) \wedge \cdots \wedge (\overline{u_3} \vee \overline{v_3})$

  $\vdots$

  $(\overline{x_1} \vee \overline{y_1}) \wedge \cdots \wedge (\overline{x_3} \vee \overline{y_3})$



Graph Coloring: Example

- $V = \{u, v, w, x, y\}$
- Colors: red (=1), green (=2), blue (=3)
- Clauses:

  "every node gets a color"

  $(u_1 \vee u_2 \vee u_3)$

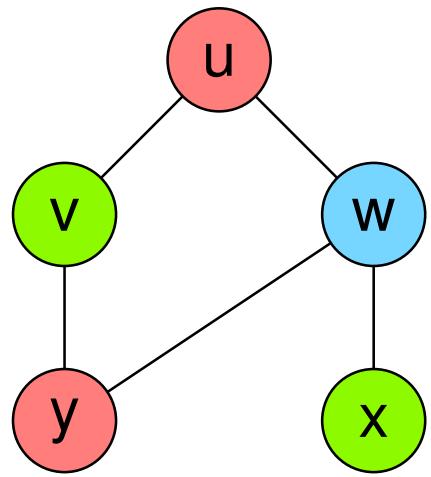
  $\vdots$

  $(y_1 \vee y_2 \vee y_3)$

  "adjacent nodes have different colors" $(\overline{u_1} \vee \overline{v_1}) \wedge \cdots \wedge (\overline{u_3} \vee \overline{v_3})$

  $\vdots$

  $(\overline{x_1} \vee \overline{y_1}) \wedge \cdots \wedge (\overline{x_3} \vee \overline{y_3})$



Conclusion

Conclusion

- SAT solving is a fundamental problem in computer science with applications in AI, verification, cryptography, and more.
- Despite being NP-complete, modern SAT solvers can handle large and complex instances efficiently.
- Techniques like CNF conversion, Tseitin's transformation, and advanced algorithms (e.g., DPLL, CDCL) make SAT solving practical.
- Encoding real-world problems into SAT is a powerful approach to tackle combinatorial challenges.

Takeaways

- SAT solvers are versatile tools for automated reasoning.
- Understanding the underlying algorithms and encodings is key to leveraging their power.
- The field continues to evolve, with ongoing research into optimization and new applications.

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Outputs:

- sat
- unsat
-
-

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Outputs:

- sat
- unsat
-
- timeout

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Outputs:

- sat
- unsat
- `Seg fault (core dumped)`
- timeout

SAT Problem:

Input: A propositional formula φ.

Question: Is φ satisfiable?

Outputs:

- Yes
- No
- Maybe
- I don't know

Hands-on

See `https://romainpascual.fr/teaching/sat/`.