Introduction to Propositional Satisfiability

IA085: Satisfiability and Automated Reasoning

Martin Jonáš

FI MUNI, Spring 2025

Contents

Propositional satisfiability (SAT)

- $(A \vee \neg B) \wedge (\neg A \vee C)$
- is it satisfiable?

Satisfiability modulo theories (SMT)

- $x = 1 \ \wedge \ x = y + y \ \wedge \ y > 0$
- is it satisfiable over reals?
- is it satisfiable over integers?

Automated theorem proving (ATP)

- axioms: $\forall x\,(x + x = 0)$, $\forall x \forall y\,(x + y = y + x)$
- do they imply $\forall x \forall y\,((x + y) + (y + x) = 0)$?

Contents

For each of the problems (SAT/SMT/ATP)

- necessary definitions and theoretical results
- algorithms to solve the problem
- usage in practice and practical considerations

Organization of the Course

Schedule and Requirements

During semester

- lecture every week (except May 7)
- seminar every other week
- project (write your own small SAT solver) – mandatory

Exam

- oral exam

Project

Implement your own SAT solver

- you can use any reasonable programming language (C, C++, C#, Go, Java, Python, Rust, . . .)
- you are encouraged to work in pairs (but you do not have to)
- technical requirements are specified in the information system
- more advanced features $\rightarrow$ bonus points for the exam
- the scores will be evaluated periodically through the semester, you will see the ranking

Who am I?

- author of SMT solver Q3B for quantified formulas over bit-vector theory
- for 3 years post-doctoral researcher in Fondazione Bruno Kessler: research focused on SMT-based verification of software and SAT-based verification of hardware
- PhD thesis about satisfiability of quantified formulas over bit-vector theory
- author of several research papers about solving SMT and using it in practice
- co-organizer of SMT-COMP 2024, 2025

Propositional Logic

Propositional logic

Propositional logic deals with propositions, their relationships, and arguments based on them.

Does not deal objects and their properties, just with separate atomic claims.

“Martin has brown hair”
“Martin does not have hair”
No relationship as far as propositional logic is concerned.

.

Propositional logic

Propositional logic deals with propositions, their relationships, and arguments based on them.

Does not deal objects and their properties, just with separate atomic claims.

"Martin has brown hair" (A)
"Martin does not have hair" (B)
No relationship as far as propositional logic is concerned.

.

Propositional logic

Propositional logic deals with propositions, their relationships, and arguments based on them.

Does not deal objects and their properties, just with separate atomic claims.

"Martin has brown hair" (A)
"Martin does not have hair" (B)
No relationship as far as propositional logic is concerned.

"Martin has brown hair"
if "Martin has brown hair" then "Martin does have hair"
Implies *"Martin does have hair"* .

Propositional logic

Propositional logic deals with propositions, their relationships, and arguments based on them.

Does not deal objects and their properties, just with separate atomic claims.

“Martin has brown hair” (A)
“Martin does not have hair” (B)
No relationship as far as propositional logic is concerned.

“Martin has brown hair” (A)
if “Martin has brown hair“ then “Martin does have hair” ($A \rightarrow B$)
Implies *“Martin does have hair”* (B).

Syntax

Let $V = \{A, B, C, \ldots\}$ be a countable set of propositional variables. The set of propositional formulas is defined inductively as

- $\top$ and $\bot$ are propositional formulas,
- v is a propositional formula for each $v \in V$ (called propositional atom),
- if φ is a propositional formula, $\neg\varphi$ is a propositional formula,
- if φ and ψ are propositional formulas, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, and $\varphi \leftrightarrow \psi$ are propositional formulas.

Example

- $A \wedge B$
- $(A \vee B) \leftrightarrow \neg C$

Formulas of form v or $\neg v$ are called literals.

Semantics: Truth Assignments

$Atoms(\varphi)$ = the set of all atoms of formula φ

(Total) truth assignment for formula φ

- assigns true ($\top$) or false ($\bot$) to each propositional variable in φ
- a function $\mu \colon V' \to \{\top, \bot\}$ where $Atoms(\varphi) \subseteq V'$
- can be written as a set of non-contradictory literals containing all variables of φ

Example

- formula $\varphi = A \vee B$,
- total assignment $\mu(A) = \top, \mu(B) = \bot$,
- written as $\mu = \{A, \neg B\}$.

Semantics: Satisfaction

Define when a truth assignment μ satisfies the formula φ, written $\mu \models \varphi$:

- $\mu \models \top$
- $\mu \models v$ if $\mu(v) = \top$
- $\mu \models \neg\psi$ if not $\mu \models \psi$
- $\mu \models \psi_1 \wedge \psi_2$ if $\mu \models \psi_1$ and $\mu \models \psi_2$
- $\mu \models \psi_1 \vee \psi_2$ if $\mu \models \psi_1$ or $\mu \models \psi_2$
- $\mu \models \psi_1 \rightarrow \psi_2$ if not $\mu \models \psi_1$ or $\mu \models \psi_2$
- $\mu \models \psi_1 \leftrightarrow \psi_2$ if $\mu \models \psi_1$ if and only if $\mu \models \psi_2$

Semantics: Model

If $\mu \models \varphi$, we say that μ is a model of φ

Example
$\{A, \neg B, C\}$ is a model of $A \wedge (B \leftrightarrow \neg C)$

An assignment μ is a partial model of φ if each extension of μ that is a truth assignment to φ (i.e., $Atoms(\varphi) \subseteq dom(\mu)$) is a model of φ

Example
$\{A, B\}$ is a partial model of $(A \wedge B) \vee (A \wedge C)$

Propositional Entailment

Formula φ propositionally entails formula ψ (written $\varphi \models \psi$) if every μ that is a truth assignment for both φ and ψ (i.e., $(Atoms(\varphi) \cup Atoms(\psi)) \subseteq dom(\mu)$) satisfies

$$\text{if } \mu \models \varphi \text{ then also } \mu \models \psi$$

Example

- $A \models A \vee B$
- $(A \rightarrow B) \wedge A \models B$
- $(A \vee B) \wedge (\neg A \vee C) \models (B \vee C)$
- $A \not\models A \wedge B$

Propositional Equivalence

Formulas φ and ψ are propositionally equivalent (written $\varphi \equiv \psi$) if

$$\varphi \models \psi \quad \text{and} \quad \psi \models \varphi$$

Example

- $A \land (B \lor C) \;\equiv\; (A \land B) \lor (A \land C)$
- $A \land (A \lor B) \;\equiv\; A$
- $\neg(A \land B) \;\equiv\; \neg A \lor \neg B$

Negation Normal Form (NNF)

- negations are applied only to propositional atoms
- the formula does not contain implication ($\rightarrow$) and equivalence ($\leftrightarrow$)

Transformation to NNF

1. rewrite all $\varphi \leftrightarrow \psi$ to $(\varphi \rightarrow \psi) \wedge (\varphi \leftarrow \psi)$
2. rewrite all $\varphi \rightarrow \psi$ to $\neg\varphi \vee \psi$
3. apply De Morgan rules until fixed point
   - rewrite $\neg(\varphi \wedge \psi)$ to $(\neg\varphi) \vee (\neg\psi)$
   - rewrite $\neg(\varphi \vee \psi)$ to $(\neg\varphi) \wedge (\neg\psi)$

Conversion to NNF: Complexity

What is the complexity of conversion to NNF?

$$\varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

Each equivalence doubles the size of the formula $\rightarrow$ translation can be exponential!

Conversion to NNF: Complexity

What is the complexity of conversion to NNF?

$$\varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

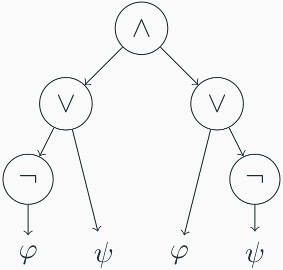
Each equivalence doubles the size of the formula $\rightarrow$ translation can be exponential!

Or is it? It depends on the representation of the formulas

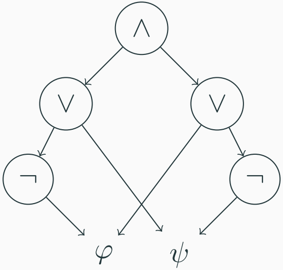
Representation of Formulas

$$(\neg\varphi \lor \psi) \land (\varphi \lor \neg\psi)$$

Tree



Directed acyclic graph (DAG)



In practice, we represent formulas as DAGS.

Conversion to NNF: Complexity

Theorem
When representing formulas as DAGs, the transformation to NNF is linear.

Proof idea.
The DAG contains two nodes for each subformula φ: one for φ, one for $\neg\varphi$. □

Conversion to NNF: Complexity

Theorem
When representing formulas as DAGs, the transformation to NNF is linear.

Proof idea.
The DAG contains two nodes for each subformula φ: one for φ, one for $\neg\varphi$. □

Proof details (bonus).
Recursively define function $NNF(\varphi) = (\varphi^+, \varphi^-)$. Given $NNF(\psi) = (\psi^+, \psi^-)$ and $NNF(\rho) = (\rho^+, \rho^-)$:

$$
\begin{aligned}
NNF(\psi \wedge \rho) &= (\psi^+ \wedge \rho^+,\ \psi^- \vee \rho^-) \\
NNF(\neg\psi) &= (\psi^-,\ \psi^+) \\
NNF(\psi \leftrightarrow \rho) &= (\underbrace{(\psi^- \vee \rho^+) \wedge (\psi^+ \vee \rho^-)}_{\text{positive}},\ \underbrace{(\psi^+ \wedge \rho^-) \vee (\psi^- \wedge \rho^+)}_{\text{negative}}).
\end{aligned}
$$

For more details, see *Property 1* in *Gabriele Masina, Giuseppe Spallitta, Roberto Sebastiani: On CNF Conversion for SAT Enumeration.* □

Conjunctive Normal Form

Clause

- disjunction of literals
- $A \vee \neg B \vee C$
- written as $\{A, \neg B, C\}$ thanks to idempotence, commutativity, and associativity
- what is $\{\}$?

Formula in Conjunctive Normal Form (CNF)

- conjunction of clauses
- $(A \vee \neg B \vee C) \wedge (B \vee \neg C) \wedge C$
- written as $\{\{A, \neg B, C\}, \{B, \neg C\}, \{C\}\}$ thanks to idempotence, commutativity, and associativity
- what are $\{\}$? and $\{\emptyset\}$?

Conjunctive Normal Form

- easy to represent (`clause = list[int]`, `formula = list[clause]`)
- easy to write algorithms, do not have to deal with the structure of the formula
- most of modern SAT solvers have input in CNF

Transformation to CNF (naive)

1. transform to NNF
2. apply distributivity until fixed point
   - rewrite $\varphi \vee (\psi \wedge \rho)$ to $(\varphi \vee \psi) \wedge (\varphi \vee \rho)$
   - rewrite $(\psi \wedge \rho) \vee \varphi$ to $(\psi \vee \varphi) \wedge (\rho \vee \varphi)$

This is again exponential, try with $\bigvee_{1 \leq i \leq n}(A_i \wedge B_i)$. ☹

Can we do better? What if the DAG representation is used? What if we use a different algorithm?

Theorem
There exists an infinite family of formulas $\Phi = \{\varphi_i \mid i \in \mathbb{N}\}$ *such that for each equivalent family of formulas with* $\varphi_i^{CNF} \equiv \varphi_i$*, the size* $|\varphi_i^{CNF}|$ *grows exponentially with respect to* $|\varphi_i|$ *(even for DAG representation).*

Conversion to CNF: Naive

Theorem
There exists an infinite family of formulas $\Phi = \{\varphi_i \mid i \in \mathbb{N}\}$ *such that for each equivalent family of formulas with* $\varphi_i^{CNF} \equiv \varphi_i$*, the size* $|\varphi_i^{CNF}|$ *grows exponentially with respect to* $|\varphi_i|$ *(even for DAG representation).*

Proof.
Let $parity_i(A_1, A_2, \ldots, A_i) = A_1 \oplus A_2 \oplus \ldots \oplus A_i$. We can show that

- $parity_i$ can be defined by a formula φ_i of size $\mathcal{O}(i)$,
- each formula φ_i^{CNF} in CNF that defines $parity_i$ has 2^{i-1} clauses. □

Conversion to CNF: Naive

Theorem
There exists an infinite family of formulas $\Phi = \{\varphi_i \mid i \in \mathbb{N}\}$ such that for each equivalent family of formulas with $\varphi_i^{CNF} \equiv \varphi_i$, the size $|\varphi_i^{CNF}|$ grows exponentially with respect to $|\varphi_i|$ (even for DAG representation).

Proof.
Let $parity_i(A_1, A_2, \ldots, A_i) = A_1 \oplus A_2 \oplus \ldots \oplus A_i$. We can show that

- $parity_i$ can be defined by a formula φ_i of size $\mathcal{O}(i)$,
- each formula φ_i^{CNF} in CNF that defines $parity_i$ has 2^{i-1} clauses. □

We cannot do better than exponential. ☹

Conversion to CNF: Naive

Theorem

There exists an infinite family of formulas $\Phi = \{\varphi_i \mid i \in \mathbb{N}\}$ *such that for each equivalent family of formulas with* $\varphi_i^{CNF} \equiv \varphi_i$*, the size* $|\varphi_i^{CNF}|$ *grows exponentially with respect to* $|\varphi_i|$ *(even for DAG representation).*

Proof.

Let $parity_i(A_1, A_2, \ldots, A_i) = A_1 \oplus A_2 \oplus \ldots \oplus A_i$. We can show that

- $parity_i$ can be defined by a formula φ_i of size $\mathcal{O}(i)$,
- each formula φ_i^{CNF} in CNF that defines $parity_i$ has 2^{i-1} clauses. □

We cannot do better than exponential. ☹

Or can we? Yes, we can! Later today.

Disjunctive Normal Form

Cube

- conjunction of literals
- $A \wedge \neg B \wedge C$

Formula in Disjunctive Normal Form (DNF)

- disjunction of cubes
- $(A \wedge \neg B \wedge C) \vee (B \wedge \neg C) \vee C$

We will not be dealing with DNF often in this course.

Propositional Satisfiability (SAT)

Satisfiability Problem

Problem (SAT)
Given a propositional formula, decide whether it is satisfiable.

Problem (CNF-SAT)
Given a propositional formula in CNF*, decide whether it is satisfiable.*

Problem (3-SAT)
Given a propositional formula in CNF *with each clause of size 3, decide whether it is satisfiable.*

Satisfiability Problem

Problem (SAT)
Given a propositional formula, decide whether it is satisfiable.

Problem (CNF-SAT)
Given a propositional formula in CNF, decide whether it is satisfiable.

Problem (3-SAT)
Given a propositional formula in CNF with each clause of size 3, decide whether it is satisfiable.

Theorem
SAT, CNF-SAT, and 3-SAT are NP*-complete.*

Theorem
SAT, CNF-SAT, and 3-SAT are NP*-complete.*

Proof ideas.

- Whether an assignment is a model can be checked in polynomial time.
- A computation of Turing machine of polynomial length can be encoded by a CNF formula of polynomial size. □

There are no known polynomial algorithms for propositional satisfiability.

Hardness of Propositional Satisfiability: Practice

Modern SAT solvers can decide satisfiability of formulas with thousands of variables and millions of clauses thanks to

- clever algorithms (worst case exponential)
- clever data structures
- clever heuristics

Give it a try:

- MiniSAT (`http://minisat.se/`)
- CaDiCaL (`https://github.com/arminbiere/cadical`)
- Kissat (`https://github.com/arminbiere/kissat`)

Applications: Other Logical Problems

Other logical problems can be reduced[1] to satisfiability

Validity

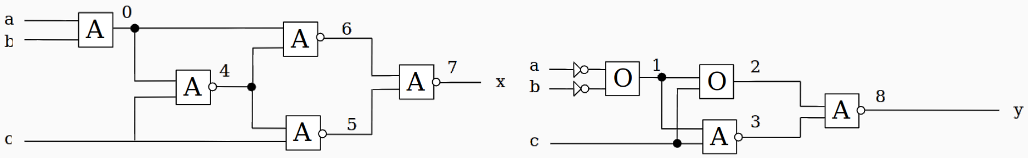
- a φ is valid if every total assignment for φ is its model
- φ is valid $\quad \Leftrightarrow \quad \neg\varphi$ is not satisfiable

Entailment

- $\varphi \models \psi \quad \Leftrightarrow \quad (\varphi \rightarrow \psi)$ is valid $\quad \Leftrightarrow \quad (\varphi \wedge \neg\psi)$ is not satisfiable

[1] in the sense of Turing reductions

Applications: Hardware Design



[Example from: https://www21.in.tum.de/~lammich/2015_SS_Seminar_SAT/resources/Equivalence_Checking_11_30_08.pdf]

Are circuits C_1 and C_2 equivalent?

Is $\neg(\mathit{formula}(C_1) \leftrightarrow \mathit{formula}(C_2))$ UNSAT? (called a miter formula)

Works only for reasonably small circuits. For larger circuits (millions of gates), more involved techniques are necessary, e.g., SAT-sweeping.

Applications: Package Dependency

- package P has n versions:
  $x_1^P, x_2^P, \ldots, x_n^P$
- only one can be installed at a time:
  $\neg x_i^P \vee \neg x_j^P$ for all packages P and versions $i \neq j$
- packages have dependencies:
  $x_3^P \to (x_1^Q \vee x_2^Q) \wedge x_8^R$
- I have version 1 of package Q and want to install version 3 of package P:
  $x_3^P \wedge x_8^Q$
- what dependencies I need to install:
  Is the formula SAT? What is its model?

Used for example by package manager Cabal for Haskell.

Applications: Open Problems in Mathematics

Definition
A triple $(a, b, c) \in \mathbb{N}$ is called Pythagorean if $a^2 + b^2 = c^2$.

Question
Can every set of numbers $N = \{1, 2, \ldots, n\}$ be colored by two colors such that there is no monochromatic Pythagorean triple?

[2] `https://www.cs.utexas.edu/~marijn/ptn/`

Definition
A triple $(a, b, c) \in \mathbb{N}$ is called Pythagorean if $a^2 + b^2 = c^2$.

Question
Can every set of numbers $N = \{1, 2, \ldots, n\}$ be colored by two colors such that there is no monochromatic Pythagorean triple?

The answer is no ($n = 7825$) and was found by a SAT solver in 2016[2]. Previous lower bound was that $n = 7664$ can be colored.

[2] `https://www.cs.utexas.edu/~marijn/ptn/`

Applications: Open Problems in Mathematics

1. Define a formula F_i whose models are two-colorings of $\{1, 2, \ldots, i\}$ with no monochromatic Pythagorean triples.

$$F_i = \bigwedge_{(a,b,c) \text{ is a Pythagorean triple}} (x_a \vee x_b \vee x_c) \wedge (\neg x_a \vee \neg x_b \vee \neg x_c)$$

2. F_{7824}: 6492 variables and 18930 clauses; F_{7825}: 6494 variables and 18944 clauses.
3. Preprocessing: reduce this to 3740 variables and 14652 clauses; and 3745 variables and 14672 clauses.
4. Use parallel SAT solver and tweak some of its heuristics.
5. Use a parallel machine with 800 cores for 2 days.
6. Find that F_{7824} is satisfiable and F_{7825} is unsatisfiable.
7. Get a largest unsatisfiability proof ever (200 terabytes).

Thinking with Clauses

Clauses = Implications

Important view during this course: clauses = implications.

$\{A, B\}$ (i.e., $A \vee B$)

- $\neg A \rightarrow B$
- $\neg B \rightarrow A$

$\{A, B, C\}$ (i.e., $A \vee B \vee C$)

- $(\neg A \wedge \neg B) \rightarrow C$
- $(\neg A \wedge \neg C) \rightarrow B$
- $\ldots$

2-CNF = formula in CNF with clauses of size 2

2-SAT = decide satisfiability of formula in 2-CNF

Example

Is the following 2-CNF formula satisfiable?

$$\{A, B\}, \quad \{\neg B, C\}, \quad \{\neg C, A\}$$
$$\{\neg A, \neg B\}, \quad \{B, \neg A\}, \quad \{C, \neg D\}$$

2-CNF = formula in CNF with clauses of size 2

2-SAT = decide satisfiability of formula in 2-CNF

Example

Is the following 2-CNF formula satisfiable?

$$\{A, B\}, \qquad \{\neg B, C\}, \qquad \{\neg C, A\}$$
$$\{\neg A, \neg B\}, \qquad \{B, \neg A\}, \qquad \{C, \neg D\}$$

Theorem

2-SAT can be solved in linear time.

Theorem
2-SAT can be solved in linear time.

Theorem
2-SAT can be solved in linear time.

Proof.
Let φ be in 2-CNF. Construct a graph $G = (V, E)$ with

- $V = \{v \mid v \in Atoms(\varphi)\} \cup \{\neg v \mid v \in Atoms(\varphi)\}$
- $E = \{(\neg a, b) \mid \{a, b\} \in \varphi\}$

φ is satisfiable $\Leftrightarrow$ G has no cycle that contains both v and $\neg v$ for some v $\square$

Encoding Graph Coloring

Given an undirected graph $G = (V, E)$, can it be colored by three colors (red, green, blue) so that no edge has endpoints of the same color?

Encoding

- variables v_r, v_g, v_b for each $v \in V$

Encoding Graph Coloring

Given an undirected graph $G = (V, E)$, can it be colored by three colors (red, green, blue) so that no edge has endpoints of the same color?

Encoding

- variables v_r, v_g, v_b for each $v \in V$
- at least one color constraint: $\{v_r, v_g, v_b\}$ for each $v \in V$

Encoding Graph Coloring

Given an undirected graph $G = (V, E)$, can it be colored by three colors (red, green, blue) so that no edge has endpoints of the same color?

Encoding

- variables v_r, v_g, v_b for each $v \in V$
- at least one color constraint: $\{v_r, v_g, v_b\}$ for each $v \in V$
- at most one color constraints $\{\neg v_r, \neg v_g\}$, $\{\neg v_g, \neg v_b\}$, $\{\neg v_r, \neg v_b\}$ for each $v \in V$

Encoding Graph Coloring

Given an undirected graph $G = (V, E)$, can it be colored by three colors (red, green, blue) so that no edge has endpoints of the same color?

Encoding

- variables v_r, v_g, v_b for each $v \in V$
- at least one color constraint: $\{v_r, v_g, v_b\}$ for each $v \in V$
- at most one color constraints $\{\neg v_r, \neg v_g\}$, $\{\neg v_g, \neg v_b\}$, $\{\neg v_r, \neg v_b\}$ for each $v \in V$
- coloring constraint $u_c \to \neg v_c$ for each edge $\{u, v\} \in E$ and each color $c \in \{r, g, b\} \equiv$ clause $\{\neg u_c, \neg v_c\}$

Given an undirected graph $G = (V, E)$, can it be colored by three colors (red, green, blue) so that no edge has endpoints of the same color?

Encoding

- variables v_r, v_g, v_b for each $v \in V$
- at least one color constraint: $\{v_r, v_g, v_b\}$ for each $v \in V$
- at most one color constraints $\{\neg v_r, \neg v_g\}$, $\{\neg v_g, \neg v_b\}$, $\{\neg v_r, \neg v_b\}$ for each $v \in V$
- coloring constraint $u_c \rightarrow \neg v_c$ for each edge $\{u, v\} \in E$ and each color $c \in \{r, g, b\} \equiv$ clause $\{\neg u_c, \neg v_c\}$
- models of the formula $\simeq$ valid colorings

Conversion to CNF: Tseitin encoding

Conversion to equivalent CNF can be exponential, but do we really need equivalence?

Definition
The formulas φ and ψ are equisatisfiable if both are satisfiable or both unsatisfiable.

Theorem
For each formula φ there exists an equisatisfiable formula φ^{CNF} with $\mathcal{O}(|\varphi|)$ clauses of size at most three.

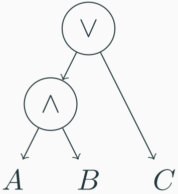
Proof.
Tseitin encoding. □

Conversion to CNF: Tseitin encoding by example

$\varphi = (A \wedge B) \vee C$

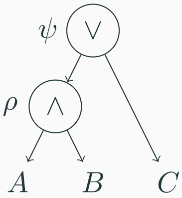
Conversion to CNF: Tseitin encoding by example

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



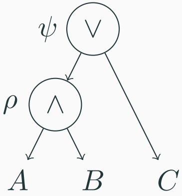
Conversion to CNF: Tseitin encoding by example

$\varphi = (A \wedge B) \vee C$



Conversion to CNF: Tseitin encoding by example

$\varphi = (A \wedge B) \vee C$



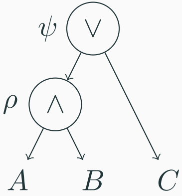
$(A_\rho \leftrightarrow (A \wedge B)) \wedge$

$(A_\psi \leftrightarrow (A_\rho \vee C)) \wedge$

A_ψ

Conversion to CNF: Tseitin encoding by example

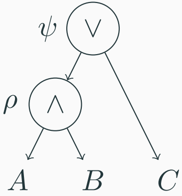
$\varphi = (A \wedge B) \vee C$



$$
\begin{aligned}
&(A_\rho \leftrightarrow (A \wedge B)) \wedge & & (A_\rho \rightarrow (A \wedge B)) \wedge \\
& & & (A_\rho \leftarrow (A \wedge B)) \wedge \\
&(A_\psi \leftrightarrow (A_\rho \vee C)) \wedge & \equiv \quad & (A_\psi \rightarrow (A_\rho \vee C)) \wedge \\
& & & (A_\psi \leftarrow (A_\rho \vee C)) \wedge \\
&A_\psi & & A_\psi
\end{aligned}
$$

Conversion to CNF: Tseitin encoding by example

$\varphi = (A \wedge B) \vee C$



$$
\begin{array}{lclcl}
(A_\rho \leftrightarrow (A \wedge B)) \wedge & & (A_\rho \rightarrow (A \wedge B)) \wedge & & \{\neg A_\rho, A\}, \{\neg A_\rho, B\}, \\
& & (A_\rho \leftarrow (A \wedge B)) \wedge & & \{\neg A, \neg B, A_\rho\}, \\
(A_\psi \leftrightarrow (A_\rho \vee C)) \wedge & \equiv & (A_\psi \rightarrow (A_\rho \vee C)) \wedge & \equiv & \{\neg A_\psi, A_\rho, C\}, \\
& & (A_\psi \leftarrow (A_\rho \vee C)) \wedge & & \{\neg A_\rho, A_\psi\}, \{\neg C, A_\psi\}, \\
A_\psi & & A_\psi & & \{A_\psi\}
\end{array}
$$

Conversion to CNF: Tseitin encoding

1. Create a new Tseitin variable A_ψ for each subformula of φ.
2. Add unit clause $\{A_\varphi\}$.
3. Define semantics of the new Tseitin variables A_ψ:

ψ	definition of A_ψ	added clauses
$\rho_1 \lor \rho_2$	$A_\psi \rightarrow (A_{\rho_1} \lor A_{\rho_2})$ $A_\psi \leftarrow (A_{\rho_1} \lor A_{\rho_2})$	$\{\neg A_\psi, A_{\rho_1}, A_{\rho_2}\}$ $\{\neg A_{\rho_1}, A_\psi\}, \{\neg A_{\rho_2}, A_\psi\}$
$\rho_1 \land \rho_2$	$A_\psi \rightarrow (A_{\rho_1} \land A_{\rho_2})$ $A_\psi \leftarrow (A_{\rho_1} \land A_{\rho_2})$	$\{\neg A_\psi, A_{\rho_1}\}, \{\neg A_\psi, A_{\rho_2}\}$ $\{\neg A_{\rho_1}, \neg A_{\rho_2}, A_\psi\}$
$\neg\rho$	$A_\psi \rightarrow \neg A_\rho$ $A_\psi \leftarrow \neg A_\rho$	$\{\neg A_\psi, \neg A_\rho\}$ $\{A_\rho, A_\psi\}$
$\rho_1 \leftrightarrow \rho_2$	$A_\psi \rightarrow (A_{\rho_1} \leftrightarrow A_{\rho_2})$ $A_\psi \leftarrow (A_{\rho_1} \leftrightarrow A_{\rho_2})$	$\{\neg A_\psi, \neg A_{\rho_1}, A_{\rho_2}\}, \{\neg A_\psi, A_{\rho_1}, \neg A_{\rho_2}\}$ $\{\neg A_{\rho_1}, \neg A_{\rho_2}, A_\psi\}, \{A_{\rho_1}, A_{\rho_2}, A_\psi\}$

Tseitin encoding

- often used in practice
- also works for DAG representation of formulas: one Tseitin variable for each node in the DAG
- transforming to increase shared subexpression helps $(B \wedge A) \rightsquigarrow (A \wedge B)$
- additional preprocessing helps: $(A \vee (B \vee C)) \rightsquigarrow (A \vee B \vee C)$ and then encode $A_\varphi \leftrightarrow (A \vee B \vee C)$ as one Tseitin variable and four implications
- some of the clauses are not needed (Plaisted-Greenbaum)

Next time

Classical SAT algorithms

- propositional resolution
- Davis-Putnam-Logemann-Loveland algorithm (DPLL)