universität innsbruck

SAT and SMT Solving

Sarah Winkler

KRDB
Department of Computer Science
Free University of Bozen-Bolzano

lecture 1
WS 2022

Outline

- Introduction
  - Organisation
  - Why SAT and SMT?
  - Course Topics
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

Important Information

- LVA 703147 (VU3)
- http://cl-informatik.uibk.ac.at/teaching/ws22/satsmt/

Time and Place

VU	Friday	14:15 – 17:00	SR12
PS	Friday	16:00 – 16:45	SR12

Grading

- 65% weekly exercises
- 35% tests on 2 December 2022 and 3 February 2023 (one hour each)
- attendence required

Exercises

- 10 points per week
- indicate solved exercises before Friday 10:00 in OLAT, submit solutions

Questions, Comments, Suggestions

- sarah.winkler@uibk.ac.at
- OLAT

Outline

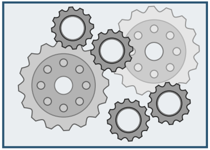
- Introduction
  - Organisation
  - Why SAT and SMT?
  - Course Topics
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

SAT Solving

input:	propositional formula φ	
output:	SAT + valuation v such that $v(\varphi) = T$	if φ satisfiable
	UNSAT	otherwise

φ $\longrightarrow$

$(q \vee \neg r) \wedge (\neg q \vee r) \wedge p$



SAT solver

$\nearrow$ SAT (v) $\quad v(p) = \mathrm{T}$, $v(q) = \mathrm{F}$, $v(r) = \mathrm{F}$

$\searrow$ UNSAT

Terminology

- decision problem P is problem with answer yes or no
- SAT encoding of decision problem P is propositional formula φ_P such that

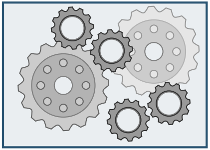
$$\text{answer to } P \text{ is yes} \quad \Longleftrightarrow \quad \varphi_P \text{ is satisfiable}$$

SMT Solving

input:	formula φ involving theory T	
output:	SAT + valuation v such that $v(\varphi) = T$	if φ is T-satisfiable
	UNSAT	otherwise

φ

$a + b \geqslant c \vee (a = 0 \wedge p)$



SMT solver

SAT (v) $\quad v(a) = 3 \quad v(b) = 0 \quad v(c) = 0 \quad v(p) = \mathrm{T}$

UNSAT

Example (Theories)

- arithmetic $\quad 2a + b \geqslant c \vee (a = 0 \wedge p)$
- uninterpreted functions $\quad \mathrm{f}(x, y) \neq \mathrm{f}(y, x) \wedge \mathrm{g}(\mathrm{f}(x, x)) = \mathrm{g}(y)$
- bit vectors $\quad ((\mathrm{zext}_{32}\ a_8) + b_{32}) \times c_{32} >_u 0_{32}$

Terminology

- SMT encoding over theory T of decision problem P is formula φ_P such that

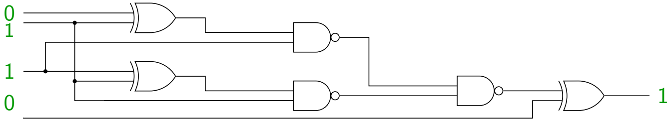
$$\text{answer to } P \text{ is yes} \iff \varphi_P \text{ is satisfiable}$$

Application 1: Hardware Verification

Problem

- errors in hardware chips are costly (Intel paid $475 million for FDIV bug)
- testing is not enough to guarantee desired behavior

Example (Formal Circuit Model)



SAT Encoding

- variables for input and output
- SAT formulas for implemented behavior and expected behavior (specification)
- check for equivalence

Impact

- ensured correctness, more reliable hardware components (formal verification)
- manufacturers rely on SAT-based verification since beginning of 2000s
  e.g., Intel Core i7 implements over 2700 distinct verified micro-instructions

Application 2: Driving License Test

Problem

Austrian driving license test consists of 80 questions out of 1500 such that the following conditions are satisfied:

- 30 "main questions" with 3 sub-questions each
- at least 12 main questions must be about crossroads
- at least 12 main questions must have pictures
- at least 5 "hard", "medium", and "easy" main questions

- how can software find valid question set?

SAT encoding

- variables q_i for $1 \leqslant i \leqslant 1500$
- idea: valuation v sets $v(q_i) = \mathrm{T}$ if question i is included, $v(q_i) = \mathrm{F}$ otherwise

- $\sum_{i \in Q_{\text{xroads}}} q_i \geqslant 12$
- $\sum_{i \in Q_{\text{pictures}}} q_i \geqslant 12$
- $\sum_{i \in Q_{\text{hard}}} q_i \geqslant 5$
- ...

Result

easy generation of valid question sets (with some random preselection)

Application 3: Pythagorean Triples

Problem
Can one color all natural numbers with two colors such that whenever $x^2 + y^2 = z^2$ not all of x, y, and z have same color?

Example

$$3^2 + 4^2 = 5^2 \qquad 5^2 + 12^2 = 13^2$$

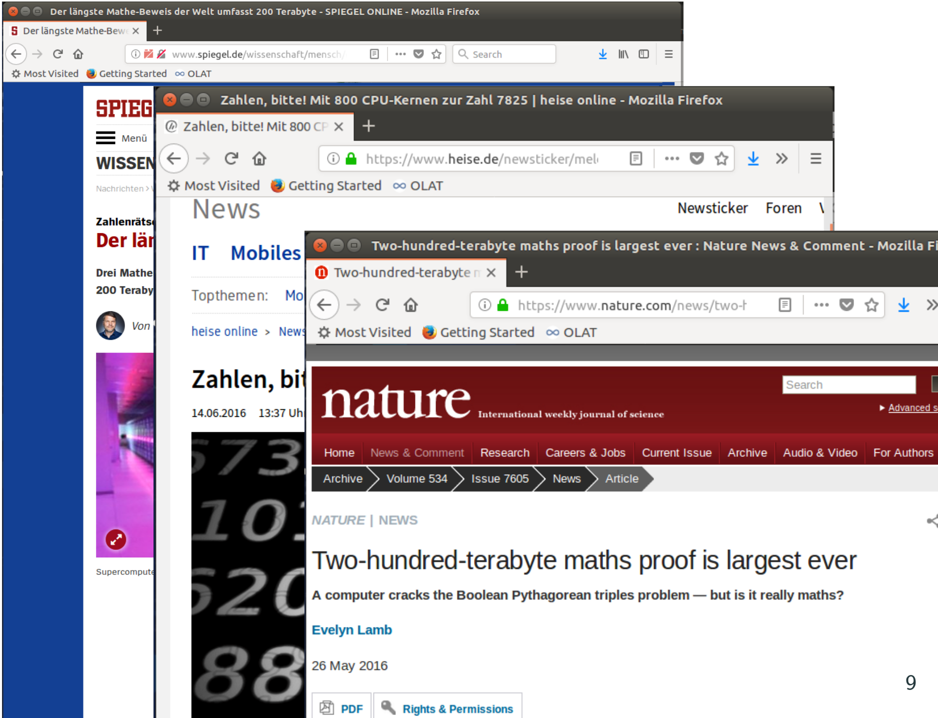
(a)	1	2	3	4	5	6	7	8	9	10	11	12	13	...	✓
(b)	1	2	3	4	5	6	7	8	9	10	11	12	13	...	✗

SAT encoding

- variables x_i for $1 \leqslant i \leqslant n$ such that x_i becomes true iff it is colored red
- SAT encoding: for all $a^2 + b^2 = c^2$ include $(x_a \vee x_b \vee x_c) \wedge (\bar{x}_a \vee \bar{x}_b \vee \bar{x}_c)$ (+ symmetry breaking, simplification, heuristics)

Result: No. Coloring exists only up to 7,825.

1000s of variables, solving time 2 days with 800 processors, 200 TB of proof


Der längste Mathe-Beweis der Welt umfasst 200 Terabyte - SPIEGEL ONLINE - Mozilla Firefox
www.spiegel.de/wissenschaft/mensch/
Most Visited
Getting Started
OLAT
Menü
Nachrichten
Zahlenräts
Der lä
Drei Mathe
200 Teraby
Von
Supercompute
Zahlen, bitte! Mit 800 CPU-Kernen zur Zahl 7825 | heise online - Mozilla Firefox
https://www.heise.de/newsticker/mel
News
Newsticker
Foren
IT
Mobiles
Topthemen:
heise online > News
Zahlen, bi
14.06.2016 13:37 Uh
Two-hundred-terabyte maths proof is largest ever : Nature News & Comment - Mozilla Fi
https://www.nature.com/news/two-t
nature
International weekly journal of science
Search
Home
News & Comment
Research
Careers & Jobs
Current Issue
Archive
Audio & Video
For Authors
Archive
Volume 534
Issue 7605
News
Article
NATURE | NEWS
Two-hundred-terabyte maths proof is largest ever
A computer cracks the Boolean Pythagorean triples problem — but is it really maths?
Evelyn Lamb
26 May 2016
PDF
Rights & Permissions

Application 4: Tournament Scheduling

Problem: Round Robin scheduling

Schedule sports league tournament for n teams, p periods of $n-1$ rounds each (+ venue restrictions, break restrictions, ...)

Example (Österreichische Fußball-Bundesliga)

10 teams play in 4 periods (9 rounds each),

periods 1 & 2 and 3 & 4 mirrored

(Part of) SAT encoding

- variable x_{ijpr} is true if team i plays team j at home in period p, round r
- $$\bigwedge_{i,p,r} \bigvee_{j \neq i} (x_{ijpr} \vee x_{jipr})$$ each team plays in every round

  $$\bigwedge_{i,p,r} \bigwedge_{j \neq i} \bigwedge_{k \neq i \wedge k \neq j} (x_{ijpr} \rightarrow \neg(x_{ikpr} \vee x_{kipr}))$$ each team plays at most once in every round

  $$\bigwedge_{i,j,r} (x_{ij1r} \rightarrow x_{ji2r}) \wedge (x_{ij3r} \rightarrow x_{ji4r})$$ mirror rounds 1& 2 and 3& 4

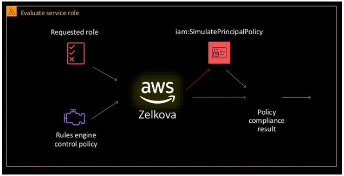
Result

SAT scheduling is 100x faster than previous industrial scheduling tools

Application 5: Policy Verification in AWS Zelkova

Problem

- in Amazon web services, users define complex access policies for services
- users want to check whether
  - policy allows agent X to do action Y
  - policy A is more/less restrictive, or equivalent to, policy B
- security critical
- should be checked automatically



SMT encoding

- using string and bitvector variables, and reasoning emulating regular expressions
- policy is encoded as $\bigvee_{S \in Allow} [S] \wedge \neg \bigvee_{S \in Deny} [S]$
- where for each statement S,
  $[S] := (\bigvee_{v \in P(S)} p{=}v) \wedge (\bigvee_{v \in A(S)} a{=}v) \wedge (\bigvee_{v \in R(S)} r{=}v) \wedge (\bigvee_{O \in C(S)} [O])$

Result

- Zelkova is invoked tens of millions of times per day
- latency in magnitude of milliseconds

Application 6: Network Verification in Microsoft Azure

Problem

- Microsoft Azure data centers must ensure cloud contracts:
  network access restrictions, forwarding tables, Border Gateway Protocol policies
- routing configuration should satisfy contracts – but routing tables change fast!

- cloud contracts should be verified automatically

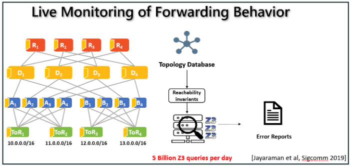
SMT encoding in SecGuru tool

- model network configuration as formula

$$
\begin{array}{l}
\mathit{Router} \equiv \\
\quad \textbf{if } \ldots \\
\quad \textbf{if } \mathit{dst} = 10.91.114.128/25 \textbf{ then } n_3 \vee n_6 \vee n_7 \textbf{ else} \\
\quad \textbf{if } \mathit{dst} = 10.91.114.0/25 \textbf{ then } n_3 \vee n_4 \vee n_5 \vee n_6 \textbf{ else} \\
\quad n_1 \vee n_2
\end{array}
$$

- express cloud contracts as formulas

$$\neg \mathit{Cluster}(\mathit{dst}) \wedge \mathit{Router}(\mathit{dst}) \Rightarrow \bigvee_n \mathit{RouterAbove}(n)$$



- assert that configuration does not satisfy contract: if satisfiable, found bug!

Result

- Azure uses billions of SMT queries for network verification every day

Hall of Fame

Herbrand Award 2019

Nikolaj Bjørner and **Leonardo de Moura**
"for their contributions to SMT solving, including its theory, implementation, and application to a wide range of academic and industrial needs"

CAV Award 2021

For pioneering contributions to the foundations of the theory and practice of satisfiability modulo theories (SMT).

Outline

- Introduction
  - Organisation
  - Why SAT and SMT?
  - Course Topics
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

Contents

Part 1: SAT

DPLL, conflict analysis, CDCL, 2-watched literals, heuristics, unsatisfiable cores, maxSAT, symmetry breaking

Part 2: SMT

DPLL(T), eager vs lazy, T-propagation, Nelson-Oppen combination, maxSMT

Part 3: Theory Solving

- equality with uninterpreted functions (congruence closure, conflict analysis)
- linear real arithmetic (simplex algorithm)
- arrays (reduction to EUF, lemmas on demand)
- bit vectors (bit blasting, preprocessing)

Practice

SAT solvers, SMT solvers, encoding, DIMACS, SMT-LIB, model checking

Outline

- Introduction
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

Propositional Logic Revisited

Concepts

- literal
- formula
- assignment
- satisfiability and validity
- negation normal form (NNF)
- conjunctive normal form (CNF)
- disjunctive normal form (DNF)

Definition (Propositional Logic: Syntax)

propositional formulas are built form

- atoms $p,\ q,\ r,\ p_1,\ p_2,\ \ldots$
- constants $\bot,\ \top$
- negation $\neg p$ "not p"
- conjunction $p \wedge q$ "p and q"
- disjunction $p \vee q$ "p or q"
- implication $p \rightarrow q$ "if p then q holds"
- equivalence $p \leftrightarrow q$ "p if and only if q"

according to the BNF grammar

$$\varphi ::= p \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

Conventions

- binding precedence $\neg \quad > \quad \wedge \quad > \quad \vee \quad > \quad \rightarrow, \leftrightarrow$
- omit outer parantheses
- $\rightarrow$, $\wedge$, $\vee$ are right-associative: $p \rightarrow q \rightarrow r$ denotes $p \rightarrow (q \rightarrow r)$

Definition (Propositional Logic: Semantics)

- valuation (truth assignment) is mapping $v : \{p, q, r, \dots\} \to \{\mathsf{F}, \mathsf{T}\}$ from atoms to truth values
- extension to formulas:

$$v(\bot) = \mathsf{F} \qquad\qquad v(\top) = \mathsf{T}$$

$$v(\varphi \wedge \psi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = v(\psi) = \mathsf{T} \\ \mathsf{F} & \text{otherwise} \end{cases} \qquad v(\neg\varphi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = \mathsf{F} \\ \mathsf{F} & \text{if } v(\varphi) = \mathsf{T} \end{cases}$$

$$v(\varphi \vee \psi) = \begin{cases} \mathsf{F} & \text{if } v(\varphi) = v(\psi) = \mathsf{F} \\ \mathsf{T} & \text{otherwise} \end{cases} \qquad v(\varphi \leftrightarrow \psi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = v(\psi) \\ \mathsf{F} & \text{otherwise} \end{cases}$$

$$v(\varphi \to \psi) = \begin{cases} \mathsf{F} & \text{if } v(\varphi) = \mathsf{T},\ v(\psi) = \mathsf{F} \\ \mathsf{T} & \text{otherwise} \end{cases}$$

Definitions

- formula φ is satisfiable if $v(\varphi) = \mathrm{T}$ for some valuation v
- formula φ is valid if $v(\varphi) = \mathrm{T}$ for every valuation v
- semantic entailment $\varphi_1, \dots, \varphi_n \vDash \psi$
  if $v(\psi) = \mathrm{T}$ whenever $v(\varphi_1) = v(\varphi_2) = \cdots = v(\varphi_n) = \mathrm{T}$
- formulas φ and ψ are equivalent ($\varphi \equiv \psi$) if $v(\varphi) = v(\psi)$ for every valuation v
- formulas φ and ψ are equisatisfiable ($\varphi \approx \psi$) if

$$\varphi \text{ is satisfiable} \iff \psi \text{ is satisfiable}$$

Theorem

formula φ is unsatisfiable if and only if $\neg\varphi$ is valid

Theorem

satisfiability and validity are decidable

Proof.
Check all assignments (for n variables, 2^n possibilities). ∎

Definition (Literal)

- literal is atom p or negation of atom $\neg p$
- literals l_1 and l_2 are complementary if $l_1 = \neg l_2$ or $l_2 = \neg l_1$
- write l^c for complementary literal of l

Definitions

- negation normal form (NNF) if formula with negation only applied to atoms
- conjunctive normal form (CNF) is conjunction of disjunctions
- 3-CNF is conjunction of disjunctions with 3 literals: $\bigwedge_i (a_i \vee b_i \vee c_i)$
- disjunctive normal form (DNF) is disjunction of conjunctions

Theorem

for every formula φ there is CNF ψ, 3-CNF χ and DNF η such that $\varphi \equiv \psi \equiv \chi \equiv \eta$

Remarks

- translation from formula to CNF can result in exponential blowup
- Tseitin's transformation is linear and produces equisatisfiable formula

Satisfiability (SAT)

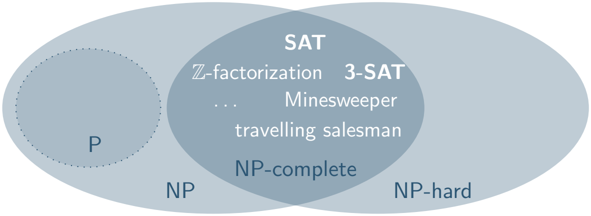
instance: propositional formula φ
question: is φ satisfiable?

3-Satisfiability (3-SAT)

instance: propositional formula φ in 3-CNF
question: is φ satisfiable?

Theorem

SAT and 3-SAT are NP-complete problems



- 1 million \$ prize money awarded for solution to $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Outline

- Introduction
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

Approach

- most state-of-the-art SAT solvers use variation of Davis - Putnam - Logemann - Loveland (DPLL) procedure (1962)
- DPLL is sound and complete backtracking-based search algorithm
- can be described abstractly by transition system (Nieuwenhuis, Oliveras, Tinelli 2006)

Definition (Abstract DPLL)

- decision literal is annotated literal l^d
- state is pair $M \parallel F$ for
  - list M of (decision) literals — partial assignment
  - formula F in CNF
- transition rules

$$M \parallel F \quad \Longrightarrow \quad M' \parallel F' \quad \text{or} \quad \text{FailState}$$

Definition (DPLL Transition Rules)

- unit propagation $\quad M \parallel F,\ C \vee l \quad \Longrightarrow \quad M\ l \parallel F,\ C \vee l$
  if $M \models \neg C$ and l is undefined in M
- pure literal $\quad M \parallel F \quad \Longrightarrow \quad M\ l \parallel F$
  if l occurs in F but l^c does not occur in F, and l is undefined in M
- decide $\quad M \parallel F \quad \Longrightarrow \quad M\ l^d \parallel F$
  if l or l^c occurs in F, and l is undefined in M
- backtrack $\quad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l^c \parallel F, C$
  if $M\ l^d\ N \models \neg C$ and N contains no decision literals
- fail $\quad M \parallel F, C \quad \Longrightarrow \quad$ FailState
  if $M \models \neg C$ and M contains no decision literals
- backjump $\quad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$
  if $M\ l^d\ N \models \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \models C' \vee l'$ backjump clause
  - $M \models \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$\varphi = (\overline{1} \vee \overline{2}) \wedge (2 \vee 3) \wedge (\overline{1} \vee \overline{3} \vee 4) \wedge (2 \vee \overline{3} \vee \overline{4}) \wedge (1 \vee 4)$

$$
\begin{array}{lrl r}
 & & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \\
\Longrightarrow & 1^d & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{decide} \\
\Longrightarrow & 1^d\ \overline{2} & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & 1^d\ \overline{2}\ 3 & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & 1^d\ \overline{2}\ 3\ 4 & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \boxed{2 \vee \overline{3} \vee \overline{4}},\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & \overline{1} & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{backtrack} \\
\Longrightarrow & \overline{1}\ 4 & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & \overline{1}\ 4\ 3^d & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{decide} \\
\Longrightarrow & \overline{1}\ 4\ 3^d\ 2 & \| \; \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ 2 \vee \overline{3} \vee \overline{4},\ 1 \vee 4 & \text{unit propagate}
\end{array}
$$

Example (Backjump)

$\varphi = (\overline{1} \vee 2) \wedge (\overline{1} \vee \overline{3} \vee 4 \vee 5) \wedge (\overline{2} \vee \overline{4} \vee \overline{5}) \wedge (4 \vee \overline{5}) \wedge (\overline{4} \vee 5)$

$$
\begin{array}{lrll}
 & \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \\
\Longrightarrow & 1^d \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{decide} \\
\Longrightarrow & 1^d\ 2 \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{unit propagate} \\
\Longrightarrow & 1^d\ 2\ 3^d \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{decide} \\
\Longrightarrow & 1^d\ 2\ 3^d\ 4^d \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{decide} \\
\Longrightarrow & 1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \boxed{\overline{4} \vee 5} & \text{unit propagate} \\
\Longrightarrow & 1^d\ 2\ 3^d\ \overline{4} \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{backtrack} \\
\Longrightarrow & 1^d\ 2\ 3^d\ \overline{4}\ 5 \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ \boxed{4 \vee \overline{5}},\ \overline{4} \vee 5 & \text{unit propagate} \\
\Longrightarrow & \ldots & & \text{backtrack} \\
\Longrightarrow & 1^d\ 2\ \overline{3} \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 & \text{backjump} \\
\Longrightarrow^+ & 1^d\ 2\ \overline{3}\ \overline{4}\ \overline{5} \parallel & \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5 &
\end{array}
$$

Decisions 1 and 3 are incompatible: $\varphi \models \overline{1} \vee \overline{3}$

Definition

basic DPLL $\mathcal{B}$ consists of unit propagation, decide, fail, and backjump

Properties

if $\parallel F \Longrightarrow_{\mathcal{B}}^{*} M \parallel F'$ then

- $F = F'$
- M does not contain complementary literals
- literals in M are distinct
- length of M is bounded by number of atoms

Lemma (Model Entailment)

Suppose $\parallel F \Longrightarrow_{\mathcal{B}}^{*} M \parallel F$ *such that*

- $M = M_0\ l_1^d\ M_1\ l_2^d\ M_2\ \ldots\ l_k^d\ M_k$ *and*
- *there are no decision literals in* M_i.

decisions imply
all other literals in M

Then $F, l_1, \ldots, l_i \vDash M_i$ *for all* $0 \leqslant i \leqslant k$.

Theorem (Termination)
for any formula F there are no infinite derivations

$$\parallel F \quad \Longrightarrow_{\mathcal{B}} \quad S_1 \quad \Longrightarrow_{\mathcal{B}} \quad S_2 \quad \Longrightarrow_{\mathcal{B}} \quad \dots$$

missing literals in M

Proof.

- for list of distinct literals M, define $a(M) = n - |M|$ where
  - n is number of propositional variables
  - $|M|$ is length of M
- measure state $\quad M_0\ l_1^d\ M_1\ l_2^d\ M_2 \dots\ l_k^d\ M_k \parallel F \quad$ by tuple

$$(a(M_0),\ a(M_1),\ \dots,\ a(M_k),\ \underbrace{\infty, \dots, \infty}_{n-k})$$

- compare tuples lexicographically by extension of $>_{\mathbb{N}}$ with ∞ maximal
- every transition step decreases measure

Example (Revisited for termination)

$\varphi = (\overline{1} \vee \overline{2}) \wedge (2 \vee 3) \wedge (\overline{1} \vee \overline{3} \vee 4) \wedge (2 \vee \overline{3} \vee \overline{4}) \wedge (1 \vee 4)$

$$
\begin{array}{lrll}
 & \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & & (n, \infty, \dots) \\
\Longrightarrow & 1^d \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{decide} & (n, n, \infty, \dots) \\
\Longrightarrow & 1^d\ \overline{2} \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{unit propagate} & (n, n-1, \infty, \dots) \\
\Longrightarrow & 1^d\ \overline{2}\ 3 \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{unit propagate} & (n, n-2, \infty, \dots) \\
\Longrightarrow & 1^d\ \overline{2}\ 3\ 4 \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{unit propagate} & (n, n-3, \infty, \dots) \\
\Longrightarrow & \overline{1} \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{backtrack} & (n-1, \infty, \dots) \\
\Longrightarrow & \overline{1}\ 4 \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{unit propagate} & (n-2, \infty, \dots) \\
\Longrightarrow & \overline{1}\ 4\ 3^d \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{decide} & (n-2, n, \infty, \dots) \\
\Longrightarrow & \overline{1}\ 4\ 3^d\ 2 \parallel \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{1} \vee \overline{3} \vee 4,\ \dots & \text{unit propagate} & (n-2, n-1, \infty, \dots)
\end{array}
$$

Observations used in proof

- decide replaces ∞ by n
- unit propagate, backtrack, and backjump replace m by $m-1$

Consider maximal derivation with final state S_n:

$$\| F \quad \Longrightarrow_{\mathcal{B}} \quad S_1 \quad \Longrightarrow_{\mathcal{B}} \quad S_2 \quad \Longrightarrow_{\mathcal{B}} \quad \dots \quad \Longrightarrow_{\mathcal{B}} \quad S_n$$

Theorem

if $S_n = \mathrm{FailState}$ then F is unsatisfiable

Proof.

- must have $\| F \Longrightarrow_{\mathcal{B}}^{*} M \| F \Longrightarrow_{\mathrm{fail}} \mathrm{FailState}$
  such that M contains no decision literals and $M \models \neg C$ for some C in F
- by Model Entailment Lemma $F \models M$, so $F \models \neg C$
- also have $F \models C$ because C is in F, so F is unsatisfiable ■

Theorem

if $S_n = M \| F'$ then F is satisfiable and $M \models F$

Proof.

- have $F = F'$
- S_n is final, so all literals of F are defined in M (otherwise decide applicable)
- $\nexists$ clause C in F such that $M \models \neg C$ (otherwise backjump or fail applicable)
- so M satisfies F ($M \models F$) ■

Outline

- Introduction
- Propositional Logic
- DPLL
- Transformations to CNF
- Using SAT Solvers

Fact
most SAT solvers require input to be in CNF

Remarks

- transforming formula to equivalent CNF can cause exponential blowup
- transforming formula into equisatisfiable CNF is possible in linear time

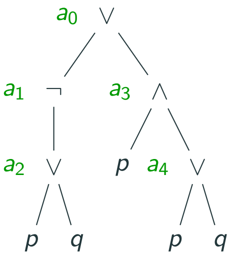
Definition
formulas φ and ψ are equisatisfiable ($\varphi \approx \psi$) if

$$\varphi \text{ is satisfiable} \quad \Longleftrightarrow \quad \psi \text{ is satisfiable}$$

Example

$$p \vee q \approx \top \qquad p \wedge \neg p \approx q \wedge \neg q \qquad p \wedge \neg p \not\approx p \wedge \neg q$$

Example (Tseitin's Transformation)



- $\varphi = \neg(p \vee q) \vee (p \wedge (p \vee q))$
- use fresh propositional variable for every connective

  $a_0 : \neg(p \vee q) \vee (p \wedge (p \vee q))$ $\quad$ $a_1 : \neg(p \vee q)$

  $a_2 : p \vee q$ $\quad$ $a_3 : p \wedge (p \vee q)$

  $a_4 : p \vee q$

- $\varphi \approx \quad a_0 \wedge (a_0 \leftrightarrow a_1 \vee a_3) \wedge (a_1 \leftrightarrow \neg a_2) \wedge (a_2 \leftrightarrow p \vee q) \wedge$
  $(a_3 \leftrightarrow p \wedge a_4) \wedge (a_4 \leftrightarrow p \vee q)$

- every $\leftrightarrow$ subexpression can be replaced by at most three clauses:

$$\begin{aligned} a \leftrightarrow b \wedge c &\quad\equiv\quad (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c) \\ a \leftrightarrow b \vee c &\quad\equiv\quad (\neg a \vee b \vee c) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \\ a \leftrightarrow \neg b &\quad\equiv\quad (\neg a \vee \neg b) \wedge (a \vee b) \end{aligned}$$

- common subexpressions can be shared

Observation

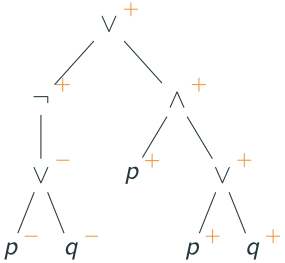
bi-implication $\leftrightarrow$ in Tseitin's transformation can be replaced by $\rightarrow$ or $\leftarrow$:
direction of implication $\rightarrow$ or $\leftarrow$ depends on polarity of subformula

Definition

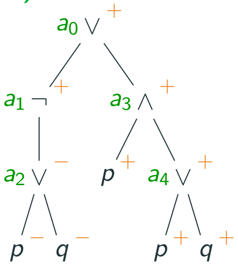
for φ subformula occurrence of ψ

- let k be number of negations above φ in syntax tree of ψ
- polarity of φ is $+$ if k is even, and $-$ otherwise

Example



Example (Plaisted and Greenbaum's Transformation)



- $\varphi = \neg(p \vee q) \vee (p \wedge (p \vee q))$
- use fresh propositional variable for every connective

  $a_0 : \neg(p \vee q) \vee (p \wedge (p \vee q))$ $\quad a_1 : \neg(p \vee q)$

  $a_2 : p \vee q$ $\quad a_3 : p \wedge (p \vee q)$

  $a_4 : p \vee q$

- add $(a_i \rightarrow \dots)$ if polarity of a_i is positive and $(a_i \leftarrow \dots)$ if negative

$$\varphi \approx \quad a_0 \wedge (a_0 \rightarrow a_1 \vee a_3) \wedge (a_1 \rightarrow \neg a_2) \wedge (a_2 \leftarrow p \vee q) \wedge (a_3 \rightarrow p \wedge a_4) \wedge (a_4 \rightarrow p \vee q)$$

- every $\leftarrow$ and $\rightarrow$ subexpression can be replaced by at most two clauses:

$$a \rightarrow b \wedge c \;\equiv\; (\neg a \vee b) \wedge (\neg a \vee c) \qquad a \leftarrow b \wedge c \;\equiv\; (a \vee \neg b \vee \neg c)$$

$$a \rightarrow b \vee c \;\equiv\; (\neg a \vee b \vee c) \qquad a \leftarrow b \vee c \;\equiv\; (a \vee \neg b) \wedge (a \vee \neg c)$$

$$a \rightarrow \neg b \;\equiv\; (\neg a \vee \neg b) \qquad a \leftarrow \neg b \;\equiv\; (a \vee b)$$

SAT Solvers

Minisat, Glucose, CaDiCaL, Glu_VC, Plingeling, MapleLRB LCM, MapleCOMPSPS, Riss, Lingeling, Treengeling, CryptoMiniSat, abcdSAT, Dimetheus, Kiel, MapleCOMSPS, Rsat, SWDiA5BY, BlackBox, SWDiA5BY, pprobSAT, glueSplit_clasp, BalancedZ, SApperloT, PeneLoPe, MXC, ROKKminisat, MiniSat_HACK_999ED, ZENN, CSHCrandMC, MiniGolf, march_rw, sattime2011, mphasesat64, sparrow2011, pmcSAT, CSHCpar8, gluebit_clasp, clasp, precosat, gNovelty, SATzilla, SatELite, Score2SAT, YalSAT, tch glucose3, ...

SAT Competition

- annual competition for different tracks (main, parallel, no-limit, ...)
- increasing set of benchmarks from industry, mathematics, cryptography, ...
- standardized input format DIMACS and proof format DRAT

`http://www.satcompetition.org/`

Minisat

- minimalistic open source solver (`http://minisat.se/` or apt, yum,...)

```
$ minisat test.sat result.txt
```

- web interface

Example (DIMACS)

formula $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (\neg x_1 \vee x_2 \vee x_4)$ can be expressed by

```
c a very simple example
p cnf 4 3
1 -3 0
2 3 -1 0
-1 2 4 0
```

The DIMACS format

- header `p cnf` n m specifies number of variables n and number of clauses m
- variables (atoms) are assumed to be $x_1, \ldots, x_n$
- literal x_i is denoted `i` and literal $\neg x_i$ is denoted `-i`
- a clause is a list of literals terminated by `0`
- lines starting with `c` are considered comments

Z3

common open source SAT/SMT solver

`https://github.com/Z3Prover/z3`

Python interface to Z3

- pip package `https://pypi.org/project/z3-solver/` (or manual installation from project site above)
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name* (calling `Bool(`*name*`)` twice yields same variable)
- `And(`$a_1, \ldots, a_n$`)` conjunction with arbitrarily many arguments
- `Or(`$a_1, \ldots, a_n$`)` disjunction with arbitrarily many arguments
- `Not(`a`)` negation
- `Implies(`a, b`)` implication
- `Xor(`a, b`)` exclusive or

Solving formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true
- `Solver.check()` check for satisfiability
- `Solver.model()` returns valuation (after successful call of `check`)

Moreover ...

- `simplify(`φ`)` simplifies formula φ
- `Solver.statistics()` is map of solving statistics

Example

```
from z3 import *

foo = Bool('foo') # create variables named 'foo', 'bar', 'qax'
bar = Bool('bar')
qax = Bool('qax')

phi = Or(foo, And(bar, Xor(foo, Not(qax)), True), False)
print(phi) # Or(foo, And(bar, Xor(foo, Not(qax)), True), False)
psi = simplify(phi)
print(psi) # Or(foo, And(bar, foo == qax))

solver = Solver()
solver.add(psi) # assert that psi should be true
solver.add(Implies(foo, qax), Or(bar, foo)) # assert something else

print(solver) # [Or(foo, And(bar, foo == qax)), Implies(foo, qax), ...]
result = solver.check() # check for satisfiability

if result:
  model = solver.model() # get valuation
  print(model[foo], model[bar], model[qax]) # False True False
```

Example (Minesweeper)

	3		1
	8		3
			2

x_1		x_2	
x_3	x_4	x_5	x_6
x_7		x_8	
x_9	x_{10}	x_{11}	

SAT Encoding

- variable x_i for each unknown cell i, $v(x_i) = \mathsf{T}$ iff cell i has mine
- constraints for every hint (number in grid)

$\boxed{1}$ $(x_2 \vee x_5 \vee x_6) \wedge ((\neg x_2 \wedge \neg x_5) \vee (\neg x_2 \wedge \neg x_6) \vee (\neg x_5 \wedge \neg x_6))$

$\boxed{8}$ $x_3 \wedge x_4 \wedge x_5 \wedge x_7 \wedge x_8 \wedge x_9 \wedge x_{10} \wedge x_{11}$

$\boxed{3}$ $((x_5 \wedge x_6 \wedge x_8) \vee (x_5 \wedge x_6 \wedge x_{11}) \vee (x_5 \wedge x_8 \wedge x_{11}) \vee (x_6 \wedge x_8 \wedge x_{11})) \wedge (\neg x_5 \vee \neg x_6 \vee \neg x_8 \vee \neg x_{11})$

$\boxed{2}$ $x_8 \wedge x_{11}$

$\boxed{3}$ $\displaystyle\bigvee_{1 \leqslant i,j \leqslant 5, i \neq j} \neg x_i \wedge \neg x_j \qquad \bigvee_{1 \leqslant i,j,k \leqslant 5, i \neq j, i \neq k, j \neq k} x_i \wedge x_j \wedge x_k$

DPLL

Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli.
Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T).
Journal of the ACM 53(6), pp. 937–977, 2006.

Application Examples

Roope Kaivola *et al*.
Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation.
Proc. 21st International Conference on Computer Aided Verification, pp. 414–429, 2009.

Andrei Horbach, Thomas Bartsch, and Dirk Briskorn.
Using a SAT solver to Schedule Sports Leagues.
Journal of Scheduling 15, pp. 117–125, 2012.

Marijn Heule, Oliver Kullmann, and Victor Marek.
Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer.
Proc. 16th International Conference on Theory and Applications of Satisfiability Testing, pp. 228–245, 2016.

John Backes *et al*:
Semantic-based Automated Reasoning for AWS Access Policies using SMT.
FMCAD 2018: 1-9