SAT/SMT Solving and Applications in Rewriting

René Thiemann [1] Sarah Winkler [2]

[1]University of Innsbruck

[2]Free University of Bolzano

Outline

Schedule

session 1	Monday	background: SAT solving, propositional logic, DPLL and CDCL
		application: search for lexicographic path orders
session 2	Tuesday	background: SMT solving, arithmetic theories, lazy approach
		application: search for Knuth–Bendix orders
session 3	Wednesday	background: eager approach, certification
		application: polynomial interpretations, max-poly certification
session 4	Friday	SAT/SMT for infeasibility and confluence, logically constraint TRSs

Outline

Definition (Propositional Logic: Syntax)

- propositional formulas are built from
  - atoms $p, q, r, p_1, p_2, \ldots$ propositional variables
  - top,bottom $\top, \bot$ "true" and "false"
  - negation $\neg$ $\neg p$ "not p"
  - conjunction $\wedge$ $p \wedge q$ "p and q"
  - disjunction $\vee$ $p \vee q$ "p or q"
  - implication $\rightarrow$ $p \rightarrow q$ "if p then q"
  - equivalence $\leftrightarrow$ $p \leftrightarrow q$ "p if and only if q"

  according to BNF grammar $\varphi ::= p \mid \bot \mid \top \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$
- notational conventions:
  - binding precedence $\neg > \wedge > \vee > \rightarrow, \leftrightarrow$ omit outer parentheses
  - $\rightarrow$, $\wedge$, $\vee$ are right-associative: $p \rightarrow q \rightarrow r$ denotes $p \rightarrow (q \rightarrow r)$

Definition (Propositional Logic: Semantics)

- valuation (truth assignment) is mapping $v\colon \{p \mid p \text{ is atom}\} \to \{\mathsf{T}, \mathsf{F}\}$

  truth values
- extension to formulas:
  - $v(\top) = \mathsf{T}$
  - $v(\bot) = \mathsf{F}$
  - $v(\neg\varphi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = \mathsf{F} \\ \mathsf{F} & \text{otherwise} \end{cases}$
  - $v(\varphi \wedge \psi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = v(\psi) = \mathsf{T} \\ \mathsf{F} & \text{otherwise} \end{cases}$
  - $v(\varphi \vee \psi) = \begin{cases} \mathsf{F} & \text{if } v(\varphi) = v(\psi) = \mathsf{F} \\ \mathsf{T} & \text{otherwise} \end{cases}$
  - $v(\varphi \to \psi) = \begin{cases} \mathsf{F} & \text{if } v(\varphi) = \mathsf{T} \text{ and } v(\psi) = \mathsf{F} \\ \mathsf{T} & \text{otherwise} \end{cases}$
  - $v(\varphi \leftrightarrow \psi) = \begin{cases} \mathsf{T} & \text{if } v(\varphi) = v(\psi) \\ \mathsf{F} & \text{otherwise} \end{cases}$

Definitions

- semantic entailment

$$\varphi_1, \varphi_2, \ldots, \varphi_n \vDash \psi$$

  if $v(\psi) = \mathsf{T}$ whenever $v(\varphi_1) = v(\varphi_2) = \cdots = v(\varphi_n) = \mathsf{T}$, for every valuation v
- formula φ is valid if $v(\varphi) = \mathsf{T}$ for every valuation v
- formula φ is satisfiable if $v(\varphi) = \mathsf{T}$ for some valuation v

Theorem

- formula φ is valid $\iff$ $\neg\varphi$ is unsatisfiable
- validity and satisfiability are decidable

Satisfiability (SAT)

instance: (propositional) formula φ

question: is φ satisfiable?

Outline

SAT Applications

Applications of SAT

- Encode logic puzzles
- Cryptanalysis
- Bounded model checking
- . . .
- Component of reasoning in more complex logics (sessions 2 and 3)
- Encode non-deterministic computations (SAT is NP complete)
- Encode problems in proof search, e.g., in context of term rewriting

Application: SAT for LPO Parameter Search

Definition (Lexicographic Path Order (LPO))

- Let $\mathcal{F}$ be some first-order signature
- Let $p : \mathcal{F} \to \mathbb{N}$ be some precedence
- LPO is a relation on terms $\succ_{LPO}$ ($\succ$ for short), defined by these inference rules

$$\frac{s_i \succ t \lor s_i = t}{s = f(s_1, \ldots, s_n) \succ t} \quad \text{(sub)}$$

$$\frac{p(f) > p(g) \quad \forall i \in \{1, \ldots, m\}.\ s \succ t_i}{s = f(\ldots) \succ g(t_1, \ldots, t_m) = t} \quad \text{(prec)}$$

$$\frac{\forall j \in \{1, \ldots, i-1\}.\ s_j = t_j \quad s_i \succ t_i \quad \forall j \in \{i+1, \ldots, n\}.\ s \succ t_i}{s = f(s_1, \ldots, s_n) \succ f(t_1, \ldots, t_n) = t} \quad \text{(lex)}$$

Theorem

LPO is a reduction order (stable, monotone, strongly normalizing)

Example

Consider a TRS for the Ackermann function

$$\begin{aligned} \mathsf{ack}(0, m) &\to \mathsf{s}(m) \\ \mathsf{ack}(\mathsf{s}(n), 0) &\to \mathsf{ack}(n, \mathsf{s}(0)) \\ \mathsf{ack}(\mathsf{s}(n), \mathsf{s}(m)) &\to \mathsf{ack}(n, \mathsf{ack}(\mathsf{s}(n), m)) \end{aligned}$$

assuming $p(\mathsf{ack}) > p(\mathsf{s})$, all rules are decreasing w.r.t. LPO;
witness for second rule

$$\dfrac{\dfrac{n = n}{\mathsf{s}(n) \succ n}\,(\mathrm{sub}) \qquad \dfrac{p(\mathsf{ack}) > p(\mathsf{s}) \qquad \dfrac{0 = 0}{\mathsf{ack}(\mathsf{s}(n), 0) \succ 0}\,(\mathrm{sub})}{\mathsf{ack}(\mathsf{s}(n), 0) \succ \mathsf{s}(0)}\,(\mathrm{prec})}{\mathsf{ack}(\mathsf{s}(n), 0) \succ \mathsf{ack}(n, \mathsf{s}(0))}\,(\mathrm{lex})$$

A Search Problem for Termination Proving

Theorem

The following "LPO-problem" is NP-complete.

Given some TRS $\mathcal{R}$, is there some precedence such that $\ell \succ_{LPO} r$ for all $\ell \to r \in \mathcal{R}$?

An opportunity

Since the LPO-problem is in NP, and SAT is NP-complete, we can encode the LPO-problem to SAT

- in early times, dedicated solvers have been implemented to search for precedences
- encoding to SAT is by far simpler and also quite flexible w.r.t. extensions
- experiments revealed: due to high efficiency of modern SAT solvers, the encoding approach is faster than existing dedicated solvers
- encoding problems to SAT: bit-blasting

Encoding of LPO

- first consider the search for an inference tree (postpone the precedence encoding)
- given two terms s and t we construct an encoding as formula $\varphi_{s \succ t}$
- since $\succ$ only occurs positively, soundness suffices:

$$\text{satisfiability of } \varphi_{s \succ t} \text{ implies } s \succ t$$

- $\varphi_{s \succ t}$ is a large conjunction, each conjunct is called a constraint
- for every $s_i \trianglelefteq s$ and $t_j \trianglelefteq t$ we use one propositional variable $\ulcorner s_i \succ t_j \urcorner$
- add constraint $\ulcorner s \succ t \urcorner$ to $\varphi_{s \succ t}$
- add the following constraints to $\varphi_{s \succ t}$ for all subterm pairs of s and t
  - $\ulcorner x \succ t_j \urcorner \to \bot$
  - $\ulcorner f(s_1, \ldots, s_n) \succ y \urcorner \to \bigvee_{i \in \{1,\ldots,n\}} \ulcorner s_i \succeq y \urcorner$
  - $\ulcorner f(s_1, \ldots, s_n) \succ g(t_1, \ldots, t_m) \urcorner \to$
    $\bigvee_{i \in \{1,\ldots,n\}} \ulcorner s_i \succeq g(t_1, \ldots, t_m) \urcorner \vee \ulcorner p(f) > p(g) \urcorner \wedge \bigwedge_{j \in \{1,\ldots,m\}} \ulcorner f(s_1, \ldots, s_n) \succ t_j \urcorner$ if $f \neq g$
  - $\ulcorner f(s_1, \ldots, s_n) \succ f(t_1, \ldots, t_n) \urcorner$: similar, encode (sub) or (lex)
  - remark: $\ulcorner s_i \succeq t_j \urcorner := \top$, if $s_i = t_j$, and $\ulcorner s_i \succeq t_j \urcorner := \ulcorner s_i \succ t_j \urcorner$, otherwise

Remarks

- encoding size: $\mathcal{O}(n^3)$ with $\mathcal{O}(n^2)$ variables
- optimizations
  - sharing: if same subterm pair occurs several times, only use one atom
  - static analysis: use knowledge about LPO to reduce encoding size
    - short cuts: $\ulcorner f(s_1,\dots,s_n) \succ y \urcorner \to \begin{cases} \top, & \text{if } y \in \mathcal{V}(f(s_1,\dots,s_n)) \\ \bot, & \text{otherwise} \end{cases}$
    - early successes: $\ulcorner s_i \succ t_j \urcorner \to \top$ if $s_i \rhd t_j$
    - early failures: $\ulcorner s_i \succ t_j \urcorner \to \bot$ if $\mathcal{V}(s_i) \not\supseteq \mathcal{V}(t_j)$ or $s_i \unlhd t_j$
- example on $\mathsf{ack}(\mathsf{s}(n),\mathsf{0}) \succ \mathsf{ack}(n,\mathsf{s}(\mathsf{0}))$
  - $\ulcorner ack(s(n),0) \succ ack(n,s(0)) \urcorner \to \ulcorner s(n) \succ ack(n,s(0)) \urcorner \vee \ulcorner ack(s(n),0) \succ s(0) \urcorner$
  - $\ulcorner s(n) \succ ack(n,s(0)) \urcorner \to \ulcorner p(\mathsf{s}) > p(\mathsf{ack}) \urcorner \wedge \ulcorner s(n) \succ s(0) \urcorner$
  - $\ulcorner ack(s(n),0) \succ s(0) \urcorner \to \ulcorner p(\mathsf{ack}) > p(\mathsf{s}) \urcorner \vee \ulcorner s(n) \succ s(0) \urcorner$
  - $\ulcorner s(n) \succ s(0) \urcorner \to \bot$
  - bottom-up computation: $\ulcorner ack(s(n),0) \succ ack(n,s(0)) \urcorner \to \ulcorner p(\mathsf{ack}) > p(\mathsf{s}) \urcorner$

Encoding of Precedence

- for signature $\mathcal{F}$ with $|\mathcal{F}| = n$ it suffices to guess $p(f) \in \{0, \ldots, n-1\}$ for each $f \in \mathcal{F}$
- several possibilities
  - encode $p(f)$ as tally sequence in $n-1$ atoms and $\ulcorner p(f) > p(g) \urcorner$ uses unary comparison
    - example for $n = 8$ and $p(f) = 3$: 0000111
    - comparison: $f_6 f_5 f_4 f_3 f_2 f_1 f_0 > g_6 g_5 g_4 g_3 g_2 g_1 g_0$ becomes $\bigvee_{i \in \{0,\ldots,6\}} f_i \wedge \neg g_i$
    - invariant: $\bigwedge_{f \in \mathcal{F}} \bigwedge_{i \in \{1,\ldots,6\}} (f_i \to f_{i-1})$
    - advantage: good structure for SAT solvers
    - disadvantage: large size
  - encode $p(f)$ in $log(n)$ atoms and $\ulcorner p(f) > p(g) \urcorner$ uses binary comparison
    - example for $n = 8$ and $p(f) = 3$: 011
    - comparison: $f_2 f_1 f_0 > g_2 g_1 g_0$ becomes $f_2 \wedge \neg g_2 \vee (g_2 \to f_2) \wedge (f_1 \wedge \neg g_1 \vee (g_1 \to f_1) \wedge f_0 \wedge \neg g_0)$
    - advantage: small size
    - disadvantage: more complex structure for SAT solving
  - use stronger logic than SAT, e.g., SMT with arithmetic primitives (see next sessions)
- selecting suitable encoding is often done with help of experiments

Summary of LPO encoding

- the search for parameters of LPO and similar orders can be encoded to SAT
- this bit-blasting approach is usually faster than dedicated solvers
- fact: many tools for (termination | confluence) analysis use SAT or SMT solvers

Exercise

- LPO on its own is quite weak for termination proving
- preprocessing term order constraints by argument filters greatly improves power
- an AF is a function π that maps every n-ary function symbol to some argument position, or to a subset of argument positions
- $\pi(x) = x$
- $\pi(f(t_1, \dots, t_n)) = \begin{cases} \pi(t_i), & \text{if } \pi(f) = i \\ f([\pi(t_i) \mid i \leftarrow [1..n], i \in \pi(f)]), & \text{if } \pi(f) \text{ is a set} \end{cases}$
- given s and t, encode whether there is some π and LPO such that $\pi(s) \succ_{LPO} \pi(t)$
- hints: (1) ; (2)

Outline

Remark

- many SAT solvers require conjunctive normal form (CNF) as input
- CNFs have the following structure
  - a literal is an atom or a negated atom: $x, \neg y, \ldots$
  - a clause is disjunction of literals: $x \vee z \vee \neg y$ or short: $\{x, z, \neg y\}$
  - a CNF is a conjunction of clauses

DIMACS Input Format

`c`	
`c comments`	
`c`	
`p cnf 4 3`	4 atoms and 3 clauses
`1 -2 4 0`	$x_1 \vee \neg x_2 \vee x_4$
`-1 2 -3 -4 0`	$\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4$
`3 -2 0`	$x_3 \vee \neg x_2$

Accessing SAT Solvers

- look at recent SAT competitions to find solver

  `https://satcompetition.github.io`

- either use DIMACS and binary of arbitrary solver
- or search for language binding, e.g.,

  `https://hackage.haskell.org/package/minisat-solver-0.1/candidate/docs/SAT-MiniSat.html`

- example

```
cd [USB-stick]/ISR24_sat_smt_isr_2024
cabal install miniTT
miniTT encoding lpo  examples/ack.ari    # see basic encoding
miniTT encoding lpo2 examples/ack.ari    # see optimized encoding
miniTT txt      lpo  examples/ack.ari    # textual proof output
# inspect miniTT/src/LPO.hs miniTT/src/LPO2.hs
```

Remarks

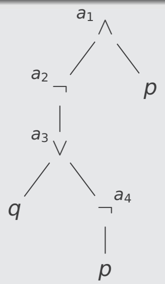
- translation from arbitrary formula to equivalent CNF is expensive
- Tseitin's transformation is linear-time translation to equisatisfiable CNF
- here: only consider formulas without $\rightarrow$ and $\leftrightarrow$

Example (Tseitin's Transformation)

- $\varphi = \neg(q \vee \neg p) \wedge p$
- introduce new variable for each propositional connective:

a_1	$\neg(q \vee \neg p) \wedge p$	a_3	$q \vee \neg p$
a_2	$\neg(q \vee \neg p)$	a_4	$\neg p$

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



Lemma

1. $(\varphi \leftrightarrow \neg\psi) \equiv (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$
2. $(\varphi \leftrightarrow \psi \wedge \chi) \equiv (\neg\varphi \vee \psi) \wedge (\neg\varphi \vee \chi) \wedge (\varphi \vee \neg\psi \vee \neg\chi)$
3. $(\varphi \leftrightarrow \psi \vee \chi) \equiv (\varphi \vee \neg\psi) \wedge (\varphi \vee \neg\chi) \wedge (\neg\varphi \vee \psi \vee \chi)$

Example (cont'd)

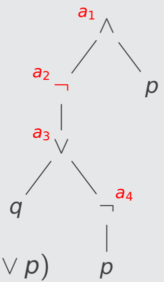
$$\begin{aligned}\varphi &\approx a_1 \wedge (a_1 \leftrightarrow a_2 \wedge p) \wedge (a_2 \leftrightarrow \neg a_3) \wedge (a_3 \leftrightarrow q \vee a_4) \wedge (a_4 \leftrightarrow \neg p) \\ &\equiv a_1 \wedge (\neg a_1 \vee a_2) \wedge (\neg a_1 \vee p) \wedge (a_1 \vee \neg a_2 \vee \neg p) \wedge (a_2 \vee a_3) \wedge (\neg a_2 \vee \neg a_3) \\ &\quad \wedge (a_3 \vee \neg q) \wedge (a_3 \vee \neg a_4) \wedge (\neg a_3 \vee q \vee a_4) \wedge (a_4 \vee p) \wedge (\neg a_4 \vee \neg p)\end{aligned}$$

Improvement (Plaisted & Greenbaum)

replace equivalence ($\leftrightarrow$) by implication ($\rightarrow$ or $\leftarrow$) based on polarity of subformulas

Example (cont'd)

- $\varphi = \neg(q \vee \neg p) \wedge p$
- $\varphi \approx a_1 \wedge (a_1 \rightarrow a_2 \wedge p) \wedge (a_2 \rightarrow \neg a_3) \wedge (a_3 \leftarrow q \vee a_4) \wedge (a_4 \leftarrow \neg p)$
- $a_1 \rightarrow a_2 \wedge p \equiv (\neg a_1 \vee a_2) \wedge (\neg a_1 \vee p) \wedge (a_1 \vee \neg a_2 \vee \neg p)$
- $a_2 \rightarrow \neg a_3 \equiv (a_2 \vee a_3) \wedge (\neg a_2 \vee \neg a_3)$
- $a_3 \leftarrow q \vee a_4 \equiv (a_3 \vee \neg q) \wedge (a_3 \vee \neg a_4) \wedge (\neg a_3 \vee q \vee a_4)$
- $a_4 \leftarrow \neg p \equiv (a_4 \vee p) \wedge (\neg a_4 \vee \neg p)$
- $\varphi \approx a_1 \wedge (\neg a_1 \vee a_2) \wedge (\neg a_1 \vee p) \wedge (\neg a_2 \vee \neg a_3) \wedge (a_3 \vee \neg q) \wedge (a_3 \vee \neg a_4) \wedge (a_4 \vee p)$



replace $a \leftrightarrow \psi$ by $a \rightarrow \psi$ if ψ occurs only positively, and by $a \leftarrow \psi$ if ψ never occurs positively

Definition

subformula ψ occurs positively in formula φ if number of negations on path from root of φ to root of ψ in parse tree of φ is even

Outline

Remarks

- most state-of-the-art SAT solvers are based on variations of Davis – Putnam – Logemann – Loveland (DPLL) procedure (1960, 1962)
- abstract version of DPLL described in JACM paper of Nieuwenhuis, Oliveras, Tinelli (2006)

Definition (Abstract DPLL)

- states $M \parallel F$ consist of
  - list M of (possibly annotated) non-complementary literals
  - CNF F
- transition rules

$$M \parallel F \quad \Longrightarrow \quad M' \parallel F' \quad \text{or} \quad \text{fail-state}$$

Example

$\varphi = (\neg 1 \vee \neg 2) \wedge (2 \vee 3) \wedge (\neg 1 \vee \neg 3 \vee 4) \wedge (2 \vee \neg 3 \vee \neg 4) \wedge (1 \vee 4)$

$$
\begin{array}{rrcll}
 & & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \\
\Longrightarrow & 1^d & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{decide} \\
\Longrightarrow & 1^d\ \neg 2 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & 1^d\ \neg 2\ 3 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & 1^d\ \neg 2\ 3\ 4 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & \neg 1 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{backtrack} \\
\Longrightarrow & \neg 1\ 4 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{unit propagate} \\
\Longrightarrow & \neg 1\ 4\ \neg 3^d & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{decide} \\
\Longrightarrow & \neg 1\ 4\ \neg 3^d\ 2 & \| & \neg 1 \vee \neg 2,\ 2 \vee 3,\ \neg 1 \vee \neg 3 \vee 4,\ 2 \vee \neg 3 \vee \neg 4,\ 1 \vee 4 & \text{unit propagate}
\end{array}
$$

Definition (Transition Rules)

- unit propagate $\qquad M \parallel F, C \vee l \implies M\, l \parallel F, C \vee l$

  if $M \vDash \neg C$ and l is undefined in M unit clause

- pure literal $\qquad M \parallel F \implies M\, l \parallel F$

  if l occurs in F and l^c (complement of l) does not occur in F and l is undefined in M

- decide $\qquad M \parallel F \implies M\, \overset{d}{l} \parallel F$

  if l or l^c occurs in F and l is undefined in M

- fail $\qquad M \parallel F, C \implies$ fail-state

  if $M \vDash \neg C$ and M contains no decision literals

- backtrack $\qquad M\, \overset{d}{l}\, N \parallel F, C \implies M\, l^c \parallel F, C$

  if $M\, \overset{d}{l}\, N \vDash \neg C$ and N contains no decision literals

Example

$\varphi = (\neg 1 \vee 2) \wedge (\neg 3 \vee 4) \wedge (\neg 5 \vee \neg 6) \wedge (6 \vee \neg 5 \vee \neg 2)$

$$
\begin{array}{rrll}
 & & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \\
\Longrightarrow & 1^d & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{decide} \\
\Longrightarrow & 1^d\ 2 & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{unit propagate} \\
\Longrightarrow & 1^d\ 2\ 3^d & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{decide} \\
\Longrightarrow & 1^d\ 2\ 3^d\ 4 & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{unit propagate} \\
\Longrightarrow & 1^d\ 2\ 3^d\ 4\ 5^d & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{decide} \\
\Longrightarrow & 1^d\ 2\ 3^d\ 4\ 5^d\ \neg 6 & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{unit propagate} \\
\Longrightarrow & 1^d\ 2\ \neg 5 & \| \ \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee \neg 2 & \text{backjump}
\end{array}
$$

conflict is due to $\overset{d}{1}\ 2$ and $\overset{d}{5}\ \neg 6$ hence $\neg 1 \vee \neg 5$ can be inferred

Definitions

- backtrack $\quad M \overset{d}{l} N \parallel F, C \implies M\, l^c \parallel F, C$

  if $M \overset{d}{l} N \vDash \neg C$ and N contains no decision literals

- backjump $\quad M \overset{d}{l} N \parallel F, C \implies M\, l' \parallel F, C$

  if $M \overset{d}{l} N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that

  backjump clause

  - $F, C \vDash C' \vee l'$
  - $M \vDash \neg C'$
  - l' is undefined in M
  - l' or l'^c occurs in F or in $M \overset{d}{l} N$

Example (cont'd)

$\neg 1 \vee \neg 5$ and $\neg 2 \vee \neg 5$ are backjump clauses with respect to $\overset{d}{1}\ 2\ \overset{d}{3}\ 4\ \overset{d}{5}\ \neg 6 \parallel \varphi$

Lemma

backjump can simulate backtrack

Terminology

backjump is also called non-chronological backtracking or conflict-driven backtracking

Question

how to find good backjump clauses ?

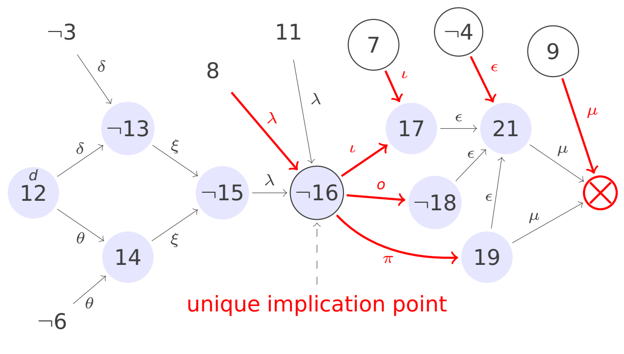
Answer

use conflict graph

Example

α	1
β	$2 \vee \neg 3$
γ	$2 \vee \neg 4$
δ	$3 \vee \neg 12 \vee \neg 13$
ϵ	$4 \vee \neg 17 \vee 18 \vee \neg 19 \vee 21$
ζ	$\neg 5 \vee \neg 6$
η	$6 \vee 7$
θ	$6 \vee \neg 12 \vee 14$
ι	$\neg 7 \vee 16 \vee 17$
κ	$\neg 8 \vee 9$
λ	$\neg 8 \vee \neg 11 \vee 15 \vee \neg 16$
μ	$\neg 9 \vee \neg 19 \vee \neg 21$
ν	$10 \vee 11$
ξ	$13 \vee \neg 14 \vee \neg 15$
o	$16 \vee \neg 18$
π	$16 \vee 19$
ρ	$\neg 19 \vee 20$

$$\underset{\alpha}{1}\ \overset{d}{\neg 2}\ \underset{\beta}{\neg 3}\ \underset{\gamma}{\neg 4}\ \overset{d}{5}\ \underset{\zeta}{\neg 6}\ \underset{\eta}{7}\ \overset{d}{8}\ \underset{\kappa}{9}\ \overset{d}{\neg 10}\ \underset{\nu}{11}\ \overset{d}{12}\ \underset{\delta}{\neg 13}\ \underset{\theta}{14}\ \underset{\xi}{\neg 15}\ \underset{\lambda}{\neg 16}\ \underset{\iota}{17}\ \underset{o}{\neg 18}\ \underset{\pi}{19}\ \underset{\rho}{20}\ \underset{\epsilon}{21}$$



$\neg 9 \vee \neg 19 \vee \neg 21$	conflict clause (μ)
$4 \vee \neg 7 \vee \neg 9 \vee 16$	resolve with ϵ, π, o, ι

backjump clause

Remarks

- computed clauses are clauses that correspond to cut in conflict graph, separating conflict node from current decision literal and literals at earlier decision levels
- not all cuts are computed in this way
- clauses corresponding to UIPs are backjump clauses
- UIPs always exist (last decision literal)
- backjumping with respect to last UIP amounts to backtracking
- most SAT solvers use backjump clause corresponding to 1st UIP

Observation

adding backjump clauses to clause database (learning) helps to prune search space

- learn $\quad M \parallel F \quad \Longrightarrow \quad M \parallel F, C$

  if $F \vDash C$ and each atom of C occurs in F or in M

Observation

restarts are useful to avoid wasting too much time in parts of search space without satisfying assignments

- restart $\qquad M \parallel F \quad \Longrightarrow \quad \parallel F$

Final Remarks

- restarts do not compromise completeness if number of steps between consecutive restarts strictly increases
- modern SAT solvers additionally incorporate
  - heuristics for selecting next decision literal
  - special data structures that allow for efficient unit propagation (two watched literals)

Outline

Further Reading

- David A. Plaisted and Steven Greenbaum
  A Structure-Preserving Clause Form Translation
  Journal of Symbolic Computation 2(3), pp. 293 – 304, 1986
- Michael Codish, Vitaly Lagoon, Peter J. Stuckey
  Solving Partial Order Constraints for LPO Termination
  Proceedings RTA 2006, pp. 4 – 18, 2006
- Martin Davis and Hilary Putnam
  A Computing Procedure for Quantification Theory
  Journal of the ACM 7(3), pp. 201 – 215, 1960
- 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
- Moshe Y. Vardi
  Boolean Satisfiability: Theory and Engineering
  Communications of the ACM 57(3), editor's letter, 2014