universität
innsbruck

SAT and SMT Solving

Sarah Winkler

KRDB
Department of Computer Science
Free University of Bozen-Bolzano

lecture 2
WS 2022

Outline

- Summary of Last Week
- From DPLL to Conflict Driven Clause Learning
- Application: Test Case Generation

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
  - 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
  - $F, C \models C' \vee l'$ $\quad$ backjump clause
  - $M \models \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Definition

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

Theorem (Termination)

there are no infinite derivations $\quad \| \; F \quad \Longrightarrow_{\mathcal{B}} \quad S_1 \quad \Longrightarrow_{\mathcal{B}} \quad S_2 \quad \Longrightarrow_{\mathcal{B}} \quad \dots$

Theorem (Correctness)

for derivation with final state S_n:

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

- *if $S_n = \text{FailState}$ then F is unsatisfiable*
- *if $S_n = M \,\|\, F'$ then F is satisfiable and $M \vDash F$*

Definition

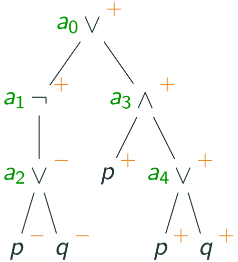
polarity of subformula φ in ψ is $+$ if number of negations above φ in ψ is even, and $-$ otherwise

Example (Efficient Transformations to CNF)

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

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

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



- Tseitin: add clause a_0 plus $(a_i \leftrightarrow \dots)$ for every subformula

  $$\begin{aligned}\varphi \approx \quad & a_0 \wedge (a_0 \leftrightarrow a_1 \vee a_3) \wedge (a_1 \leftrightarrow \neg a_2) \wedge (a_2 \leftrightarrow p \vee q) \wedge \\ & (a_3 \leftrightarrow p \wedge a_2)\end{aligned}$$

- Plaisted & Greenbaum: $(a_i \rightarrow \dots)$ if polarity of a_i is $+$ and $(a_i \leftarrow \dots)$ if $-$

  $$\begin{aligned}\varphi \approx \quad & a_0 \wedge (a_0 \rightarrow a_1 \vee a_3) \wedge (a_1 \rightarrow \neg a_2) \wedge (a_2 \leftarrow p \vee q) \wedge \\ & (a_3 \rightarrow p \wedge a_4) \wedge (a_4 \rightarrow p \vee q)\end{aligned}$$

- replace $\leftrightarrow$ and $\rightarrow$ by 2 or 3 clauses each

Outline

- Summary of Last Week
- From DPLL to Conflict Driven Clause Learning
  - Conflict Analysis
  - Heuristics and Data Structures
- Application: Test Case Generation

Conflict Driven Clause Learning (CDCL)

```
function dpll(φ)
 M = []
 while (true)
   if all_variables_assigned(M)
     return satisfiable
   M = decide(φ, M)
   M = unit_propagate(φ, M)
   if (conflict(φ, M))
     try
       M,C = backjump(φ, M)
       φ = φ ∪ {C}
     catch (fail_state)
       return unsatisfiable
   φ = forget(φ)
   if (do_restart(M))
     return dpll(φ)
```

choice of decision literals matters for performance

more than 90% of time spent in unit propagation

backjump clauses are useful: learn them!

forgetting implied clauses improves performance

occasional restarts improve performance

Definition (CDCL)

CDCL system $\mathcal{R}$ extends DPLL system $\mathcal{B}$ by following three rules:

- learn $$M \parallel F \quad \Longrightarrow \quad M \parallel F,\ C$$
  if $F \models C$ and all atoms of C occur in M or F
- forget $$M \parallel F,\ C \quad \Longrightarrow \quad M \parallel F$$
  if $F \models C$
- restart $$M \parallel F \quad \Longrightarrow \quad \parallel F$$

Theorem (Termination)
any derivation $\parallel F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*
- *restart is applied with increasing periodicity*

Theorem (Correctness)
for derivation with final state S_n*:*

$$\parallel F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots \Longrightarrow_{\mathcal{R}} S_n$$

- *if* $S_n = \mathsf{FailState}$ *then* F *is unsatisfiable*
- *if* $S_n = M \parallel F'$ *then* F *is satisfiable and* $M \models F$

Outline

- Summary of Last Week
- From DPLL to Conflict Driven Clause Learning
  - Conflict Analysis
  - Heuristics and Data Structures
- Application: Test Case Generation

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$

Backjump to Definition

- backjump $\qquad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$

  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$$\underbrace{1^d\ 2}_{M}\ \ \underbrace{3^d}_{l}\ \ \underbrace{4^d\ \overline{5}}_{N} \parallel \underbrace{\overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \boxed{\overline{4} \vee 5},\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}}_{F,C}$$

$$\Longrightarrow\ 1^d\ 2\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$$

$M = 1^d\ 2 \qquad l = 3 \qquad N = 4^d\ \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\ 2\ 3^d\ 4^d\ \overline{5}\ \vDash\ \neg(\overline{4} \vee 5)$
- backjump clause $C' \vee l' = \overline{1} \vee \overline{5}$ satisfies $F, C \vDash C' \vee l'$
- $1^d\ 2 \vDash 1$,and 5 is undefined in $1^d\ 2$ but occurs in F

Outline

- Summary of Last Week
- From DPLL to Conflict Driven Clause Learning
  - Conflict Analysis
  - Heuristics and Data Structures
- Application: Test Case Generation

Desirable Properties of Backjump Clauses

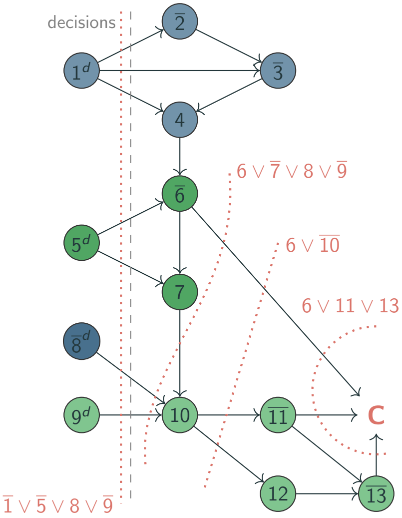
- small
- should trigger progress

How to Determine Backjump Clauses?

- implication graph
- resolution

Example: Implication Graph

$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

next

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l
- first UIP is UIP closest to conflict node

Key Observations

- if $l_1 \to l_1', \dots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause
- last decision literal is UIP
- backjump clause: cut with exactly one literal l at last decision level (l is UIP)

Example

- cuts: $\overline{1} \vee \overline{5} \vee 8 \vee \overline{9}$ $\quad 6 \vee 11 \vee 13$ $\quad 6 \vee \overline{10}$ $\quad 6 \vee \overline{7} \vee 8 \vee \overline{9}$
- UIPs are 9 and 10
- first UIP is 10

Definition (Implication Graph)
Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \dots \vee l_k'$ in F such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

potential backjump clause

Lemma
if edges intersected by cut are $l_1 \to l_1', \dots, l_k \to l_k'$ *then* $F \vDash l_1^c \vee \dots \vee l_k^c$

Resolution

Remarks

- keeping track of implication graph is too expensive in practice
- compute clauses associated with cuts by resolution instead

Definition (Resolution)

$$\frac{C \vee l \qquad C' \vee \neg l}{C \vee C'}$$

(assuming literals in clauses can be reordered)

Example

$$\frac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}}$$

How to Derive Backjump Clause by Resolution

- let C_0 be the conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \lor 11 \lor 13$
- $C_1 = 6 \lor 11 \lor \overline{12}$
- $C_2 = 6 \lor 11 \lor \overline{10}$
- $C_3 = 6 \lor \overline{10}$
- $C_4 = 6 \lor \overline{7} \lor 8 \lor \overline{9}$

$$\frac{6 \lor 11 \lor 13 \qquad \overline{12} \lor \overline{13}}{6 \lor 11 \lor \overline{12}} \qquad \overline{10} \lor 12$$

$$\frac{10}{}$$

Decision Variable Selection

Observations

- choice of next decision variable is critical
- prefer variables that participated in recent conflict

VSIDS: Variable State Independent Decaying Sum

- first presented in SAT solver Chaff (2001)
- variant of this heuristic nowadays implemented in most CDCL solvers
- compute score for each variable, select variable with highest score
  - initial variable score is number of literal occurrences
  - learned (conflict) clause C: increment score for all variables in C
  - periodically divide all scores by constant

Example (VSIDS)

$$\| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5}$$

initial scores: $\{1 \mapsto 4,\ 2 \mapsto 4,\ 3 \mapsto 5,\ 4 \mapsto 4,\ 5 \mapsto 2\}$

$$\Longrightarrow\ 3^d \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5}$$

$$\Longrightarrow\ 3^d 1 \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5}$$

$$\Longrightarrow\ 3^d 1 4^d \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5}$$

$$\Longrightarrow^*\ 3^d 1 \overline{4} \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5},\ \overline{4} \vee \overline{3}$$

after adding learned clause: $\{1 \mapsto 4,\ 2 \mapsto 4,\ 3 \mapsto 6,\ 4 \mapsto 5,\ 5 \mapsto 2\}$

division by 2: $\{1 \mapsto 2,\ 2 \mapsto 2,\ 3 \mapsto 3,\ 4 \mapsto \frac{5}{2},\ 5 \mapsto 1\}$

$$\Longrightarrow^*\ \overline{3} \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5},\ \overline{4} \vee \overline{3},\ \overline{1} \vee \overline{3} \vee 4$$

after adding learned clause: $\{1 \mapsto 3,\ 2 \mapsto 2,\ 3 \mapsto 4,\ 4 \mapsto \frac{7}{2},\ 5 \mapsto 1\}$

$$\Longrightarrow^*\ \overline{3} 2 4^d \; \| \; 1 \vee \overline{2},\ 2 \vee \overline{3} \vee 4,\ \overline{1} \vee 4,\ \overline{4} \vee 3 \vee 5,\ 3 \vee \overline{5},\ \overline{3} \vee 1,\ \overline{1} \vee \overline{2},\ 2 \vee 3,\ \overline{4} \vee \overline{5},\ \overline{4} \vee \overline{3},\ \overline{1} \vee \overline{3} \vee 4$$

$$\Longrightarrow^*\ \text{FailState}$$

Efficient Unit Propagation?

Suppose input formula φ has n clauses and m literals in total.

Unit propagation in practice

- each unit propagation step requires to traverse entire formula φ $\mathcal{O}(m)$
- takes 90% of computation time when implemented naively

Observation

at any point of DPLL run, literal in clause is either true, false, or unassigned

First idea

- maintain counter how many false literals are in every clause C
- when assigning false to literal in clause, increment counter
- if counter is $|C| - 1$ and remaining literal unassigned, unit propagate $\mathcal{O}(n)$

Drawbacks

- upon backjump, must adjust all counters
- overhead to adjust counter if not yet $|C| - 1$

Two-Watched Literal Scheme

Idea

- maintain two pointers p_1 and p_2 for each clause C
- each pointer points to a literal in the clause that is:
  unassigned or true if possible, otherwise false
- ensure invariant that $p_1(C) \neq p_2(C)$

Key properties

- clause C enables unit propagation if $p_1(C)$ is false and $p_2(C)$ is unassigned literal
  or vice versa $\mathcal{O}(n)$
- clause C is conflict clause if $p_1(C)$ and $p_2(C)$ are false literals

Setting pointers

- initialization: set p_1 and p_2 to different (unassigned) literals in clause
- assigning variables by decide or unit propagate:
  when assigning literal l true, redirect all pointers to l^c to other literal in their clause if possible
- backjump: no need to change pointers!

Example (Two-Watched literal scheme)

Outline

- Summary of Last Week
- From DPLL to Conflict Driven Clause Learning
- Application: Test Case Generation

Problem

given software system with n parameters, generate set of test cases which covers all problematic situations while being as small as possible

Pairwise Testing

- well-practiced software testing method
- observation: most faults are caused by interaction of at most two parameters

some combinations may be infeasible

Example (Testing on Mobile Phones)

property	values
storage	32GB, 64GB, 128GB
cores	2, 4, 8
camera	8MP, 12MP, 16MP
SIM	single, dual
OS	Android, iOS

(a) testing model for mobile phones

	storage	cores	camera	SIM	OS
1	128GB	4	12MP	single	Android
2	32GB	2	8MP	single	Android
3	64GB	2	12MP	dual	iOS
4	32GB	4	16MP	dual	iOS
5	64GB	8	16MP	single	Android
6	128GB	8	8MP	dual	iOS
7	128GB	2	12MP	dual	Android
8	32GB	8	16MP	single	iOS
9	64GB	4	8MP	single	iOS

(b) test case set with pairwise coverage

Encode Test Set of Fixed Size in SAT

- have n parameters, and parameter i has C_i values
- for all m test cases use variables x_{ij} meaning that parameter i has value j
- parameter j has exactly one value

$$\text{one\_value}(x_{j1}, \ldots, x_{jC_j}) = \bigvee_{1 \leqslant k \leqslant C_j} x_{jk} \wedge \bigwedge_{1 \leqslant k < k' \leqslant C_j} \neg x_{jk} \vee \neg x_{jk'}$$

- in test case every parameter has one value

$$\text{test\_case}(x_{11}, \ldots, x_{nC_n}) = \bigwedge_{1 \leqslant j \leqslant n} \text{one\_value}(x_{j1}, \ldots, x_{jC_j})$$

- constraints on test case can be expressed by formula constraints$(x_{11}, \ldots, x_{nC_n})$
- use overall encoding assuming set of parameter pairs P

$$\bigwedge_{1 \leqslant i \leqslant m} \text{test\_case}(\overline{x^i}) \wedge \text{constraints}(\overline{x^i}) \wedge \bigwedge_{(j,k),(j',k') \in P} \bigvee_{1 \leqslant i \leqslant m} x^i_{jk} \wedge x^i_{j'k'}$$

- Minimal test set can be found by repeating approach with smaller m

CDCL

João Marques-Silva, Inês Lynce, Sharad Malik.
Conflict-Driven Clause Learning SAT Solvers.
Handbook of Satisfiability 2021: 133-182.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik.
Chaff: Engineering an Efficient SAT Solver
DAC 2001: 530-535.