Lecture 12

Monday, March 2, 2020 5:03 PM

- Recap:

L9: Polynomial time algorithm for 2-SAT

? ↑ ↓ ?

L8: Polynomial time algorithm for Horn-SAT

L7: The Boolean satisfiability problem.

Unit propagation / DPLL

CNF formula:

$(l_{11}$ or l_{12} or $\cdots$ or $l_{1k_1})$ and

$(l_{21}$ or l_{22} or $\cdots$ or $l_{2k_2})$ and

$\vdots$

$(l_{n1}$ or l_{n2} or $\cdots$ or $l_{nk_n})$

Clause

Literal

Either a variable v_{22} or its complement $\overline{v_{22}}$

Unit propagation

Consider a clause $(a$ or $\overline{b}$ or $\overline{c})$

Partial assignment $\{a \mapsto$ false, $b \mapsto$ true$\}$

For the clause to be satisfied, it has to be the case that $c \mapsto$ false.

Might result in a domino effect.

Ex: a and $(\overline{a}$ or $b)$ and $(\overline{b}$ or $c)$

Repeat until fixpoint.

DPLL (CNF formula φ, Partial assignment s)

- $s :=$ UnitPropagate $(\varphi \;\; s)$
- If $\exists$ variable v s.t. $v \in S$ and $\overline{v} \in S$,

— $S := \text{UnitPropagate}(\varphi, S)$

— If $\exists$ variable v s.t. $v \in S$ and $\bar{v} \in S$, then return unsat (Check consistency)

$$\boxed{\text{Ex: } a \text{ and } (a \Rightarrow b) \text{ and } (b \Rightarrow \bar{a})}$$

— If $\forall$ variables v, either $v \in S$ or $\bar{v} \in S$, then: (Check completeness)

- Assert (every clause is satisfied)
- Return SAT

Ex: $\varphi = a$ and $(a \Rightarrow \bar{b})$ and $(\bar{b} \Rightarrow c)$

$S = \{a, \bar{b}, c\}$

— Assert: There has to exist a variable v s.t. neither v nor $\bar{v}$ occur is S.

Ex: $(a \wedge b \Rightarrow c)$ and $(c \Rightarrow \bar{a})$ and $(c \Rightarrow \bar{b})$

$S = \{\}$ (Empty partial assignment)

— Pick such a variable v.

Execute $a_+ = \text{DPLL}(\varphi, S \wedge v)$

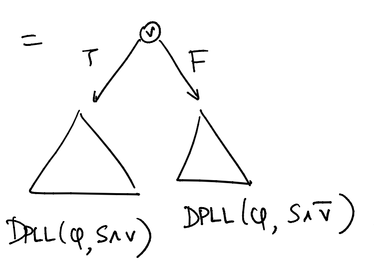
$a_- = \text{DPLL}(\varphi, S \wedge \bar{v})$

If either a_+ or $a_- = \text{SAT}$,

then return SAT.

Else return UNSAT

$\text{DPLL}(\varphi, S) =$



Clause Learning

$(x_1 \text{ or } x_4)$ and

$(x_1 \text{ or } \overline{x_3} \text{ or } \overline{x_8})$ and

$(x_1 \text{ or } x_8 \text{ or } x_{12})$ and

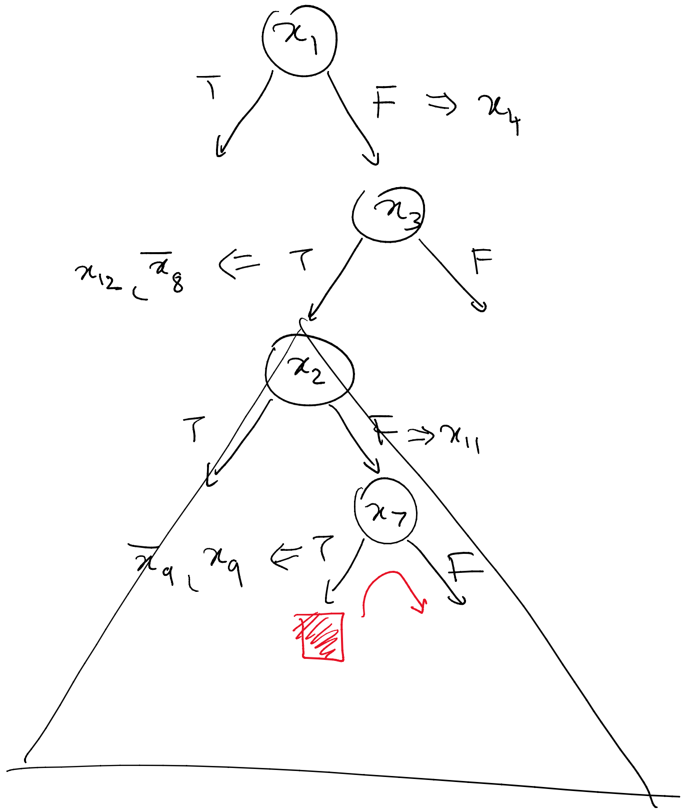
$(x_2 \text{ or } x_{11})$ and

C_5: $(\overline{x_7} \text{ or } \overline{x_3} \text{ or } x_9)$ and

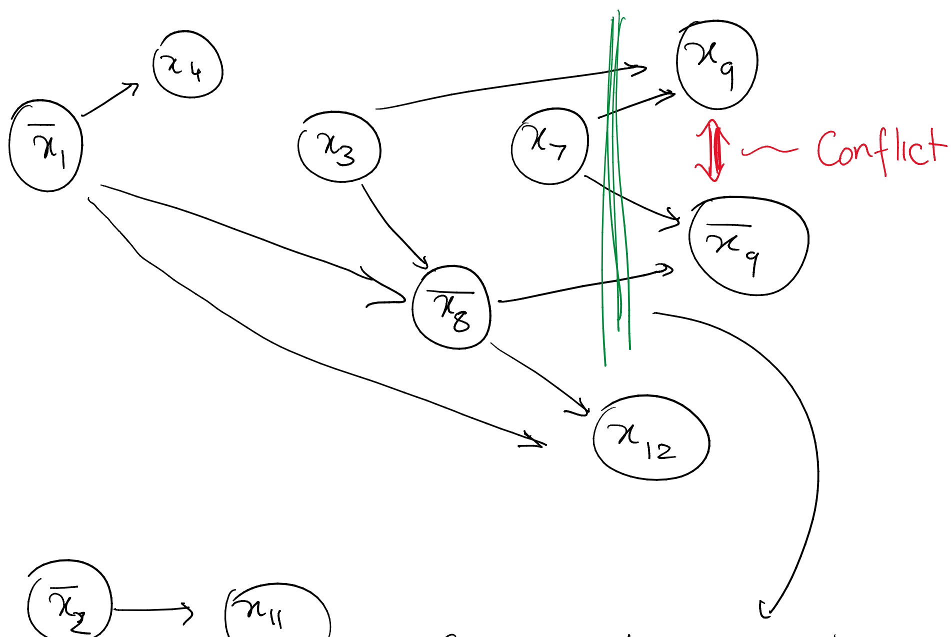
C_6: $(\overline{x_7} \text{ or } x_8 \text{ or } \overline{x_9})$ and

$(x_7 \text{ or } x_8 \text{ or } \overline{x_{10}})$ and

$(x_7 \text{ or } x_{10} \text{ or } \overline{x_{12}})$



Learned clause:
$\overline{x_3}$ or $\overline{x_7}$ or x_8



Conflict happened because

$t = x_3$ and x_7 and $\overline{x_8}$

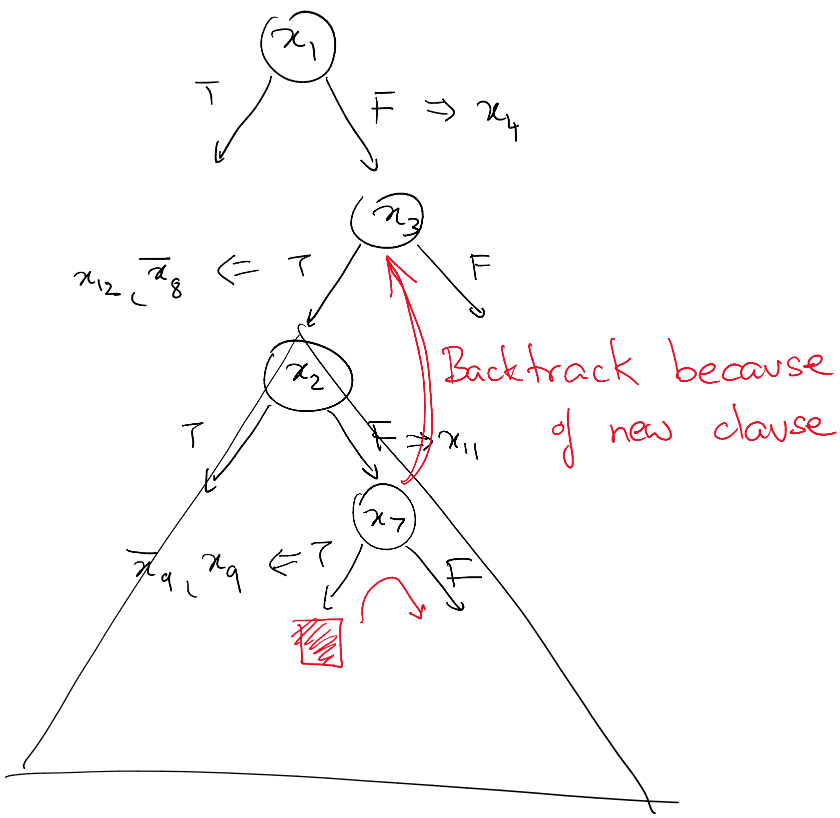
Therefore, in all satisfying assignments, Therefore, in all satisfying assignments,

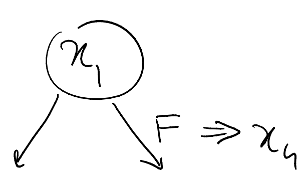
$$t = (\overline{x_3} \text{ or } \overline{x_7} \text{ or } x_8) \leftarrow \text{clause}$$

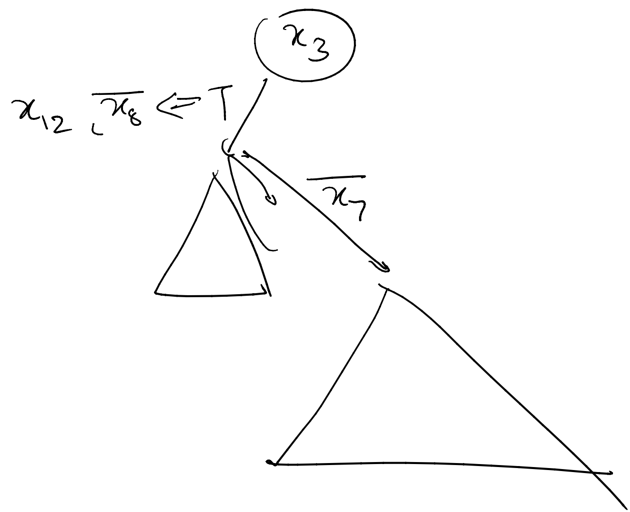
has to be true.

<u>Idea 1</u>: Learn new clauses by analysing conflict. $t = (\overline{x_3} \text{ or } \overline{x_7} \text{ or } x_8)$

<u>Idea 2</u>: Non-chronological backtracking







Restarts : Gomes et al., AAAI 1998

Boosting combinatorial search through randomization.

- Keep learned clauses
- Restart from last truth assignment

Restart after

- 1, 2, 4, 8, 16, ... conflicts (geometric progression)
- 1, 1, 2, 1, 1, 2, ... conflicts (Luby sequence)
- 128, 128, 128, ... conflicts (uniform sequence)

- Restarts work very well in practice.

  Unclear theoretical basis.

Decision heuristics

- Dynamic Largest Individual Sum

  Choose variable which satisfies most unsatisfied clauses.

- Jeroslow-Wang method

  For each literal l,

  $$J(l) = \sum_{\substack{\text{clauses in which} \\ l \text{ appears, } \omega}} 2^{-|\omega|}$$

- Variable State Independent Decaying Sum

  VSIDS

  c_v = # of clauses in which v occurs positively

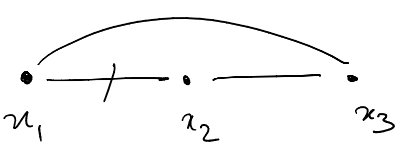
  $c_{\bar{v}}$ = # of clauses in which v occurs negatively.

- Make decisions to maximize this score

- Periodically divide all counters by a constant.

Satisfiability Modulo Theories (SMT)

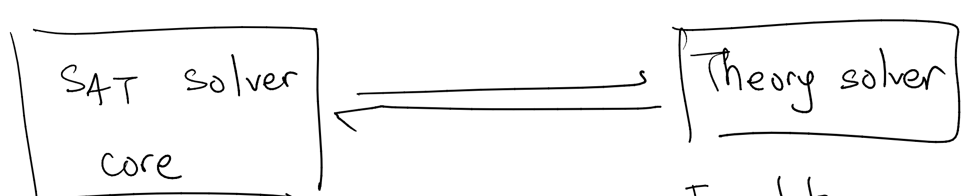
Ex: $\quad x_1 \neq x_2$ and $x_2 = x_3$ and $x_3 = x_1$



$$\underbrace{\text{not } (x_1 = x_2)}_{\bar{a}} \text{ and } \underbrace{(x_2 = x_3)}_{b} \text{ and } \underbrace{(x_3 = x_1)}_{c}$$

SMT: Level 1: Propositional core

Level 2: Each atomic proposition is interpreted over some theory



Equality

Equality with uninterpreted fns

Linear integer arithmetic

Difference logic

Arrays
Bitvectors
Strings

Difference logic

Difference between two integer-valued variables

$\exists x_1\ x_2\ x_3$ s.t. $\underbrace{x_1 - x_2 \leq 3}$ and — Difference $\leq$ constant

$x_2 - x_3 \leq 5$ and

$x_3 - x_1 \leq -16$

$$\begin{tikzcd} x_1 \arrow[rr, bend left, "-16"] & x_2 \arrow[l, "3"'] & x_3 \arrow[l, "5"'] \end{tikzcd}$$

$$\left.\begin{array}{l} x_2 \leq x_3 + 5 \\ x_1 \leq x_2 + 3 \end{array}\right\} \rightarrow x_1 \leq x_3 + 8$$

$x_3 \leq x_1 - 16$

$x_3 \leq x_3 - 8$

Absurd. Therefore unsat.

$\exists x_1\ x_2\ x_3$ s.t. $x_1 - x_2 \leq 3$ and

$x_2 - x_3 \leq 5$ and

$\exists x_1 \quad x_2 \quad x_3$

$x_2 - x_3 \leq 5$ and

$x_3 - x_1 \leq 16$

Constraints of the form $x \leq x + c$ (positive constant)

trivially satisfiable

Constraints of the form $x \leq x - c$ (positive constant)

obviously absurd.

Conjunction of propositions in difference logic is satisfiable iff no negative weight cycle exists in the graph.