# CS228 Logic for Computer Science 2022

## Lecture 10: SAT Solvers

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2022-01-24

# Propositional satisfiability problem

Consider a propositional logic formula $F$.

Find a model $m$ such that

$$m \models F.$$

## Example 10.1

*Give a model of $p_1 \wedge (\neg p_2 \vee p_3)$*

# CNF input

We assume that the input formula to a SAT solver is always in CNF.

Topic 10.1

DPLL (Davis-Putnam-Loveland-Logemann) method

# Notation: partial model

**Definition 10.1**
*We will call elements of* Vars $\hookrightarrow \mathcal{B}$ *as partial models.*

# Notation: state of a literal

Under partial model $m$,

a literal $\ell$ is true if $m(\ell) = 1$ and
$\ell$ is false if $m(\ell) = 0$.

Otherwise, $\ell$ is unassigned.

## Exercise 10.1
*Consider partial model $m = \{p_1 \mapsto 0, p_2 \mapsto 1\}$*

*What are the states of the following literals under $m$?*

▶ $p_1$

▶ $p_2$

▶ $p_3$

▶ $\neg p_1$

# Notation: state of a clause

Under partial model $m$,

a clause $C$ is true if there is $\ell \in C$ such that $\ell$ is true and
$\quad\quad\quad$ $C$ is false if for each $\ell \in C$, $\ell$ is false.

Otherwise, $C$ is unassigned.

## Exercise 10.2
*Consider partial model* $m = \{p_1 \mapsto 0, p_2 \mapsto 1\}$

*What are the states of the following clauses under* $m$*?*

- $p_1 \vee p_2 \vee p_3$
- $p_1 \vee \neg p_2$

- $p_1 \vee p_3$
- $\emptyset$ *(empty clause)*

# Notation: state of a formula

Under partial model $m$,

CNF $F$ is true if for each $C \in F$, $C$ is true and
  $F$ is false if there is $C \in F$ such that $C$ is false.

Otherwise, $F$ is unassigned.

## Exercise 10.3
*Consider partial model $m = \{p_1 \mapsto 0, p_2 \mapsto 1\}$*

*What are the states of the following formulas under $m$?*

- $(p_3 \vee \neg p_1) \wedge (p_1 \vee \neg p_2)$
- $(p_1 \vee p_2 \vee p_3) \wedge \neg p_1$

- $p_1 \vee p_3$
- $\emptyset$ *(empty formula)*

# Notation: unit clause and unit literal

## Definition 10.2
$C$ is a *unit clause* under $m$ if a literal $\ell \in C$ is unassigned and the rest are false. $\ell$ is called *unit literal*.

## Exercise 10.4
Consider partial model $m = \{p_1 \mapsto 0, p_2 \mapsto 1\}$

Are the following clauses unit under $m$? If yes, please identify the unit literals.

- $p_1 \vee \neg p_3 \vee \neg p_2$
- $p_1 \vee \neg p_3 \vee p_2$
- $p_1 \vee \neg p_3 \vee p_4$
- $p_1 \vee \neg p_2$

# DPLL (Davis-Putnam-Loveland-Logemann) method

DPLL

- ▶ maintains a partial model, initially $\emptyset$
- ▶ assigns unassigned variables 0 or 1 randomly one after another
- ▶ sometimes forced to choose assignments due to unit literals(why?)

# DPLL

## Algorithm 10.1: DPLL($F$)

**Input:** CNF $F$     **Output:** sat/unsat
**return** $DPLL(F, \emptyset)$

## Algorithm 10.2: DPLL($F$,$m$)

**Input:** CNF $F$, partial assignment $m$     **Output:** sat/unsat
**if** $F$ is true under $m$ **then** **return** sat ;
**if** $F$ is false under $m$ **then** **return** unsat ;
    <span style="color:purple">Backtracking at conflict</span>
**if** $\exists$ unit literal $p$ under $m$ **then**
  | **return** $DPLL(F, m[p \mapsto 1])$
    <span style="color:purple">Unit propagation</span>
**if** $\exists$ unit literal $\neg p$ under $m$ **then**
  | **return** $DPLL(F, m[p \mapsto 0])$
    <span style="color:purple">Decision</span>
Choose an unassigned variable $p$ and a random bit $b \in \{0, 1\}$;
**if** $DPLL(F, m[p \mapsto b]) ==$ sat **then**
  | **return** sat
**else**
  | **return** $DPLL(F, m[p \mapsto 1 - b])$

# Three actions of DPLL

A DPLL run consists of three types of actions

- ▶ Decision
- ▶ Unit propagation
- ▶ Backtracking

## Exercise 10.5
*What is the worst case complexity of DPLL?*

# Example: decide, propagate, and backtrack in DPLL

Example 10.2

$$c_1 = (\neg p_1 \vee p_2)$$
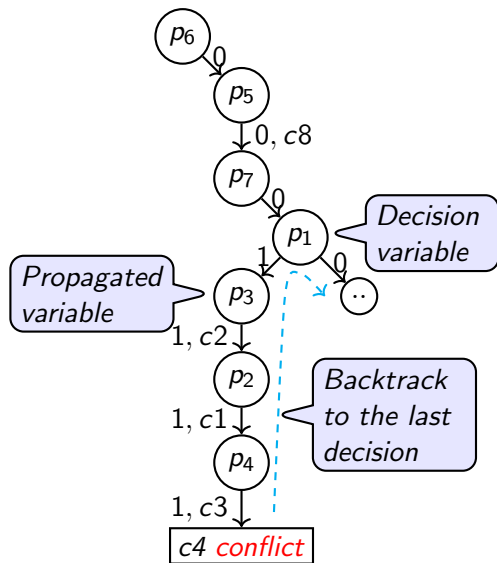$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



*Blue* : *causing unit propagation*
*Green*/*Blue* : *true clause*

Exercise 10.6 *Complete the DPLL run*

# Optimizations

DPLL allows many optimizations.

We will discuss many optimizations.

- **clause learning**
- 2-watched literals
- ...

First, let us look at a revolutionary optimization.

Topic 10.2

Clause learning

# Clause learning

As we decide and propagate,

we may construct a data structure to

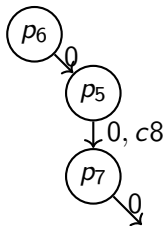observe the run and avoid unnecessary backtracking.

# Run of DPLL

## Definition 10.3
*We call the current partial model a run of DPLL.*

## Example 10.3
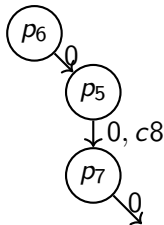*Borrowing from the earlier example, the following is a run that has not reached to the conflict yet.*

# Decision level

### Definition 10.4

*During a run, the decision level of a true literal is the number of decisions after which the literal was made true.*

### Example 10.4



*Given the run, we write $\neg p_5 @1$ to indicate that $\neg p_5$ was set to true after one decision.*

*Similarly, we write $\neg p_7 @2$ and $\neg p_6 @1$.*

# Implication graph

During the DPLL run, we maintain the following data structure.

## Definition 10.5
*Under a partial model $m$, the implication graph is a labeled DAG $(N, E)$, where*

- ▶ *$N$ is the set of true literals under $m$ and a conflict node*
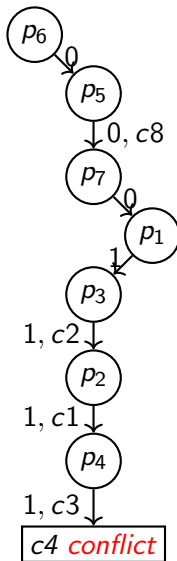- ▶ *$E = \{(\ell_1, \ell_2) | \neg\ell_1 \in causeClause(\ell_2) \text{ and } \ell_2 \neq \neg\ell_1\}$*

$$causeClause(\ell) \triangleq \begin{cases} \text{clause due to which unit propagation made } \ell \text{ true} \\ \emptyset \text{ for the literals of the decision variables} \end{cases}$$
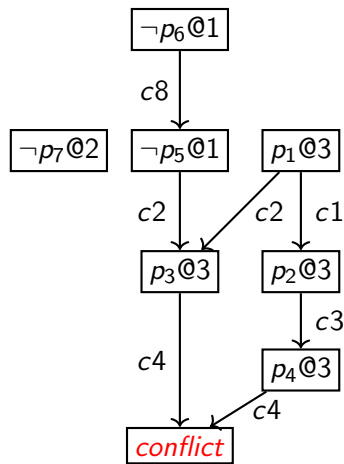
We also annotate each node with decision level.

# Example: implication graph

Example 10.5

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
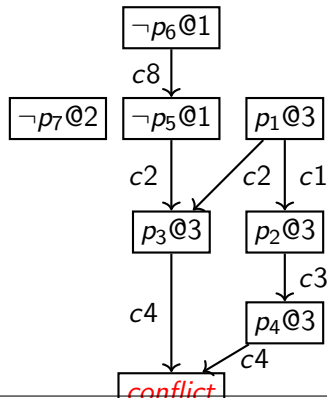$$c_8 = (p_6 \vee \neg p_5)$$



**Implication graph**

# Conflict clause

We traverse the implication graph backwards to find the set of decisions that caused the conflict.

## Definition 10.6
*The clause of the negations of the causing decisions is called conflict clause.*

## Example 10.6



Conflict clause : $p_6 \lor \neg p_1$

**Commentary:** In the above example, $p_6$ is set to 0 by the first decision. Therefore, literal $p_6$ is added in the conflict clause. Since the second decision does not contribute to the conflict, nothing is added in the conflict clause for the decision. Since $p_1$ is set to 1 by the third decision, literal $\neg p_1$ is added in the conflict clause. Not an immediately obvious idea. You may need to stare at the definition for sometime.

# Clause learning

Clause learning heuristics

▶ add conflict clause in the input clauses and
▶ backtrack to the second last conflicting decision, and proceed like DPLL

## Theorem 10.1
*Adding conflict clause*

1. *does not change the set of satisfying assignments*
2. *implies that the conflicting partial assignment will never be tried again*

Multiple clauses can satisfy the above two conditions.

## Definition 10.7 (Functional definition of conflict clause)
*We will say if a clause satisfies the above two conditions, it is a conflict clause.*

# Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver
3. Enables very many other heuristics without much complications. Covered in CS433.

## Example 10.7

*In the previous example, we made decisions : $m(p_6) = 0$, $m(p_7) = 0$, and $m(p_1) = 1$*

*We learned a conflict clause : $p_6 \vee \neg p_1$* ⟨ There are other clever choices for conflict clauses. ⟩

*Adding this clause to the input clauses results in*

1. *$m(p_6) = 0$, $m(p_7) = 1$, and $m(p_1) = 1$ will never be tried*
2. *$m(p_6) = 0$ and $m(p_1) = 1$ will never occur simultaneously.*

Topic 10.3

CDCL(conflict driven clause learning)

# DPLL to CDCL

Impact of clause learning was profound.

Some call the optimized algorithm CDCL(conflict driven clause learning) instead of DPLL.

## CDCL as an algorithm

**Algorithm 10.3:** CDCL

**Input:** CNF $F$

$m := \emptyset;\ dl := 0;\ dstack := \lambda x.0;$   [dl stands for decision level]

$m := \text{UNITPROPAGATION}(m, F);$

**do**

    // backtracking

    **while** $F$ *is false under* $m$ **do**

        **if** $dl = 0$ **then return** *unsat*;

        $(C, dl) := \text{ANALYZECONFLICT}(m, F);$

        $m.resize(dstack(dl));\ F := F \cup \{C\};$

        $m := \text{UNITPROPAGATION}(m, F);$

    // Boolean decision

    **if** $F$ *is unassigned under* $m$ **then**   [dstack records history of backtracking]

        $dstack(dl) := m.size();$

        $dl := dl + 1;\ m := \text{DECIDE}(m, F);$

        $m := \text{UNITPROPAGATION}(m, F);$

**while** $F$ *is unassigned or false under* $m$;

**return** *sat*

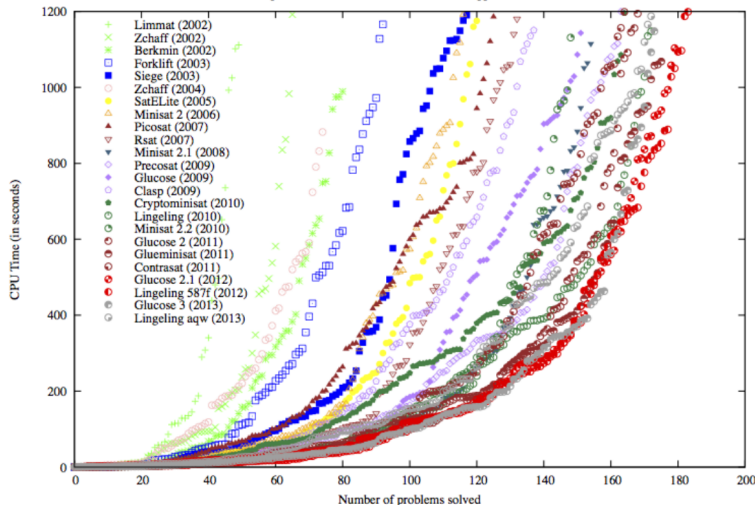▶ $\text{UNITPROPAGATION}(m, F)$ - applies unit propagation and extends $m$

▶ $\text{ANALYZECONFLICT}(m, F)$ - returns a conflict clause learned using implication graph and a decision level upto which the solver needs to backtrack

▶ $\text{DECIDE}(m, F)$ - chooses an unassigned variable in $m$ and assigns a Boolean value

# Efficiency of SAT solvers over the years



Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

Legend:
- Limmat (2002)
- Zchaff (2002)
- Berkmin (2002)
- Forklift (2003)
- Siege (2003)
- Zchaff (2004)
- SatELite (2005)
- Minisat 2 (2006)
- Picosat (2007)
- Rsat (2007)
- Minisat 2.1 (2008)
- Precosat (2009)
- Glucose (2009)
- Clasp (2009)
- Cryptominisat (2010)
- Lingeling (2010)
- Minisat 2.2 (2010)
- Glucose 2 (2011)
- Glueminisat (2011)
- Contrasat (2011)
- Glucose 2.1 (2012)
- Lingeling 587f (2012)
- Glucose 3 (2013)
- Lingeling aqw (2013)

CPU Time (in seconds) vs Number of problems solved

Cactus plot:
Y-axis: time out
X-axis: Number of problems solved
Color: a competing solver

### Exercise 10.7
a. *What is the negative impact of SAT competition?*
b. *What are look-ahead based SAT solvers?*

Source: http://satsmt2014.forsyte.at/files/2014/07/SAT-introduction.pdf

# Impact of SAT technology

Impact is enormous.

Probably, the greatest achievement of the first decade of this century in science after sequencing of human genome

A few are listed here

- ▶ Hardware verification and design assistance
  Almost all hardware/EDA companies have their own SAT solver
- ▶ Planning: many resource allocation problems are convertible to SAT
- ▶ Security: analysis of crypto algorithms
- ▶ Solving hard problems, *e. g.,* travelling salesman problem

Topic 10.4

Problems

# Exercise : run CDCL

Exercise 10.8
*Give a run of CDCL to completion on the CNF formula in example 10.2*

# Exercise: CDCL termination

### Exercise 10.9
*Prove that CDCL always terminates.*

# DPLL to Resolution*

## Exercise 10.10
*Let us suppose we run DPLL on an unsatisfiable formula. Give a linear time algorithm in terms of the number of steps in the run to generate resolution proof of unsatisfiability from the run of DPLL.*

# Lo`vasz local lemma vs. SAT solvers

Here, we assume a $k$-CNF formula has clauses with exactly $k$ literals.

## Theorem 10.2 (Lo`vasz local lemma)
*If each variable in a $k$-CNF formula $\phi$ occurs less than $2^{k-2}/k$ times, $\phi$ is sat.*

## Definition 10.8
*A Lo`vasz formula is a $k$-CNF formula that has all variables occurring $\frac{2^{k-2}}{k} - 1$ times, and for each variable $p$, $p$ and $\neg p$ occur nearly equal number of times.*
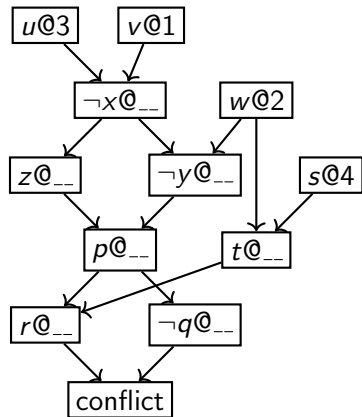
## Exercise 10.11
▶ *Write a program that generates uniformly random Lo`vasz formula*
▶ *Generate 10 instances for $k = 3, 4, 5, \dots$.*
▶ *Solve the instances using some sat solver*
▶ *Report a plot $k$ vs. average run times*

# Conflict clauses

## Exercise 10.12

*Consider the following implication graph generated in a CDCL solver.*



a. Assign decision level to every node (write within the node)

b. What is the conflict clause due to the implication graph?

# End of Lecture 10