CS228 Logic for Computer Science 2021

Lecture 11: SAT Solvers

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2021-01-28

Propositional satisfiability problem

Consider a propositional logic formula F.

Find a model m such that

$$m \models F.$$

Example 11.1

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

Some terminology

- Propositional variables are also referred as atoms
- A literal is either an atom or its negation
- A clause is a disjunction of literals.

Since $\vee$ is associative, commutative, and absorbs multiple occurrences, a clause may be referred as a set of literals

Example 11.2

- *p is an atom but $\neg p$ is not.*
- *$\neg p$ and p both are literals.*
- *$p \vee \neg p \vee p \vee q$ is a clause.*
- *$\{p, \neg p, q\}$ is the same clause.*

Conjunctive normal form(CNF)

Definition 11.1

A formula is in CNF if it is a conjunction of clauses.

Since $\wedge$ is associative, commutative, and absorbs multiple occurrences, a CNF formula may be referred as a set of clauses

Example 11.3

- $\neg p$ *and* p *both are in CNF.*
- $(p \vee \neg q) \wedge (r \vee \neg q) \wedge \neg r$ *in CNF.*
- $\{(p \vee \neg q), (r \vee \neg q), \neg r\}$ *is the same CNF formula.*
- $\{\{p, \neg q\}, \{r, \neg q\}, \{\neg r\}\}$ *is the same CNF formula.*

Exercise 11.1

Write a formal grammar for CNF

CNF input

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

Tseitin encoding can convert each formula into a CNF without any blowup.

- introduces fresh variables

Topic 11.1

DPLL (Davis-Putnam-Loveland-Logemann) method

Notation: partial model

Definition 11.2

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

Notation: state of a literal

Under partial model m,

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

Otherwise, ℓ is unassigned.

Exercise 11.2

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 ℓ is true and
C is false if for each $\ell \in C$, ℓ is false.

Otherwise, C is unassigned.

Exercise 11.3

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

What are the states of the following clauses under m?

- $p_1 \lor p_2 \lor p_3$ T
- $p_1 \lor \neg p_2$ F
- $p_1 \lor p_3$ Unc
- $\emptyset$ *(empty clause)* F

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 11.4

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

What are the states of the following formulas under m?

- $(p_3 \lor \neg p_1) \land (p_1 \lor \neg p_2)$ F
- $(p_1 \lor p_2 \lor p_3) \land \neg p_1$
- $p_1 \lor p_3$ Una
- $\emptyset$ *(empty formula)* T

Notation: unit clause and unit literal

Definition 11.3

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

Exercise 11.5

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.

- $p_1 \lor \underline{\neg p_3} \lor \neg p_2$ ✓
- $p_1 \lor \neg p_3 \lor p_2$ ✗
- $p_1 \lor \neg p_3 \lor p_4$ ✗
- $p_1 \lor \neg p_2$ ✗

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 11.1: DPLL(F)

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

Algorithm 11.2: 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* ;
if $\exists$ *unit literal* p *under* m **then**
  return $DPLL(F, m[p \mapsto 1])$
if $\exists$ *unit literal* $\neg p$ *under* m **then**
  return $DPLL(F, m[p \mapsto 0])$
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 $DPLL(F, m[p \mapsto 1 - b])$

Backtracking at conflict

Unit propagation

Decision

Three actions of DPLL

A DPLL run consists of three types of actions

- Decision
- Unit propagation
- Backtracking

Exercise 11.6

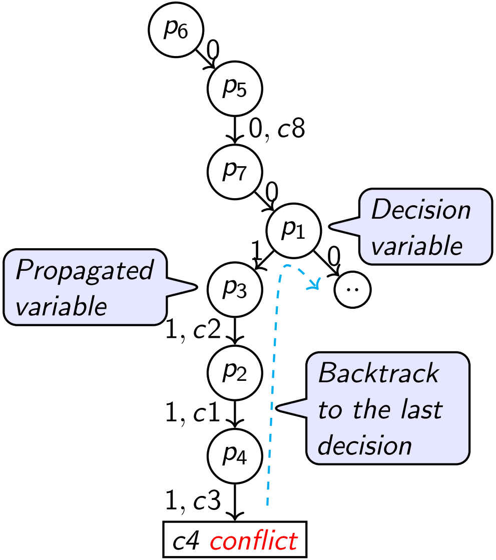
What is the worst case complexity of DPLL?

Example: decide, propagate, and backtrack in DPLL

Example 11.4

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

Blue : causing unit propagation
Green/Blue : true clause



Exercise 11.7 *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 11.2

Clause learning

Clause learning

As we decide and propagate,

we may construct a data structure to

observe the run and avoid unnecessary backtracking.

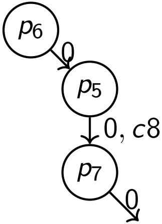
Run of DPLL

Definition 11.4

We call the current partial model a run of DPLL.

Example 11.5

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

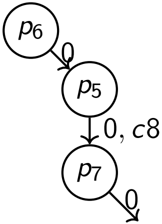


Decision level

Definition 11.5

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

Example 11.6



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 11.6

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*
- $E = \{(\ell_1, \ell_2) | \neg\ell_1 \in causeClause(\ell_2) \text{ and } \ell_2 \neq \neg\ell_1\}$

$$causeClause(\ell) \triangleq \begin{cases} \text{clause due to which unit propagation made } \ell \text{ true} \\ \emptyset \text{ for the literals of the decision variables} \end{cases}$$

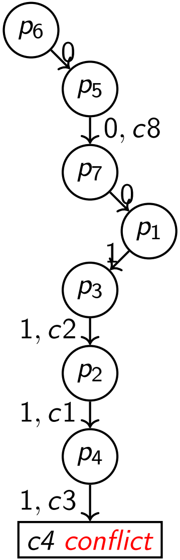
We also annotate each node with decision level.

Commentary: DAG = directed acyclic graph. *conflict* node denotes contradiction in the run. *causeClause* definition works with the conflict node.(why?)

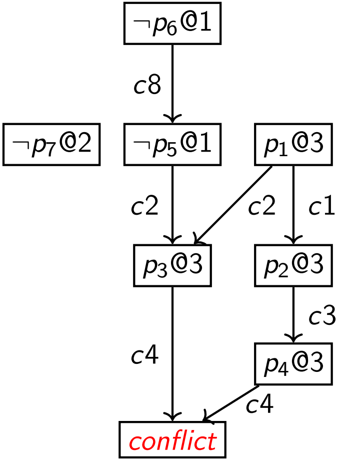
Example: implication graph

Example 11.7

$$
\begin{aligned}
c_1 &= (\neg p_1 \vee p_2)\\
c_2 &= (\neg p_1 \vee p_3 \vee p_5)\\
c_3 &= (\neg p_2 \vee p_4)\\
c_4 &= (\neg p_3 \vee \neg p_4)\\
c_5 &= (p_1 \vee p_5 \vee \neg p_2)\\
c_6 &= (p_2 \vee p_3)\\
c_7 &= (p_2 \vee \neg p_3 \vee p_7)\\
c_8 &= (p_6 \vee \neg p_5)
\end{aligned}
$$



Implication graph



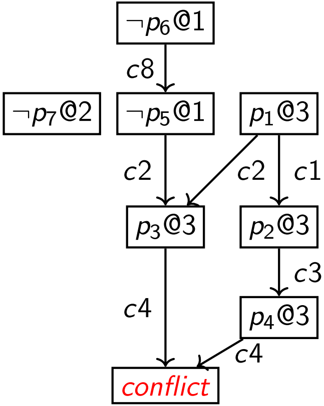
Conflict clause

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

Definition 11.7

The clause of the negations of the causing decisions is called conflict clause.

Example 11.8



Conflict clause : $p_6 \vee \neg p_1$

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. 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 11.1

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 11.8 (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. We will see them shortly.

Example 11.9

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 11.3

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.

Commentary: The name change is not a gimmick. All SAT solving algorithms find a satisfying assignment by attempting assignments in some sequence. An algorithm is characterized by the order of attempts. Due to the conflict clauses, CDCL exhibits very distinct probability distribution of the order of attempts from DPLL. Therefore, the change in name is warranted. Furthermore, we have very limited theoretical understanding of the success of CDCL.

CDCL as an algorithm

Algorithm 11.3: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

dl stands for decision level

$m :=$ UnitPropagation(m, F);

do

// backtracking

while $m \not\models F$ **do**

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

  $(C, dl) :=$ AnalyzeConflict(m, F);

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

  m := UnitPropagation(m, F);

// Boolean decision

if F *is unassigned under* m **then**

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

dstack records history of backtracking

  $dl := dl + 1$; $m :=$ Decide(m, F);

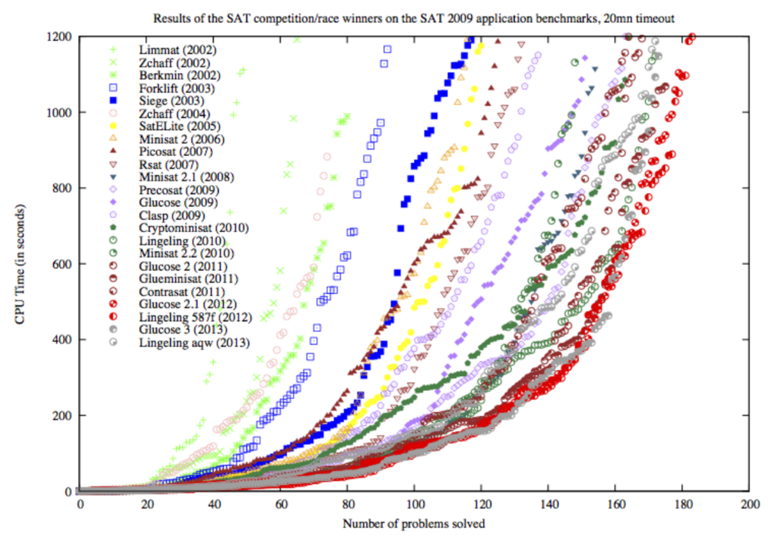
  m := UnitPropagation(m, F) ;

while F *is unassigned under* m *or* $m \not\models F$;

return *sat*

- UnitPropagation(m, F) - applies unit propagation and extends m
- AnalyzeConflict(m, F) - returns a conflict clause learned using implication graph and a decision level upto which the solver needs to backtrack
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value

Efficiency of SAT solvers over the years



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

Cactus plot:

Y-axis: time out

X-axis: Number of problems solved

Color: a competing solver

Exercise 11.8

a. What is the negative impact of SAT competition?

b. What are look-ahead based SAT solvers?

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 11.4

Problems

Exercise : run CDCL

Exercise 11.9

Give a run of CDCL to completion on the CNF formula in example 11.4

Exercise: CDCL termination

Exercise 11.10

Prove that CDCL always terminates.

Lovàsz local lemma vs. SAT solvers

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

Theorem 11.2 (Lovàsz local lemma)

If each variable in a k-CNF formula ϕ occurs less than $2^{k-2}/k$ times, ϕ is sat.

Definition 11.9

A Lovàsz 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 11.11

- *Write a program that generates uniformly random Lovàsz formula*
- *Generate 10 instances for $k = 3, 4, 5, ...$*
- *Solve the instances using some sat solver*
- *Report a plot k vs. average run times*

Commentary: There are many sat solvers available online. Look into the following webpage of sat competition to find a usable and downloadable tool. `http://www.satcompetition.org`. Please discuss with the instructor if there is any confusion.

End of Lecture 11