CS228 Logic for Computer Science 2020

Lecture 11: SAT solvers

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-02-05

Propositional satisfiability problem

Consider a propositional logic formula F.

Find an assignment m such that

$$m \models F.$$

Example 11.1

Give an assignment of $p_1 \land (\neg p_2 \lor p_3)$

Topic 11.1

DPLL (Davis-Putnam-Loveland-Logemann) method

Notation: partial assignment

Definition 11.1

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

Notation: state of a literal

Under partial assignment m,

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

Otherwise, ℓ is unassigned.

Exercise 11.1

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

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 assignment 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 11.2

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

What are the states of the following clauses under m?

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

Notation: state of a formula

Under partial assignment 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 11.3

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

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 11.2

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 11.4

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

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

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

DPLL (Davis-Putnam-Loveland-Logemann) method

DPLL

- takes CNF input
- maintains a partial assignment, 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*

Backtracking at conflict

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 $DPLL(F, m[p \mapsto 0])$

Unit propagation

Decision

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 11.5

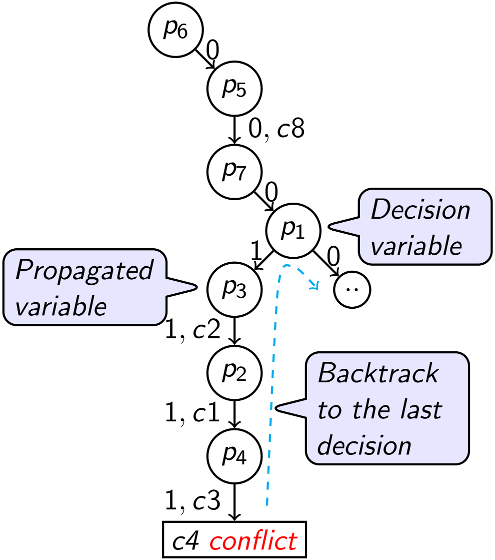
What is the worst case complexity of DPLL?

Example: decide, propagate, and backtrack in DPLL

Example 11.2

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

Blue : causing unit propagation
Green/Blue : true clause



Exercise 11.6

Complete the DPLL run

Optimizations

DPLL allows many optimizations.

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

We will only 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.

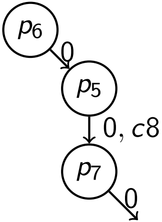
Run of DPLL

Definition 11.3

We call the current partial assignment a run of DPLL.

Example 11.3

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

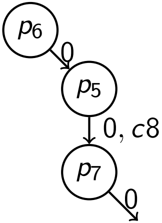


Decision level

Definition 11.4

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

Example 11.4



Given the above run, we write $\neg p_5@1$ to indicate that $\neg p_5$ was set to true after one decision. Similarly, we write $\neg p_7@2$.

Implication graph

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

Definition 11.5

Under a partial assignment m, the implication graph is a labeled DAG (N, E), where

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

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

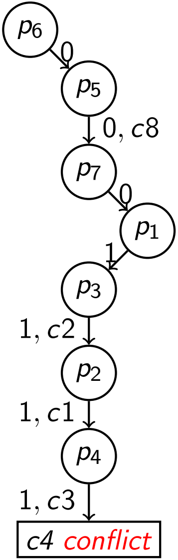
We also annotate each node with decision level.

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

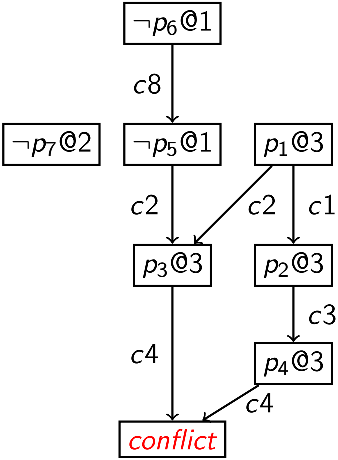
Example: implication graph

Example 11.5

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



Implication graph



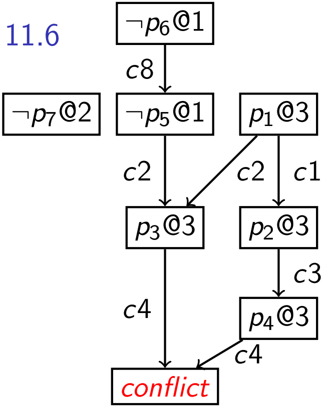
Conflict clause

In the case of conflict, we traverse the implication graph backwards to find the set of decisions that caused the conflict.

Definition 11.6

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

Example 11.6



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*

Exercise 11.7

Prove the above theorem

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 11.7

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 : $p_6 \vee \neg p_1$

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.*

Impact of clause learning was so profound that some people call the optimized algorithm CDCL(conflict driven clause learning) instead of DPLL

CDCL as an algorithm

Algorithm 11.3: CDCL

```
Input: CNF F
m := ∅; dl := 0; dstack := λx.0;
UnitPropagation(m, F);
do
    // backtracking
    while m ⊭ F do
        if dl = 0 then return unsat;
        (C, dl) := AnalyzeConflict(m, F);
        m.resize(dstack(dl)); F := F ∪ {C}; m := UnitPropagation(m, F);
    // Boolean decision
    if m is partial then
        dstack(dl) := m.size();
        dl := dl + 1; Decide(m, F); UnitPropagation(m, F) ;
while m is partial or m ⊭ F;
return sat
```

dl stands for decision level

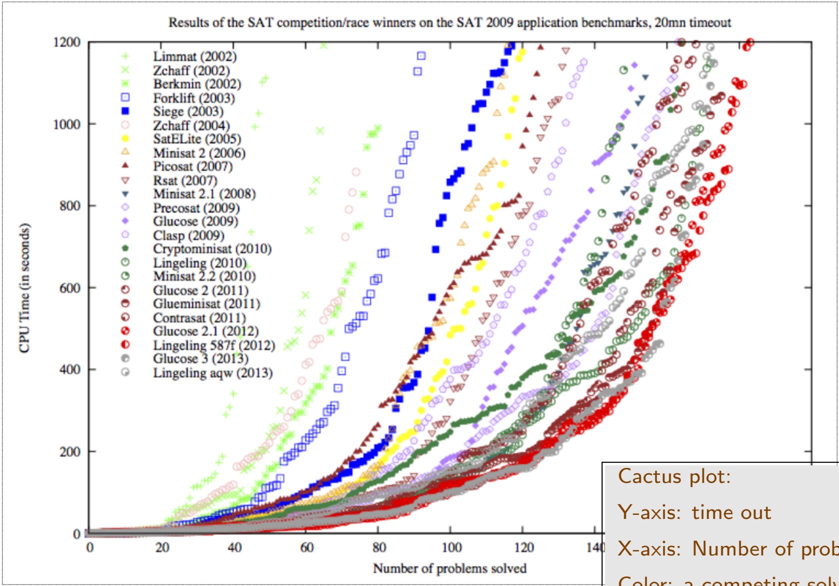
dstack records history for backtracking

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

Topic 11.3

SAT technology and its impact

Efficiency of SAT solvers over the years



Cactus plot:

Y-axis: time out

X-axis: Number of problems solved

Color: a competing solver

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 11.4

Problems

Run DPLL

Exercise 11.8

Run DPLL on the following 2-SAT problem

$$\underbrace{(p_1 \lor p_2)}_{c1} \land \underbrace{(p_3 \lor p_4)}_{c2} \land \underbrace{(p_5 \lor p_6)}_{c3} \land \underbrace{(\neg p_1 \lor \neg p_3)}_{c4} \land \underbrace{(\neg p_1 \lor \neg p_5)}_{c5} \land \underbrace{(\neg p_3 \lor \neg p_5)}_{c6} \land$$

Use the variable ordering $p_1, p_2, p_3, p_4, p_5, p_6$. Always use 0 as first choice for a decision. Annotate a unit propagation with clause number. Also annotate the conflict(s) with the clause number.

End of Lecture 11