COL750: Foundations of Automatic Verification (Jan-May 2023)

Lectures 15 & 16 (Propositional SAT Solving)

Kumar Madhukar

madhukar@cse.iitd.ac.in

Mar 13th and 16th

Propositional Satisfiability

- consider the formula:
  $(\neg A \vee B) \wedge (\neg B \vee \neg C) \wedge (C \vee \neg D)$

- we can also write this equivalently, in the set notation, as follows:
  $\{\{\neg A, B\}, \{\neg B, \neg C\}, \{C, \neg D\}\}$

- is this formula satisfiable? how do we know?

- explore all the 2^4 assignments possible to the propositional variables A, B, C, and D (every variable can take one of the two values – *true* or *false*)

- we can do this systematically as a depth-first search (can we do better?)

Proof Calculus for Propositional Logic

- consists of rules of inference which can be used to derive a series of conclusions from a series of hypotheses, in a mechanical way
- the resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals
- a literal is a propositional variable or the negation of a propositional variable; two literals are said to be complements if one is the negation of the other.
- along with a complete search algorithm, resolution yields a sound and complete algorithm for deciding the satisfiability of a propositional formula

Propositional Resolution

- in this lecture, we use the set representation of CNF formulas
- from the clauses $\{p_1, p_2, \neg p_4\}$ and $\{\neg p_2, p_3\}$, resolution lets us derive $\{p_1, p_3, \neg p_4\}$ (called the resolvent)
- adding the resolvent to the original set of formulas gives us an equivalent formula (why?)
- we will restrict ourselves to resolution where one of the participating clauses is a unit clause (also called unit-resolution), and where the resolvent is a unit literal

$(I, \Gamma) = \textit{unit-resolution}(\Delta)$

Δ – the original set of clauses (in the formula)

I – set of literals that were either present as unit clauses in Δ or derived by unit-resolution

Γ – the new set of clauses resulting from conditioning Δ on I

- example:
  $\Delta = \{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, D\}, \{A\}\}$
  $I = \{A, \neg B, C, D\}$
  $\Gamma = \{\}$

- another example:
  $\Delta = \{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, D\}, \{C\}\}$
  $I = \{C, D\}$
  $\Gamma = \{\{\neg A, \neg B\}\}$

Revisiting our first example

$\{\{\neg A, B\}, \{\neg B, \neg C\}, \{C, \neg D\}\}$

- suppose we set A to *true* and then, before setting any other variable, we perform unit-resolution(Δ), where Δ is:
  $\{\{\neg A, B\}, \{\neg B, \neg C\}, \{C, \neg D\}, \{A\}\}$
- we immediately get $I = \{A, B, \neg C, \neg D\}$, and an empty formula by conditioning Δ on I
- unlike earlier, we do not need to explore the entire tree of valuations in the DFS
- this gives us our first algorithm for propositional SAT: DPLL[1] (= DFS + unit-resolution)

[1] Davis - Putnam - Logemann - Loveland (DPLL), in the name of Martin Davis, Hilary Putnam, George Logemann and Donald W. Loveland, who introduced this in 1961

DPLL(Δ)

```
$(I, \Gamma) = \textit{unit-resolution}(\Delta)$

if $\Gamma = \{\}$, return $I$
if $\{\} \in \Gamma$, return fail

choose a literal $l \in \Gamma$

if L $=$ DPLL($\Gamma \cup \{\{l\}\}$) $\neq$ fail, return $I \cup L$
if L $=$ DPLL($\Gamma \cup \{\{\neg l\}\}$) $\neq$ fail, return $I \cup L$

return fail
```

Example

1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$

Example

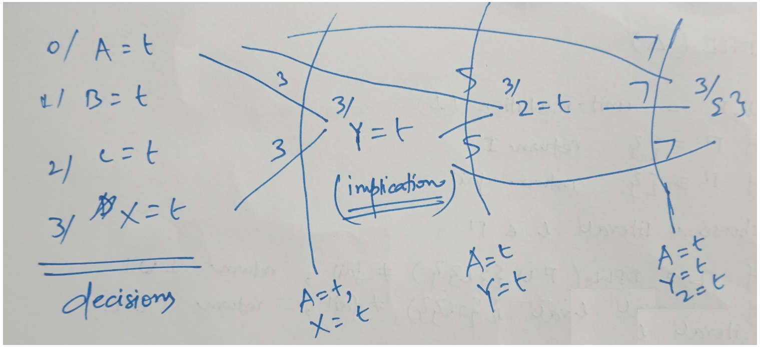
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$

- suppose we decide that A be set to *true* (let us call this decision-level 0)
- unit resolution after adding $\{A\}$ as the 8th clause does not help!
- suppose we then set B to *true* (decision-level 1), C to *true* (decision-level 2), and then X to *true* (decision-level 3)
- it is now that unit-resolution can give us something; in this case it gives us a conflict (an empty clause)
- the algorithm then backtracks, and tries the other branch for X, and then for C and then for B (i.e., it backtracks chronologically) but ends up in a conflict in every case until it tries the other branch for A

Non-chronological backtracking

- in the last example, the decisions to make B and C true have no bearing on the conflict; the problem was that A and X cannot both be true (but how do we identify this?)
- if we could add the clause $\{\neg A, \neg X\}$ in the set (easy to do, once we have identified that A and X cannot both be true), we would have got, immediately after adding $\{A\}$, $\{negX\}$ from unit-resolution, and the clauses $\{X, Z\}$ and $\{X, \neg Z\}$ in Γ
- resolving further would give a conflict, revealing that A cannot be set true (and telling us that the backtracking should jump to the level where A was decided true, and change that decision – non-chronological backtracking)

Implication graph and clause learning

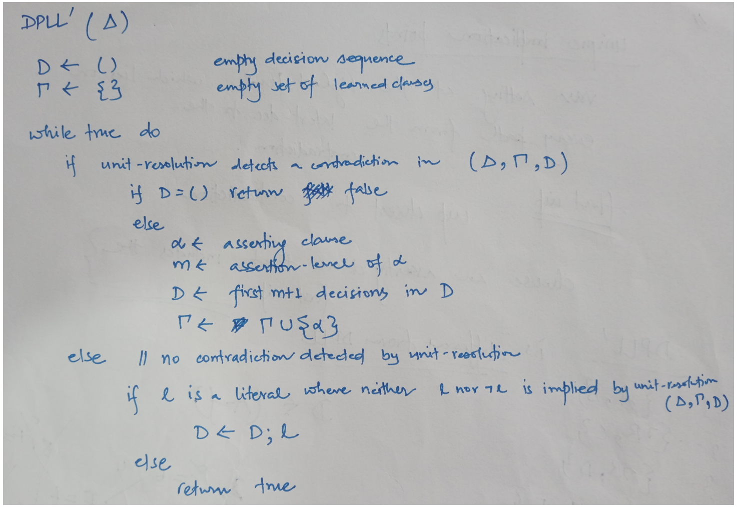


the cuts (separating the conflict from the decisions) in the implication graph give us the conflict sets – e.g. $\{A, X\}$, $\{A, Y\}$, and $\{A, Y, Z\}$ above – which give us the learned clauses: $(\neg A \vee \neg X)$, $(\neg A \vee \neg Y)$, and $(\neg A \vee \neg Y \vee \neg Z)$

Asserting clause and assertion level

- an asserting clause is one that includes only one variable at the highest decision level
- of the three clauses that could be learned on the last slide – $(\neg A \vee \neg X)$, $(\neg A \vee \neg Y)$, and $(\neg A \vee \neg Y \vee \neg Z)$ – only the first two are asserting
- the third clause includes two variables at last decision-level (3)
- assertion-level of a clause is the second highest decision level in that clause
- the improved algo: add an asserting clause, and backtrack to its assertion-level

DPLL'



```
DPLL' ($\Delta$)

    $D \leftarrow ()$          empty decision sequence
    $\Gamma \leftarrow \{\}$          empty set of learned clauses

while true do
    if unit-resolution detects a contradiction in $(\Delta, \Gamma, D)$
        if $D = ()$ return false
        else
            $\alpha \leftarrow$ asserting clause
            $m \leftarrow$ assertion-level of $\alpha$
            $D \leftarrow$ first $m+1$ decisions in $D$
            $\Gamma \leftarrow \Gamma \cup \{\alpha\}$
    else    // no contradiction detected by unit-resolution
        if $\ell$ is a literal where neither $\ell$ nor $\neg\ell$ is implied by unit-resolution $(\Delta, \Gamma, D)$
            $D \leftarrow D; \ell$
        else
            return true
```

this algorithm is generally referred to as CDCL (Conflict-Driven Clause Learning)

Revisiting the example

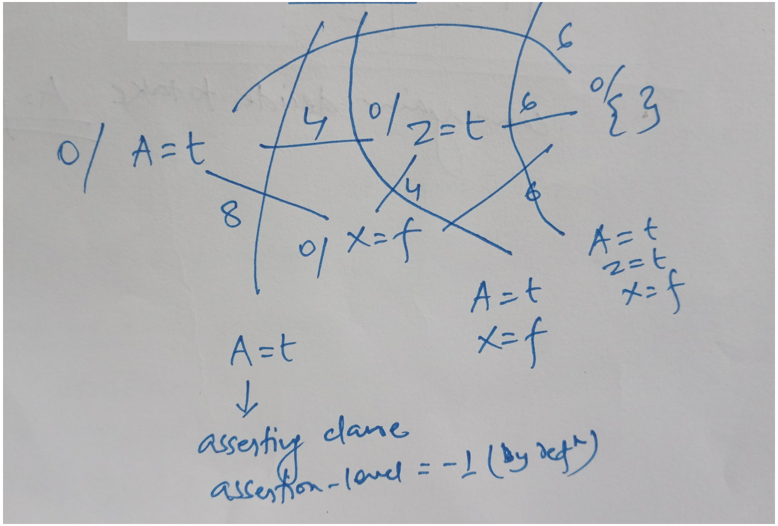
Let us see how CDCL works on our earlier example.

We start with the empty sequence of decisions, and an empty set of learned clauses. Then we make the decisions $(A = \mathit{true}, B = \mathit{true}, C = \mathit{true}, X = \mathit{true})$ (at decision levels 0, 1, 2, and 3, resp.).

We have seen that this leads to a conflict, and that $(\neg A \vee \neg X)$ is an asserting clause.
So, we set:
$\alpha = \{\neg A, \neg X\}$
$m = 0$
$D = (A = \mathit{true})$
$\Gamma = \{\{\neg A, \neg X\}\}$

Revisiting the example

This leads to another conflict, shows below, and the algorithm continues..

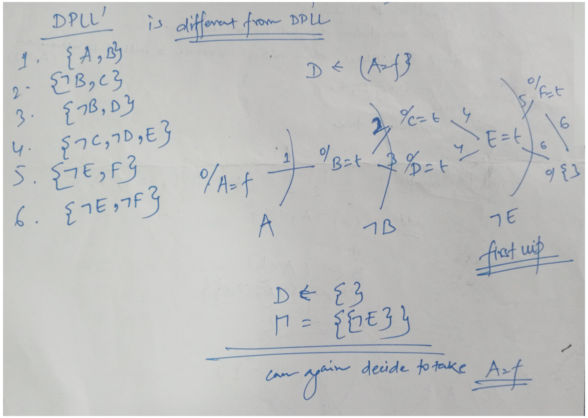


Unique Implication Points (UIP)

- uip's are variable assignments (through decisions or implications) at the highest decision-level which lie on every path from the latest decision to the contradiction
- the latest decision is trivially a uip
- first-uip: uip closest of the contradiction
- while choosing an asserting clause to add, we should choose one that includes the first-uip

Difference between DPLL' and DPLL

Note that we gave a new name – CDCL – to the improved algo DPLL'. We did that because the improvement makes a substantial change to the algorithm. In particular, DPLL' can revisit a decision that had led to conflict earlier (but only after learning new clauses). This example illustrates this:



Thank you!