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

Lectures 15 & 16 (Propositional SAT Solving)

Kumar Madhukar

madhukar@cse.iitd.ac.in

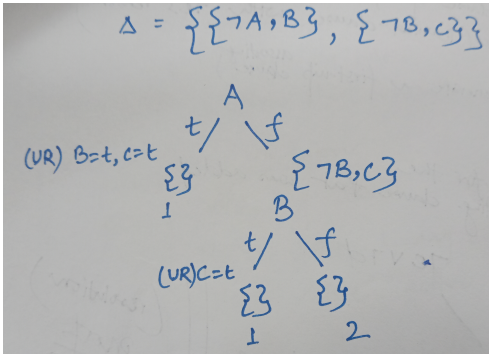
Mar 13th and 16th

Exhaustive DPLL

Consider the formula:
$\{\{\neg A, B\}, \{\neg B, C\}\}$

- to check for satisfiability, let us say we start with the decision of making A true
- immediately, unit-resolution gives a solution that B and C must also be true
- at this point, we must go on and explore other paths if we have to find all satisfying assignments (instead of just one)

Example



- the count of 2 along the rightmost branch just indicates that we are counting over 3 variables – A, B, and C – and only two have been decided along that branch (leaving both options – true and false – open for C)

C(ounting-)DPLL(Δ, n)

if $\Delta = \{\}$, return 2^n

if $\{\} \in \Delta$, return 0

choose $L \in \Delta$

$(I^+, \Gamma^+) = \mathit{unit} - \mathit{resolution}(\Delta \cup \{\{L\}\})$
$(I^-, \Gamma^-) = \mathit{unit} - \mathit{resolution}(\Delta \cup \{\{\neg L\}\})$

return CDPLL($\Gamma^+, n - |I^+|$) + CDPLL($\Gamma^-, n - |I^-|$)

Learning an asserting clause

learned-clause = clause that became empty during unit-resolution

while (learned-clause is not asserting)

L $\leftarrow$ literal in learned-clause falsified last
learned-clause $\leftarrow$ resolve (learned-clause, $\neg$L's reason)

Is this guaranteed to generate a first-uip asserting clause? Why?

Example

Consider the following clauses:

1. $\{\neg a, \neg b, c\}$
2. $\{\neg b, \neg c, d\}$
3. $\{\neg c, \neg d\}$
4. $\{\neg x, y\}$
5. $\{\neg x, \neg y, z\}$
6. $\{\neg y, \neg z\}$
7. $\{x, y\}$
8. $\{x, z\}$
9. $\{\neg y, w\}$
10. $\{\neg z, \neg w\}$

- decide $a = \mathit{true}$ at level 0 (does not help with any unit-propagation)
- decide $b = \mathit{true}$ at level 1
- the decisions now imply $c = \mathit{true}$ (at level 1, reason: $\{\neg a, \neg b, c\}$), and also $d = \mathit{false}$ (at level 1, reason: $\{\neg c, \neg d\}$)
- contradiction ($\{\neg b, \neg c, d\}$ becomes empty)

Example: Asserting clause

How do we find an asserting clause from the contradiction in the last example (using the algorithm given just before the example)?

1. learned-clause $= \{\neg b, \neg c, d\}$ *(to begin with)*
2. learned-clause $= \{\neg b, \neg c\} =$ resolve $(\{\neg b, \neg c, d\}, \{\neg c, \neg d\})$
3. learned-clause $= \{\neg a, \neg b\} =$ resolve $(\{\neg b, \neg c\}, \{\neg a, \neg b, c\})$

This method also gives us a way to certify the unsat answer if we get one. The certificate is essentially this resolution proof that derives any clause that gets added from the clauses that we already have.

Implementing a sat solver

- Variable ordering (VSIDS)
- Phase selection
- 2 watched literals
- Pure literal elimination
- Restarts
- Learned clause deletion

Variable ordering (VSIDS)

- the basic idea is to select a literal that has the maximum number of occurrences in unsatisfied clauses
- vsids improves upon this
- initial score is calculated as above
- but the scores are updated with new clauses being learned, and at regular intervals, all the scores are divided by a constant (to give a higher priority to the literals that are "currently" causing the conflict)

Phase selection

- many solvers try to try the same phase for a variable that was tried earlier

2 watched literals

- to avoid looking at all the clauses all the time
- identify two literals in every clause as watched, and do not revisit the clause until one of the watched literals has been assigned
- if the watched literals are non-false, the clause is not unit
- if any of the watched literals becomes false, we find a new pair
- if we cannot find a pair, the clause is a unit clause
- backtracking does not cause any extra work

Pure literal elimination

- a literal is pure if it appears in only one form (positive or negative) in the entire formula
- pure literals can simply be assigned without compromising satisfiability
- often not implemented – at odds with the 2 watched literals scheme because it requires looking at every clause after every assignment (decision or implication)

Restarts

- restarts (with a different variable ordering) are used to prevent the solvers from getting stuck in an unyielding parts of the solution-space
- how frequently should this be done – often short restarts, but with significant chances of long restarts
- what about learned clauses? useful to keep (many of) them across restarts

Learned clause deletion

- delete long clauses with higher probability
- not a good idea to delete binary clauses or active clauses (the ones that have participated in unit-resolution)
- when to delete? at restarts? or when the number of learned clauses exceeds some threshold
- a good measure is literal block distance (number of decision levels appearing in a learned clause); larger this distance, more chances of deletion

Other approaches for SAT

SAT using local search

- start with a random assignment
- if the current assignment does not work, flip one of the variables (or, go to a neighboring assignment that leads to fewer unsatisfied clauses)
- space efficient but incomplete
- restart (or move to a worse-off neighbor with a small probability) if none of the neighboring assignments are better (alternatively, pick an unsatisfied clause and move to a neighbor that satisfies the chosen clause)

Thank you!