CS202A: Mathematics For Computer Science-II

Assignment 2 Report

Aditya Tanwar
200057

Akhil Agrawal
200076

February 14, 2022

SAT Solver

Statement: Implement a SAT solver. Given a formula in the DIMACS representation, your implementation should return:

i. A model if the formula is satisfiable

ii. Report that the formula is unsatisfiable

You are free to use any language and algorithm (Semantic Tableau, Analytic Tableau, DPLL or a combination).

Overview: We have used the *DPLL algorithm* to check for *SAT* for a formula given in *CNF* form. We begin by simplifying the formula in two steps, firstly by removing tautologies and then by using concepts *(specifically the concepts of "unit propagation" and "pure literal elimination")* derived from the *DPLL algorithm*. Finally, we use use some branching heuristics to move ahead as dictated in standard *DPLL algorithm* to check for *SAT*.

Notations & Conventions :

- ⋆ Tautologies: We restrict our definition of tautologies to mean only clauses which are always true and do not analyse formulae more "complex" than a single clause. Hence, we classify a clause as a *tautology* iff it includes both a literal and its negation (i.e., it contains both p and $\neg p$).
- ⋆ Unit literals: We use the same meaning as that in the context of the *DPLL algorithm* but for the sake of completeness, we elaborate it here as well. If any *"clause"* has a single literal residing in it *(before or after "updation")*, then we call that literal a *"unit literal"*, and "fix" its (boolean) value according to its appearance in the same clause (i.e., p is assigned T if p appears in the clause, and assigned F if $\neg p$ appears in the clause), in order to satisfy that clause.
- ⋆ Pure literals: We use the same meaning as that in the context of the *DPLL algorithm* but for the sake of completeness, we elaborate it here as well. If any literal appears in the same form in each clause (for example, $\neg p$ does not appear in any clause, while p does in some clause(s)), then its value can be "fixed" to satisfy those clauses without affecting any others.
- ⋆ Fixed literals: We use this set to collectively refer to set of "unit literals" ∪ "pure literals". We also store the *"fixed values"* of these literals in this set. We denote it by $\mathscr{F}$.
- ⋆ Updation: For each literal $p \in \mathscr{F}$, let p_v denote the value of p in $\mathscr{F}$. The updation corresponding to a "unit literal" will be the same as "pure literal" which will be elaborated later, as will be the the choice of their chronology. We update the present state of clauses in the *CNF*, in the following way-
  - ∗ If neither p_v nor $\neg p_v$ is present in the clause, we leave it untouched.
  - ∗ If p_v is present in the clause, we delete the clause altogether from the *CNF*, because it is true due to the *"fixed value"* and as a result, would not affect the remaining *CNF*.
  - ∗ If $\neg p_v$ is present in the clause and p_v is not, then we delete the occurrence of $\neg p_v$ from the clause because it will not affect the state of the clause as it is F anyway.
- ⋆ Empty Clause: In case, the updation results in the deletion of all literals present in a clause, then that implies the clause had all literals (or their negations) assigned the value F, and hence it could not be satisfied, and since, we are operating in the *CNF* here, any empty clause leads to the whole formula being unsatisfiable.

Results used :

- ⋆ From *DPLL*, unit literals simply lead to simplifications and neither fabricate false solutions nor reject any true solutions.
- ⋆ From *DPLL*, pure literals do not contribute to any false solutions, but they might lead to lesser number of model solutions than possible in some cases.

Lemmas:

- ⋆ "Removal of Tautologies only leads to simplification of the *(CNF)* formula, and has no effect on the possible solutions."
  Proof- Since tautologies are always true, and they are in *conjunction* with other clauses, removing them altogether has no effect on the satisfiability of the solution and nor the number of solutions.
- ⋆ "Simplifications/ Updates resulting from unit literals does not decrease the number of solutions, rather it keeps the number of solutions intact. Further, they can give rise to both unit literals and pure literals."
  Proof- For a clause with a single literal, there is a single way to satisfy it, and since it is in *conjunction* with other clauses, it is necessary for this uni-literal clause to be satisfied in order for the whole formula to be satisfied. Hence, each clause of "size" 1 fixes its literal globally. As a result, for the whole *CNF* to be satisfiable, it is necessary for this clause to be true, and consequently, for the literal to be fixed.
  They can give rise to unit literals because in some clauses, their negation has to be deleted, which might lead to isolation of the remaining literal.
  Eg.- $[(p_1),\ (\neg p_1,\ p_2)] \xRightarrow[\textit{Update}]{p_1} [\varnothing,\ (p_2)]$, thus making p_2 a unit literal.
  They can give rise to pure literals because they might delete some clauses which would lead to appearance of some clause in a singular fashion.
  Eg.- $[(p_1),\ (p_1,\ p_2),\ (\neg p_2)] \xRightarrow[\textit{Update}]{p_1} [\varnothing,\ \varnothing,\ (\neg p_2)]$, thus making $\neg p_2$ a pure literal.
- ⋆ "Simplifications/ Updates resulting from pure literals can give rise to only pure literals and strictly cannot give rise to unit literals."
  Proof- All the clauses which contain a pure literal can be satisfied just by satisfying the pure literal itself. Hence we can remove these clauses completely from the CNF formula by satisfying the pure literal. There would be no effect on remaining clauses, therefore no unit literals will be generated. Unit literals are generated only when individual literals are deleted from some clause, and not when clauses are deleted altogether
- ⋆ "Clauses to be updated from any particular literal can only be decreased and never be increased from updates due to other literals. More precisely, the clauses to be updated are always a subset of the set of all clauses, the literal was present in, in the original *CNF*."
  Proof- "Updation corresponding to any literal only affects the clauses containing itself or its negation. Since "updates from a clause cannot give rise to more occurrences of other literals, and after an "update from a clause, all its occurrences are deleted one way or the other, the lemma follows.
- ⋆ "Empty clauses can only be made from unit literals and not from pure literals."
  Proof- Can be inferred from this lemma. To reiterate, only unit literals can lead to removal of individual literals from a clause. Pure literals, by their definition, can lead to deletion of only complete clauses.
- ⋆ "The same update routine as followed in unit literals can be followed in pure literals despite the difference in consequences."
  Proof- In both unit literals and pure literals, a literal is fixed and all the clauses that contain the literal are removed. Moreover in case of unit literals, if any clause contains the negation

of the unit literal, then the negation of literal is erased from the clause. This case cannot arise in case of pure literals.
Since all occurrences of a literal are removed during its corresponding “update, hence, updating again corresponding to this literal in the same DPLL branch is moot.

Heuristics : We have majorly used *3* branching heuristics while deciding the literal *(during branching)* in the code for *DPLL*. We describe the property they rely on below-

2 Literal Clauses- Uses the literal which has the maximum count of occurrences in clauses of size 2 literals. This is our primary heuristic, we have assumed that since the clauses have only 2 literals, assuming one of the literals to be false, will give rise to a lot of unit literals, which is by far the most important part of the *DPLL algorithm* .

3 Literal Clauses- Uses the literal which has the maximum count of occurrences in clauses of size 3 literals. This is our secondary heuristic, in case, the primary heuristic is not able to find a valid candidate.
The choice is not immediately justifiable, but the intuition that the smaller sized clauses might collapse earlier resulting in unit literals is the main motivation behind its choice.

Occurrence- Our last resort is choosing a literal based solely on the number of its occurrences in the clauses. It trivially guarantees a literal for a non-empty *CNF*.
The intuition behind this choice is that the number of clauses will be decreased in large chunks on fixing a literal returned by this, hence reducing the number of clauses in size and possibly giving rise to pure literals.

The motivation behind implementing these heuristics at all was the run-time for larger inputs. It helps bring down the run-time to a few seconds from practically forever.

Assumptions : One of the most important reasons for our decreased run-time is the choice of heuristics. We do not justify their optimality formally, rather count on the argument that they are at least as good as a random choice for branching, which was our initial heuristic anyway and it did not prove to be *fast enough* for us in practice.
We also do not analytically prove the run-time complexity of the *DPLL algorithm* , rather use the simple argument that all the updates and heuristics are polynomial in the number of literals and clauses (with the used data structures). After that, the branching can be brute-force in the worst case which is exponential in the number of clauses.

References

- Wikipedia
- MOMS heuristic
- Heuristics Motivation
- Heuristics