CS202 Assignment 2

Manas Gupta (200554)

Priya Gole (200727)

---

1. SAT Solver:

Sol. We used the DPLL algorithm for building our SAT Solver.

DPLL is a backtracking-based algorithm for checking the satisfiability of the propositional logic formula of a cnf (conjunctive normal form).

First let us define all the terms we are going to use in our algorithm:

- Literal : a proposition or its negation (p or $\neg p$)
- Clause : disjunction of literals
- CNF : conjunction of clauses

The algorithm is based on building solution while trying assignments (true or false) to the literals, you build a partial solution which further proves the satisfiabilty or unsatisfiablity as we move on.

First, let us define a **unit clause** :

A unit clause is a clause where exactly one literal is unassigned and the other literals are assigned false.

The significance of a unit clause is that, we can determine the value of the unassigned literal - the literal must be true so as to satisfy the clause.

For eg, say we have $(x_1 \vee x_2 \vee \neg x_3)$ and we have $x_1 =$ false and $x_3 =$ true, then the clause is a unit clause, and for the assignments to be valid, x_2 must be true.

The basic idea of DLPP algorithm is:

- Randomly assign true or false to a literal.
- If there is a unit clause due to last assignment, then assign the needed value to the unassigned variable.
- Repeat the above steps; we either get a model or a conflict.
- If there is a conflict, backtrack to the last assignment and reverse its value (unit propagation) and again recursively apply DLPP.

Termination of DPLL:

- If there is nowhere to "backtrack" to change the assignment of variable then we get UNSAT.

- While assigning, we satisfy all clauses then we get SAT and the corresponding model to the CNF.

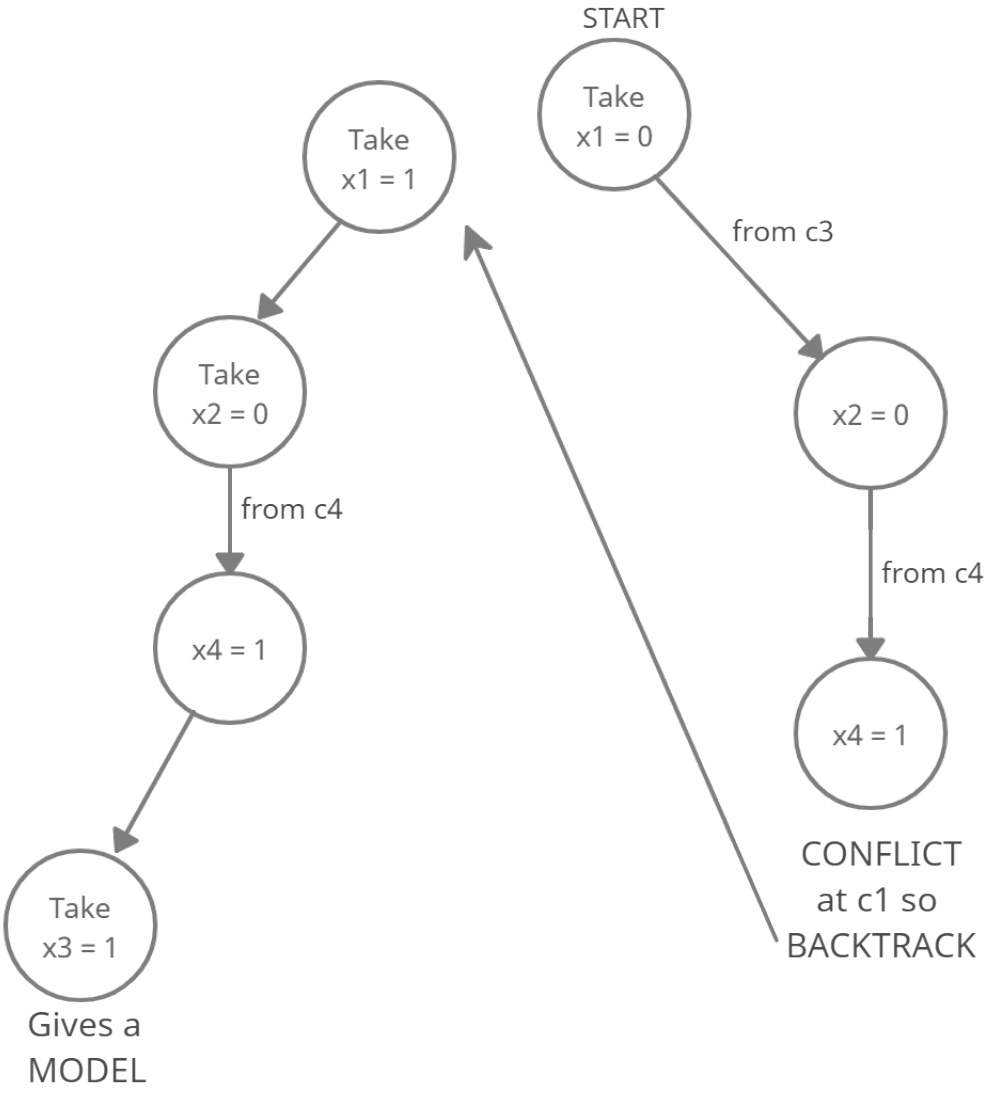
For example, we have the following CNF formulae:

$c_1 = x_1 \vee x_2 \vee \neg x_4$

$c_2 = x_3 \vee x_4$

$c_3 = x_1 \vee \neg x_2$

$c_4 = x_2 \vee x_4$



Our code implements DPLL algorithm to solve the CNFs.

- We declare a class called *Formula* to store the vector which will store all the clauses and literals.

- Then, we first take input from a CNF file and store the literals in a 2D vector clauses. We also keep track of the frequency of the literal, truth value of that literal and the difference in polarity of those literals.
  - We store positive literals in the form $10n$ or $n +' 0'$.
  - We store the negative literals in the form $10n + 1$ or $n +' 1'$.
  - We resize the vectors depending on the literal count and the clause count.
- Then we initiate the $solve()$ function with the $Formula$ object as a parameter. The solve function implements the DPLL algorithm on the Formula. The $Algorithm()$ function is initiated with the $Formula$ object as a parameter. The result of the $Algorithm()$ function is passed on to the $result()$ function to display the result.
- Inside the $Algorithm()$ function, we perform the unit propagation on the formula. If the formula get satisfied, we show the result and mark it as completed else if the formula isn't satisfied then we return it normally i.e. without any propagation. Then we find a variable with maximum frequency and which is not assigned a value and we start assigning the truth values. We apply a loop, for truth values, first for true and then for false. We assign the truth values and apply that transformation to all the clauses using $transform()$ function. If the result from this transformation come satisfied then we show the result and mark it as completed. Otherwise, if the formula is not satisfied, we return the formula normally. For any other case, we recursively call the $Algorithm()$ function and forward the result.
- In the $uni\_prop()$ function, we declare a bool named $found$ to get if a unit clause is there or not. If the size of the clauses is zero i.e. if there is no literal, then the formula is already satisfied. Else we loop through the clauses while a unit clause is found and assign the truth value to the unassigned literal in the clause and mark the clause as closed. We apply the transformations and return the result.
- In the $transform()$ function which takes the formula and the target literal as parameters, we loop through all the clauses and the literals and if the literal applies with same polarity as it is being targeted we remove the clause from the list and if no clause remains then return satisfied. Else, we remove the literal from the clause as it is false in it. If no clause remains empty, we return formula as unsatisfied for now.
