CS202 Assignment 2

README

We have used C++ as the language for implementation and used the DPLL algorithm. For running the SAT solver, add the CNF file path in the solver.cpp (in line 87 of the file) and then run the cpp code using terminal as follows:-

```
g++ solver.cpp
./a.out
```

For making the SAT solver, we are using DPLL ,the **Davis–Putnam–Logemann–Loveland** (**DPLL**) **algorithm**, which is complete backtracking algorithm which starts by assigning truth value to a literal and check satisfiability by simplifying formula using recursion.If the model comes out to be satisfiable, very good. Else, assign a false value and repeat the recursion part until we reach SAT or UNSAT.
The algorithm can be further explained by:-

- First, we start by choosing a clause with the minimum of literals among all the list of clauses.
- Then, in the clause, we find the literal and its negation occurring for the maximum number of times in all clauses and try to simplify the formula

- For simplifying the formula we remove the complete clause with the literal and for the negation of the literal we only remove the literal from the clause.
- We repeat the simplification until we get either an empty clause or a contradiction i.e UNSAT for both the assignment of literal in that case we return UNSAT. Else, in case we have no clauses left , then we return SAT.
- For storing the model we use a vector which we update depending on the satisfiability of the model.

Pseudo-code

```
Algorithm DPLL
    Input: A set of clauses Φ.
    Output: A truth value indicating whether Φ is satisfiable.

function DPLL(Φ)
    while there is a unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    while there is a literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    if Φ is empty then
        return true;
    if Φ contains an empty clause then
        return false;
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {not(l)});
```

Assumptions

1. We have assumed the input to be in CNF format.
2. If the formula has multiple models that satisfy it, then we have considered the don’t care literal (whose truth value will not affect the satisfiability of the model) to be true.

Dependencies

GNU C++ compiler or any other compiler to compile C++ code.