Bonus SAT solver algorithm description

21017474 Cheng Dai

The algorithm I used in the bonus part is the DPLL sat solver with BCP and PLP:

As the input conjunctive normal form (CNF) is the form of vector<vector<int>>, I use int number as literal, inserting its value or conversed value to the set<int> assignment map to proceed DPLL.

In the BCP and PLP functions, I implemented them in a traditional way as taught in class, the <algorithm> library in C++ gives me a convenient way to drop clauses from formula and drop literals from clause by using function “remove” and “remove_if”

In the DPLL function, The novel point is that I used a helper function named “checkcnf” to check and simplify the CNF by removing clauses that are already satisfied and by removing literals that are made false by the assignment. If the clause is already satisfied, then it won’t be added to the new formula.

The DPLL function works by recursively trying to simplify the formula by setting one of its variables to true or false, it terminates when the formula is either empty (indicating that it is satisfied) or if it contains an empty clause (indicating that it is unsatisfiable).

```
vector<vector<int>> checkcnf(const vector<vector<int>>& formula, int literal) {
    vector<vector<int>> simplified_formula;
    for (const auto& clause : formula) {
        // check if the clause is already satisfied
        bool satisfied = false;
        for (const auto& l : clause) {
            if (l == literal || l == -literal) {
                satisfied = true;
                break;
            }
        }
        if (!satisfied) {
            // remove the literal from the clause if it is made false by the assignment
            vector<int> new_clause;
            for (const auto& l : clause) {
                if (l != -literal) {
                    new_clause.push_back(l);
                }
            }
            simplified_formula.push_back(new_clause);
        }
    }
    return simplified_formula;
}
```

By implementing this function, the efficiency of DPLL is boosted. Because in every

recursion the checkcnf function will reduce the CNF size or reduce the size of certain clauses in CNF. With the help of BCP and PLP, after only a relatively lower amount of recursion, answer will be generated. And even without the help of BCP and PLP, this algorithm will generate a correct answer in a short time.

It should be noted that the AST generate method I used in PA1 takes relatively longer time to proceed. This might lead to longer respond time (up to few seconds) for extra complicate test case, but the algorithm used in PA2 and Bonus is high efficient and robust.