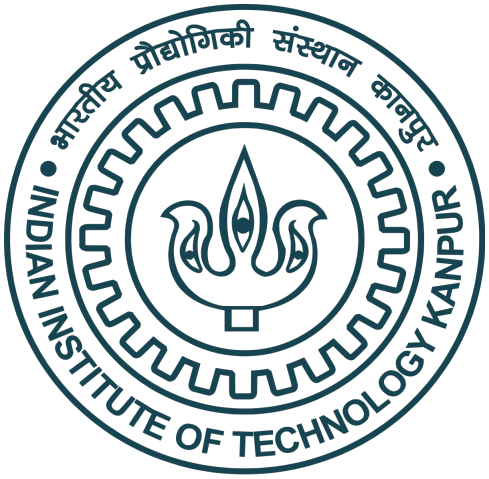
CS202A: Logic in Computer Science



Assignment 2

Suket Raj 201013

Sweta Kumari 201036

February 18, 2022

Problem Statement

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

1. a model if the formula is satisfiable
2. report that the formula is unsatisfiable

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

Approach

We will be using improvised **DPLL algorithm** for solving SAT problems.

- **Motivation :** It is important to not to get close to worst case while solving a CNF-SAT problem. DPLL provides variety of open options for optimizations, some major optimizations mentioned below.
- **Introduction to DPLL :** Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a backtracking algorithm for soving CNF-SAT problems.
  The core of the algorithm is, assigning truth value to a literal and recursively check for satisfiability. If not satisfiable, check for the other truth value.
  So, at each step the problem gets broken down to two simpler subproblems.
  We will apply a few optimizations at each recursive step to enhance the algorithm.
  - **Unit Propagation :** If a clause contains only one literal, this clause can only be satisfied by assigning the literal true. So we iterate through clauses and search for such literals assign them respective truth value throughout the formula.
  - **Pure Literal Elimination :** If a literal appears in only one polarity(either positive or negative but not both) throughout the formula, it is called pure literal. Thus, assigning the pure literals any truth value does not lead to contradiction. So we assign them in a way such that the clauses containing the literals become true.
- **Some more optimizations :** Apart from the ones mentioned above, we can also perform some optimizations while solving the problem recursively. As stated above, each problem gets broken into two subproblems when chosen a literal for assumption of truth value.

  We propose *Two level of optimizations* here:

  1. **The literal to be chosen** : We will choose the literal whose frequency is maximum. This will maximise the clauses to be checked in a single go.
  2. **The subproblem to be solved first** : The literal will be assigned a true value if the frequency of positive polarity will be more in the formula than negative, and vice versa. The equality condition can be taken in either way.

  Before implementing any of these, we will first make sure that each clause :

  - do not contain duplicates.
  - if contains both positive and negative polarity of a literal, the clause is removed as $p \vee \neg p = 1$.

Implementation

- Class **Formula :** Keeps the formula to be solved, frequencies of literals, and the truth value assigned to the literals.
- Class **Solver :** Performs following functions:
  - *take_input :* Takes input in CNF form from given file.(*Public Function*)
  - *decipher :* Calls the private solve function with the formula contained.(*Public Function*)
  - *unit_propagate :* Performs unit propagation on the formula given(*Private Function*)
  - *eliminate_pure_literals :* Performs pure literal elimination on the formula given(*Private Function*)
  - *update_formula :* apply the truth value of a given literal to whole formula(*Private Function*)
  - *solve :* Apply all the optimizations mentioned, and solve the formula recursively(*Private Function*)
  - *print_model :* gets called when SAT or UNSAT is proved, prints the literals with positive or negative sign to denote their truth values.(*Private Function*)
- After taking input from the file and storing in a set of clauses, we recursively call the solve function to solve the formula given. At each step of recursion, we first do unit propagation till no unit clause is left. Then we eliminate all the pure literals. After that, we check which literal has maximum frequency, we choose that and assign both the truth values one by one depending upon which polarity has higher frequency. If the solver reaches **SAT** at any branch, it prints the current model. However if the solver reaches **UNSAT** at any branch, it **backtracks** and reaches the previous state. If all the branches return UNSAT then the formula is UNSAT.
- **Conventions :**
  - While solving the problem, **zero-based indexing** is followed.
  - For functions *unit_propagate, eliminate_pure_literals, update_formula* the following conventions for return statement is used:
    - res=1 : SAT
    - res=0 : UNSAT
    - res=2 : Nothing proved yet
  - For function *solve* the following convention is used:
    - res=1 : current branch is SAT, model achieved
    - res=0 : current branch is UNSAT
  - Two types of frequencies are taken in the solving process, one for signed literals (signed means different for positive and negative values of literals), the other for signed literals.
  - Literals are assigned 1 if given positive (true) value, and 0 for the other. -1 means the literal haven't been assigned a value yet.
  - The function *print_model* contains exit(0) so that the program terminates after printing the model and only one model is printed.

Test Bench Results

Test Case Name	Number of Literals	Number of clauses	Result	Time Taken (is s)
uf20-01.cnf	20	91	SAT	0.005
uf20-02.cnf	20	91	SAT	0.003
uf20-03.cnf	20	91	SAT	0.004
uf20-04.cnf	20	91	SAT	0.004
uf20-05.cnf	20	91	SAT	0.003
uf20-06.cnf	20	91	SAT	0.003
uf20-07.cnf	20	91	SAT	0.003
uf20-08.cnf	20	91	SAT	0.002
uf20-09.cnf	20	91	SAT	0.003
uf20-010.cnf	20	91	SAT	0.003
uf150-01.cnf	150	645	SAT	15.822
uf150-02.cnf	150	645	SAT	4.561
uf150-03.cnf	150	645	SAT	25.404
uf150-04.cnf	150	645	SAT	30.123
uf150-05.cnf	150	645	SAT	0.568
uf150-06.cnf	150	645	SAT	10.756
uf150-07.cnf	150	645	SAT	5.864
uf150-08.cnf	150	645	SAT	0.520
uf150-09.cnf	150	645	SAT	0.600
uf150-010.cnf	150	645	SAT	18.987
uuf150-01.cnf	150	645	UNSAT	41.066
uuf150-02.cnf	150	645	UNSAT	49.693
uuf150-03.cnf	150	645	UNSAT	61.818
uuf150-04.cnf	150	645	UNSAT	38.771
uuf150-05.cnf	150	645	UNSAT	28.920
uuf150-06.cnf	150	645	UNSAT	24.646
uuf150-07.cnf	150	645	UNSAT	29.364
uuf150-08.cnf	150	645	UNSAT	17.952
uuf150-09.cnf	150	645	UNSAT	41.135
uuf150-010.cnf	150	645	UNSAT	43.172

- **Observations and Limitations :**
  - It is observed that the program runs within a second for formulae with 20 literals.
  - The program takes about half a minute on average for formulae with 150 literals which is still far better when compared to exponential time complexity.
  - The optimizations used while solving the problem recursively seem to work properly.

References

- **DPLL Algorithm (Wikipedia)**
- **CNF-SAT Problem (Wikipedia)**
- **Provided Test Bench**