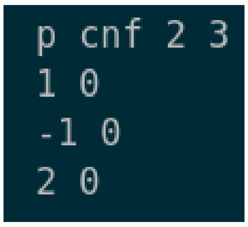
CS202-Assign2

Soham(180771) | Aryash(190189)

Representation:

We store the formula in a vector of vectors.
Let us say the formula in DIMACS representation is:



```
p cnf 2 3
1 0
-1 0
2 0
```

Then we have v[0]={1}, v[1]={-1}, v[2]={2}, where v is the vector of vectors.
Note that 0 is not a literal, and is only used as a delimiter.
Each sub-vector corresponds to the literals in one particular clause, where clauses are disjunctions of literals.

Logic:

We have borrowed a few ideas from the DPLL algorithm.
Unit propagation - If a clause is a *unit clause*, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. Unit propagation consists in removing every clause containing a unit clause's literal and in discarding the complement of a unit clause's literal from every clause containing that complement. In practice, this often leads to deterministic cascades of units, thus avoiding a large part of the naive search space.

After storing the formula in a vector of vectors, we pass it to a recursive function "solve".
Here are the steps which "solve" performs:

1. Find the shortest length clause among all clauses present in the formula.
2. Consider all literals present in that clause. Find the literal which occurs the maximum number of times in the entire formula. This is done so that we can reduce the remaining formula by a significant amount every time, thereby speeding up the recursive algorithm.
3. Set this literal to TRUE.
4. Shorten the list of clauses. For all clauses where this literal was present, we can safely remove the entire clause, because the disjunction of this literal with any other arbitrary number of literals will of course be true. For all clauses where the negation of this literal was present, we can remove that particular negation of the literal from that clause. This is because a false value in the disjunction of a false value with some other literals does not matter.
5. Recursively call solve. If we can find a solution, then the formula is satisfiable, with the variable in question set to TRUE.
6. Otherwise, try to set this literal to FALSE.
7. Again, shorten the list of clauses.
8. Recursively call solve. If we can find a solution, then the formula is satisfiable, with the variable in question set to FALSE.
9. If no solution is found, the formula must be UNSAT.

Logic for checker:

Checker stores the clauses in the form of vector of vectors. It then reads “output.txt” to find the assignment for every literal. Then it takes the disjunction of all the literals in a clause. If every such clause evaluates to TRUE, then it shows correct assignment. Otherwise, it will show wrong assignment.

Utility of each function:

- shortest_clause returns the index of the clause with the minimal length.
- shorten_formula trims the formula, given some variable is TRUE or FALSE.
- maxoccur gives the variable which occurs the most number of times in the entire formula.
- solve- recursive formula to solve the given sat-problem.

Assumptions:

- The directory will contain “output.txt”,”input.txt” and “satsolver.cpp” along with Makefile,README.
- The input cnf file will be a valid CNF file, according to the DIMACS representation.

Limitations:

- Since recursion is involved, the code takes a significant time to execute for a large number of literals (>150).

Further Optimisations:

- We can borrow ideas from the DPLL algorithm to speed up the recursive procedure.
- Pure literal elimination - If a propositional variable occurs only with one polarity in the formula, it is called pure. A pure literal can always be assigned in a way such that makes all clauses containing it as true. Thus, when it is assigned in such a manner, these clauses don’t constrain the search anymore, and can be deleted.
