CMSC 301 A Backtracking SAT Solver*
Due 12.April.2019

Objective

Develop a faster solver in Java for the Satisfiability problem using a backtracking approach. You will modify methods in Formula.java. Do not change the class or method names. You may need to add new methods.

Preliminaries

Refer to the previous coding project if you have forgotten the 3-SAT problem or the organization of the formula files. I have provide the basic algorithm in SATBacktrack.java and the file-reading code within Formula.java. Your job is to efficiently implement the methods in Formula that support variable assignment and backtracking. Download the code proj02.tgz on Moodle.

Your programming project

You will develop a solver for the Satisfiability problem, a program that receives a Boolean formula in conjunctive normal form (CNF) as input and computes a satisfying assignment for it. If no satisfying assignment exists, your program will report that the input formula is unsatisfiable.

For this project, you will develop a solver that uses the Davis-Putnam algorithm. At any given point, a DP solver has:

- A partial assignment (an assignment of values to some of the variables in the input formula); and
- An unsatisfied formula (a set of clauses, that may be different from the input formula).

There are two things to note about this unsatisfied formula:

1. It contains only clauses that are unsatisfied by the current partial assignment.

2. Each of these clauses only contains unassigned literals (assigned literals that do not satisfy a clause are "removed" from the clause, either literally removed, or marked and ignored). For example, suppose that the algorithm starts out with a formula with a single clause (-1 2 3), and the completely unassigned partial assignment of 3 variables (-, -, -). Now suppose that the algorithm assigns the value true to x1. The clause cannot be satisfied on account of its first term, which is assigned but to a value that is not helpful to the clause, so this term will be (temporarily) removed from the clause.

The DP algorithm (see SATBacktrack.java:dp()) proceeds as follows:

- Any variable that is currently unassigned is chosen to be the next "branch variable," var.
- The branch variable var is assigned an initial (arbitrary) value of true. This assignment of a value to var has two consequences that preserve the properties of the formula discussed above:
  - any clauses in the formula that contain var are now satisfied, thus they are "removed" from the formula; and
  - any occurrences of the literal -var in other clauses are "removed."

Note that the DP algorithm example just outlined is for variables assigned true. The same logic applies to variables assigned false, since they satisfy clauses containing their negation.

*This material is based upon work supported by the National Science Foundation under Grant No. 1140753.

The dp algorithm extends the partial assignment by assigning a value to the branch variable. It then calls itself recursively to attempt to extend the current partial assignment even more. It may encounter the following cases:

- If the formula is empty (because all its clauses have been satisfied and removed) then the current assignment is a satisfying assignment (the first base case in the dp method).
- If one of the clauses in the formula is empty (because its last literal was removed and thus no refinement of the current partial assignment could possibly satisfy this clause or the formula), the algorithm backtracks. The algorithm backtracks by changing the value of var from true to false and calls recursively to try to extend this new partial assignment; if this also results in an empty clause, var will be unassigned (backtracking to an earlier partial assignment saved in a stack) and some earlier branch variable assignment will be changed. Notice that unassigning or changing the value assigned to a variable may cause changes in the formula. When a variable is set or unset, you must update the clauses in the stack as well as the variables in each clause that are still viable. In the implementation I have given you, this is handled as follows:
    - A vars variable indicates whether variables are true, false, or unassigned.
    - A clauseStack is maintained. This is a stack of LinkedList<Integer> lists. These lists provide a list of unsatisfied clauses at any point in the execution of dp. Only unsatisified clauses need be considered when considering the effect of a new variable assignment. Keeping this stack of lists of clauses in good order is one of your challenges.
    - A form variable to hold the formula. This is a two-dimensional array in which each row corresponds to a clause. Individual values are positive or negative integers the absolute value of which corresponds to a particular boolean variable. I have not specified how you ought to mark variables as they are removed from consideration. Again, that is for you to figure out.

You should run extensive tests using the small problems I have provided in this directory as well as previous 3-SAT problems from your Brute Force Solver. A good solution to this problem will solve u30.cnf in less than 500 milliseconds and uuf50-010.cnf in about 90 seconds.

Once your testing is complete, submit Formula.java on Moodle (no tar files, zip files, class files, other files, etc.).