DPLL SAT Solver

Implementation

- We have implemented Davis-Putnam-Logemann-Loveland algorithm-based SAT solver in C++ language.
- First, we take the input directly from the CNF file using ifstream.
- Then, we store the clauses in a 2D vector.
- Now for each literal, we have also kept two more 2D vectors containing the clauses, one in which the literal is present and the other containing the clauses in which its negation is present.
- For each literal, we maintain track of its positive activity as well as negative activity as we get a conflict while in a decision track. This helps us to backtrack more efficiently as recent conflicts are given more priority.
- If all literals lead to conflicts, then the result is given as “unSAT”. Else, if a model is found, then “SAT” is displayed along with a satisfying model.

Member functions:-

- void takeInput(string filename) :- It takes the input from the file and store the clauses.
- unsigned int var(int lit) - absolute value of a literal which is the variable ● int currValLit(int lit) :- It returns the current value of literal from the model.
- void setTrueLit(int lit) :- This sets the value of a literal to true.
- void updLitAct(int lit) :- This increments the value of activity of literal based on whether the literal was present or its negation was present in the conflicting clause of a decision path.
- void updConflictClauseAct(const vector<int>& clause) :- This function updates the activity of the literals present in the conflicting clause using updLitAct. As recent conflicts need to be given higher priority, thus a regular decrease in the activity of all literals is required which is taken care of by this function.
- bool propGiveConflict() :- This function checks whether a propagation path gives a conflict or not.
- void backtrack() :- This function backtracks to the previous valid decision level.
- int getNextDecLit() :- This function returns the most active literal on which the next decision path need to be traversed.
- void chkmodel() :- This function check the model.
- void outSAT(bool sat, vector<int> models) :- This gives the output.
- void exeDPLL() :- This executes the DPLL algorithm on the clauses given.
- void chkUnitClause() :- DPLL algorithm needs to check the special case if the clause given is a unit clause or not. This checks that case and gives output directly if that is the case.

Member variables:-

- Unsigned int numVar - to store the number of variables in the system
- Unsigned int numClause - to store the number of clauses passed to the file

- Vector <int> model - to store the computed model
- Vector <int> modelStack - to temporarily hold the decision path that we have chosen
- Unsigned int nextLiteralIndex - to store the index of the upcoming literal in the decision tree
- Unsigned int decLvl - to store the level of our decision (every assumption corresponds to a progression in decision level
- Vector <double> posLitActivity - to store the number of conflicts associated with each variable
- Vector <double> negLitActivity - to store the number of conflicts
- Unsigned int conflicts - to store the number of conflicts
- Unsigned int propagation - to store the propagation
- Unsigned int decisions - to store the number of decisions made

<u>Assumptions</u>

There are no assumptions.

<u>Limitations</u>

There are no limitations.

<u>How to Run?</u>

- We ran the program by passing the DIMACS CNF file as command line argument:

```
./solver <filename-including-path>
```

- To run the solver on cnf files by running the cpp code:
    - First run the command “g++ solver.cpp -o solver”
    - Then run the command “./solver <path_to_cnf_file>” where <path_to_cnf_file> refers to the path to the CNF file on which the code needs to be executed.

<u>How to check with the testcases</u>

- We tested the program by passing the following command.

```
./tester.sh
```

- We are using python-sat to check if the model given by our solver is correct or not.
- To check with the testcases:
    - Keep the “solver.cpp”,”tester.sh”, “checker.py” files and the directory “testcases” (having testcases files) at same location
    - Run the bash script “**./tester.sh**” in the terminal.
    - It will give the name of the file along with True or False depending on whether our model is correct or not

- Joint Project of Shashwat Gupta and Ujwal Jyot Panda