SAT SOLVER Based on DPLL

We have implemented an efficient SAT SOLVER using DPLL and some heuristics. It is implemented in C++ for lower solving times.

Index

Installing and Running The Solver

For using this sat solver extract all

the contents of zip file into another folder. Open unix terminal and `cd` into the extracted solver directory. Using `ll` check that **run_all_cases.sh** has executable permissions for the current user. If not, execute the following command:

```
sudo chmod +x run_all_cases.sh
```

Input

Put the test file(s) in Testcases folder. The input file must be in DIMACS format.

Output

If the solver is running on `test.cnf`, then output will be as follows:

- UNSAT/SAT
- Model if SAT
- Execution time in seconds

Now run the following command to run the SAT SOVLER on all the testcases in *Testcases* folder.

```
./run_all_cases.sh
```

Alogrithm and Heuristics

We have used DPLL as base algorithm along with some heuristics to build SAT SOVLER. The main difference which makes this SOLVER so efficient is that we backtrack using decision levels and not single literals.

Pre Processing Details

We start with reading the input into the respective data structures (see comments in the code for more details), and after populating the data structures, *filterClauses* function is called which :

- Removes the Unit Clauses from the original clauses and sets the literal to true. If we encounter two unit clauses which contain opposite literals, then we exit the solver and output UNSAT.
- Removes the Tautoligies from the remaining clauses.
- Removes the clauses containing Pure Literals from the remaining ones.

Now `initializePosandNegClauses` initializes two array of array of addressses corresponding to clauses containing the respective literal in positive state and negative state.
Finally after all the pre-processing has been done, we move to the actual solving part.

Algorithm

A `model_stack` is maintained which contains decision levels propogated till now. A decision level is created by pushing literals to the stack and checking for conflicts in that decision level. If conflict arises in any clause, program backtracks to previous decision level and again push first literal in removed decision level, but in opposite state. Along with that value of that literal in *model* is also changed. The removed literals are set to `UNDEFINED`. Program checks for conflicts and backtrack till it reaches `DECISION_LEVEL=0` or get no conflict before that. If first case occurs, program exits with UNSAT. Else it continues to next decision level by choosing next literal to propagate. Two adjacent decision levels are seperated by `0` between them in the `model_stack`. The next literal to propagate is chosen by a special heuristic. If it is 0, then the program exits by SAT and gives the required model.

Heuristics

Activity based Decision Heuristic

The heuristic used to choose the next literal to be decided by the DPLL algorithm has also been updated with respect of what was implemented in the example code that was provided.

The decision heuristic works as follows:

- An activity counter is kept for each literal, be it positive or negative, indicating the number of conflicts in which the literal was involved.
- Each time a conflict is found, the activity of all the literals in the clause that causes the conflict is incremented by a constant factor (1, in our case).
- When a new decision has to be made, the literal with the highest activity of those variables still undefined in the model is chosen and returned to the DPLL procedure.

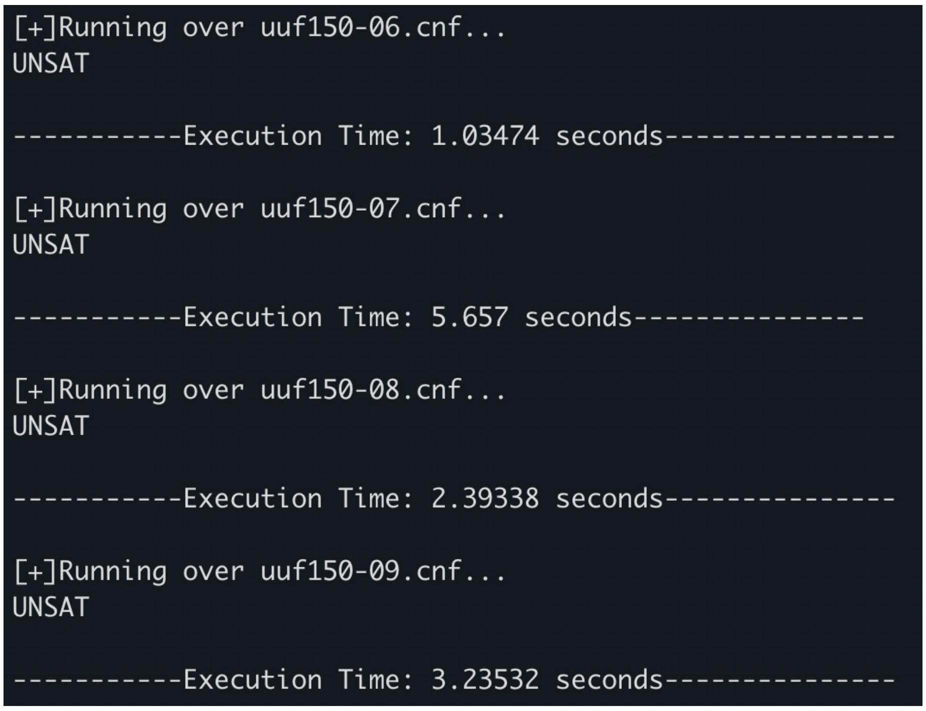
Inference based Heuristic

Another heuristic implemented is that whenever during checking for conflicts, program finds a clause which has all of its literals set to `FALSE` except for one `UNDEFINED`, then it sets the literal to `TRUE` and pushes it to the current decision stack. It considerably reduces the execution time as number of literals to propagate reduces in constant operation time.

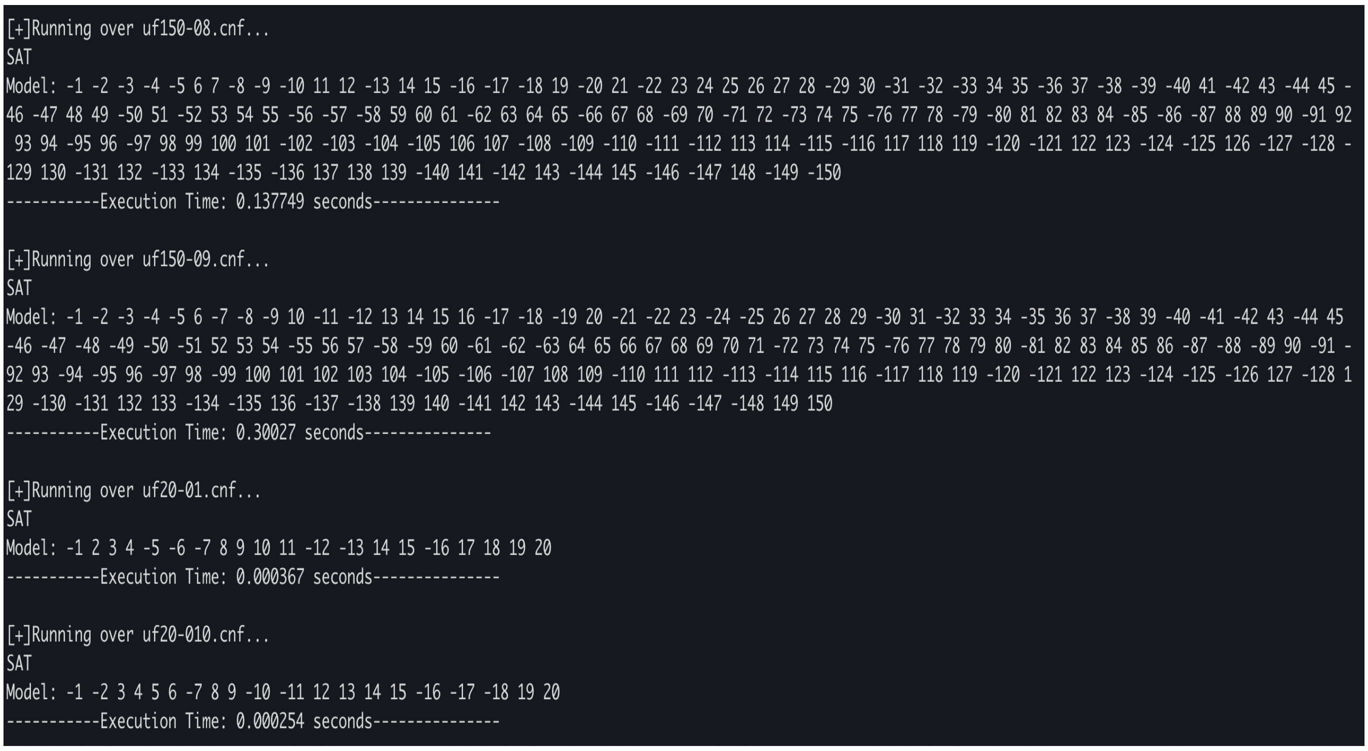
Testbench

We tested our solver on the provided testcases, and the Solver was able to process all the testcases in the zip file under 2 minutes.

Execution Results



```
[+]Running over uuf150-06.cnf...
UNSAT

-----------Execution Time: 1.03474 seconds---------------

[+]Running over uuf150-07.cnf...
UNSAT

-----------Execution Time: 5.657 seconds---------------

[+]Running over uuf150-08.cnf...
UNSAT

-----------Execution Time: 2.39338 seconds---------------

[+]Running over uuf150-09.cnf...
UNSAT

-----------Execution Time: 3.23532 seconds---------------
```



```
[+]Running over uf150-08.cnf...
SAT
Model: -1 -2 -3 -4 -5 6 7 -8 -9 -10 11 12 -13 14 15 -16 -17 -18 19 -20 21 -22 23 24 25 26 27 28 -29 30 -31 -32 -33 34 35 -36 37 -38 -39 -40 41 -42 43 -44 45 -
46 -47 48 49 -50 51 -52 53 54 55 -56 -57 -58 59 60 61 -62 63 64 65 -66 67 68 -69 70 -71 72 -73 74 75 -76 77 78 -79 -80 81 82 83 84 -85 -86 -87 88 89 90 -91 92
 93 94 -95 96 -97 98 99 100 101 -102 -103 -104 -105 106 107 -108 -109 -110 -111 -112 113 114 -115 -116 117 118 119 -120 -121 122 123 -124 -125 126 -127 -128 -
129 130 -131 132 -133 134 -135 -136 137 138 139 -140 141 -142 143 -144 145 -146 -147 148 -149 -150
-----------Execution Time: 0.137749 seconds---------------

[+]Running over uf150-09.cnf...
SAT
Model: -1 -2 -3 -4 -5 6 -7 -8 -9 10 -11 -12 13 14 15 16 -17 -18 -19 20 -21 -22 23 -24 -25 26 27 28 29 -30 31 -32 33 34 -35 36 37 -38 39 -40 -41 -42 43 -44 45
-46 -47 -48 -49 -50 -51 52 53 54 -55 56 57 -58 -59 60 -61 -62 -63 64 65 66 67 68 69 70 71 -72 73 74 75 -76 77 78 79 80 -81 82 83 84 85 86 -87 -88 -89 90 -91 -
92 93 -94 -95 96 -97 98 -99 100 101 102 103 104 -105 -106 -107 108 109 -110 111 112 -113 -114 115 116 -117 118 119 -120 -121 122 123 -124 -125 -126 127 -128 1
29 -130 -131 132 133 -134 -135 136 -137 -138 139 140 -141 142 143 -144 145 -146 -147 -148 149 150
-----------Execution Time: 0.30027 seconds---------------

[+]Running over uf20-01.cnf...
SAT
Model: -1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20
-----------Execution Time: 0.000367 seconds---------------

[+]Running over uf20-010.cnf...
SAT
Model: -1 -2 3 4 5 6 -7 8 9 -10 -11 12 13 14 15 -16 -17 -18 19 20
-----------Execution Time: 0.000254 seconds---------------
```

Glossery

Unit Clauses: The clauses which contain only one literal. For them to satisfy, that literal must be true.
Tautology: The clause which is always true, irrespective of the interpretation of the literals.
Pure Literals: They are the literals that for which their negation is not present in any of the clauses. They can be assumed to be true and then all the clauses containing 'a' can be deleted and all the clauses '-a' can be reduced by in size by eliminating '-a'.