Introduction to Formal Methods **Fall 1397**

Assignment 1: SAT Solving

Hossein Hojjat & Fatemeh Ghassemi *Mehr 03*

1 Introduction

The goal of this assignment is to write some simple SAT solvers and to compare them against an off-the-shelf solver. For implementing the SAT solvers in this assignment you are free to use any programming language that you are comfortable with.

2 DIMACS format

The standard way to represent a Boolean formula in conjunctive normal form is the DIMACS format. The format is used as input to modern SAT-solvers. A file in the DIMACS format begins with a line specifying that the formula is in normal form, the number of variables in the formula, and how many clauses it contains. For example, `p cnf 4 3` specifies that the file contains a formula in conjunctive normal form with 4 variables and 3 clauses. Then, the next lines describe the clauses, one on each line. Each line contains a list of positive or negative integers, and ends with a zero. A positive integer `i` indicates that the ith variable appears in the clause, whereas a negative integer `-i` indicates that the negation of the ith variable appears in the clause. For example, here's a formula, followed by its encoding using the DIMACS format: $(x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_4) \wedge (x_2 \vee \overline{x_3})$

```
c
c start with comments
c
p cnf 4 3
1  3  -4  0
4  0
2  -3  0
```

Write a function that takes three parameters (number of variables, number of clauses, filename) and makes a random Boolean formula for the given parameters. The function writes the result formula in DIMACS format into the given filename.

3 Truth Table Algorithm

Write a function that gets a DIMACS benchmark file and uses the truth table algorithm to solve it. You can return a Boolean value from the function to show the satisfiability of the given input. You may assume that the input file is a correct DIMACS benchmark for a CNF formula, you do not need to check the input file.

4 Parallel Truth Table Algorithm

Write a function that gets a DIMACS benchmark file and uses a concurrent version of the truth table algorithm to solve it. Write a process that takes a Boolean formula and checks a subset of rows in its truth table. Spawn multiple instances of that process to cover all the rows. For example, for a Boolean formula with 4 variables, there are $2^4 = 16$ rows in the truth table. You can use one process to check 8 rows and another for the rest of 8 rows. Or, you can use 4 processes that each of them checks 4 rows of the truth table. The process gets a parameter (in the message) to know which subset of the table it should check. The function in this task should also take an argument to know how many processes it should spawn to check the Boolean formula. The function should assert that the given number is only a power of 2. You can return a Boolean value from the function to show the satisfiability of the given input. You may assume that the input file is a correct DIMACS benchmark for a CNF formula, you do not need to check the input file.

5 DPLL

Write a function that gets a DIMACS benchmark file and uses the DPLL algorithm to solve it. You can return a Boolean value from the function to show the satisfiability of the given input. You may assume that the input file is a correct DIMACS benchmark for a CNF formula, you do not need to check the input file.

6 Report

You are now able to compare the different versions of the your SAT solvers. To have a comparison with a real SAT solver, your comparison will also include the Sat4j solver. First make a good library of benchmarks using the function of the first task. The sizes of the benchmarks should be large enough to challenge your SAT solver, try to avoid creating tiny hello-world examples that can be solved in milliseconds. The smallest benchmark should take the truth-table solver several seconds to solve. Try to generate at least 50 different benchmarks with different sizes. Record the elapsed time for the truth-table solver, parallel truth-table solver (with 2, 4 and 8 processes), your DPLL solver and finally Sat4j. To ensure that your SAT solvers are correct, you need also to check their results against the output of Sat4j. Draw a plot to compare the results similar to the plot that was shown in the class (Moshe Vardi). Write a report (maximum one page) to justify the results of your experiments. In case that the truth table algorithm works better than DPLL, you need to provide some reasons that why your DPLL solutions did not work as fast as intended. Also include some of your suggestions to improve the performance of the DPLL SAT solver.

7 Deadline and Deliverables

The deadline of this project is Mehr 13th at 11:59pm. Please email only your source files and your final report (do not include benchmark files or executable files). Make a zip from your files to make uploading the files easier. Late submissions won't be accepted for this assignment.