Executing WhyML Programs

15-414: Automated program verification

Introduction

In the previous lab, we have seen how to execute WhyML code by using the Why3 interpreter. For part I of Lab 4, we will start by testing if your SAT solver using Boolean Constraint Propagation (BCP) is returning the correct answer for a small set of benchmarks. Instead of using the Why3 interpreter, we will compile the WhyML code to OCaml. [1] Since testing does not guarantee that your code is correct, you will need to verify its correctness for part II of Lab 4. Additionally, you will also have to implement at least one technique from the list of suggestions that we will provide on a separate document and that were introduced in Lecture 20. [2] The final submission of Lab 4 will be used as your entry to the verified SAT competition.

Executing WhyML Programs

To help you run WhyML programs as OCaml code, we provide you a CNF parser and scripts to transform and execute your OCaml code. These files are available at:

`https://www.cs.cmu.edu/~15414/labs/dimacs_parser.zip`

This .zip file contains:

- `'build.sh'`: build script that will compile your WhyML code into OCaml. This script takes as input an '.mlw' file that contains your SAT solver implementation. We suggest to use your implementation from Lab 3 to do some initial testing. To run this script execute the following command in your terminal:

[1] This is the same procedure that will be done for the verified SAT competition.

[2] Available at `https://www.cs.cmu.edu/~15414/lectures/20-sat-techniques.pdf`

```
$ ./build.sh 'filename.mlw'
```

This will create a subdirectory 'exec' that will contain the OCaml code.

- `'main.ml'`: contains a parser that reads CNF formulas and gives them as input to your `'sat'` function.
- `'formulas'`: this directory contains 20 benchmarks from pigeon holes, counting and parity problems. The filename of each benchmark describes if the benchmark is either satisfiable or unsatisfiable.
- `'test.sh'`: this script will the OCaml code of your implementation on all the benchmarks in the 'formulas' directory.

Testing your SAT solver implementation

To test your SAT solver implementation you should first compile your code into OCaml by running the `'build.sh'` script as described in the previous section. You should have already all dependencies installed in your system to run this script. To run your SAT solver implementation on a single instance you can run the following command:

```
$ exec/main.native 'filename.cnf
```

For example, running:

```
$ exec/main.native formulas/count-4-2-sat.cnf
```

Should have the following output:

```
sat
execution time:0.000000s
```

All the benchmarks in the `'formulas'` directory should be solved in less than 10 seconds in a common laptop with your SAT implementation of Lab 3. Therefore, they should be relatively easy to be solved by your new SAT solver implementation that uses BCP. Note that the name of the filename will tell you if the benchmark is either satisfiable ('sat') or unsatisfiable ('unsat').

```
[15-414] Bug Catching: Automated Program Verification
[15-414] Benchmark formulas/count-4-2-sat.cnf                    [OK]
[15-414] Benchmark formulas/count-4-3-unsat.cnf                  [OK]
[15-414] Benchmark formulas/count-5-2-unsat.cnf                  [OK]
[15-414] Benchmark formulas/count-5-3-unsat.cnf                  [OK]
[15-414] Benchmark formulas/count-6-2-sat.cnf                    [OK]
[15-414] Benchmark formulas/count-6-3-sat.cnf                    [OK]
[15-414] Benchmark formulas/count-7-2-unsat.cnf                  [OK]
[15-414] Benchmark formulas/count-8-2-sat.cnf                    [OK]
[15-414] Benchmark formulas/parity-3-unsat.cnf                   [OK]
[15-414] Benchmark formulas/parity-4-sat.cnf                     [OK]
[15-414] Benchmark formulas/parity-5-unsat.cnf                   [OK]
[15-414] Benchmark formulas/parity-6-sat.cnf                     [OK]
[15-414] Benchmark formulas/parity-7-unsat.cnf                   [OK]
[15-414] Benchmark formulas/parity-8-sat.cnf                     [OK]
[15-414] Benchmark formulas/php-3-2-unsat.cnf                    [OK]
[15-414] Benchmark formulas/php-4-3-unsat.cnf                    [OK]
[15-414] Benchmark formulas/php-4-4-sat.cnf                      [OK]
[15-414] Benchmark formulas/php-4-5-sat.cnf                      [OK]
[15-414] Benchmark formulas/php-5-4-unsat.cnf                    [OK]
[15-414] Benchmark formulas/php-5-5-sat.cnf                      [OK]
```

Figure 1: Output of running the script `'test.sh'`

You can use the script `'test.sh'` to run your SAT solver on all benchmarks in the `'formulas'` directory. This can be performed by executing this following command in the terminal:

```
$ ./test.sh
```

If the answer to all the benchmarks is correct, than you should get a similar output to the one presented in Figure 1.

Lab 4

For Part I of Lab 4, we will just check if your SAT solver implementation using BCP passes all tests. Even though you do not need to have a fully verified solution by Tuesday, 27th of November, we strongly encourage you to have as much as possible done. We will provide you some feedback on your specifications and the efficiency of your algorithm. For Part II of Lab 4, you will need to fully verify your implementation of BCP and implement one additional technique from the suggestions discussed in Lecture 20.