Task: 1
SAT solver

deadline: 12 June, 2020, 20:00

The objective of this assignment is to build a DPLL-style SAT solver based on depth-first search and the splitting rule.

Programming languages. The assignment can be solved using any modern programming language such as C, C++ (gcc), Java, Python, OCaml, Haskell... It is mandatory to provide a Makefile allowing the project to be automatically built and run in a modern computing environment. Typing "`make`" should result in an executable file called "`sat-solver`", which will be run by the test suite to score the solution. In the file `sat-solver.hs` we provide a simple skeleton of SAT solver written in Haskell correctly parsing the input and producing the output in the required format (but not the correct answer). This can be used as a starting point to write the solution.

Input. The solution program must read the standard input to parse the input to the problem. The input consists of a single line encoding a formula of propositional logic φ according to the following Backus-Naur grammar (c.f. `Formula.hs`):

$$\varphi, \psi :\equiv \mathsf{T} \mid \mathsf{F} \mid (\mathsf{Var}\ \texttt{"}\mathsf{string}\texttt{"}) \mid (\mathsf{Not}\ \varphi) \mid (\mathsf{And}\ \varphi\ \psi) \mid (\mathsf{Or}\ \varphi\ \psi) \mid (\mathsf{Implies}\ \varphi\ \psi) \mid (\mathsf{Iff}\ \varphi\ \psi)$$

where string is any sequence of alphanumeric characters not containing "`"`". An example of input is the following formula:

```
Iff (Var "r") (Iff (Var "q") (Or (Iff (Var "q") (Var "p")) (Iff (Var "q") (Iff (Var "r") F))))
```

See also the provided test files `./tests/test{00,01,...,09}.txt` for further examples of input formulas.

Output. The solution program must produce in the standard output the correct answer to the satisfiability problem for the formula given in input. The output consists of a single line which is either `0` if the input formula is not satisfiable, or `1` if the input formula is satisfiable.

Testing. The script `./check.sh` can be executed to test the solver against 10 provided examples of satisfiable and unsatisfiable formulas (five each). This can be used to make sure that the solution program correctly handles the input/output specification above and that it satisfies a minimum level of program correctness. We *strongly encourage* running the test suite prior to the solution submission. Solutions not conforming to the input/output specification will not be scored and will be discarded.

Submission. The submission is done on GitHub Classroom. After the deadline, GitHub Classroom will save the latest commit, which will then be scored.

Score. After submission, the solution program will be tested against 10 additional satisfiable and unsatisfiable formulas (five each). Therefore, there will be 20 tests in total. Each test will be run with a timeout of 60 seconds. A score will be computed as follows:

$$\mathsf{score} = \mathsf{correct} - \mathsf{incorrect},$$

where "correct" is the number of correct answers computed within the timeout and "incorrect" is the number of incorrect answers computed within the timeout. Therefore, the minimum score is -20 and the maximum score is 20.