Prof. Dr.-Ing. Klaus Wehrle

COM SYS Communication and Distributed Systems | RWTH AACHEN UNIVERSITY

Programming Task

Deadline: 2022-01-17 10:00

1 Background

1.1 SAT Theory

SAT is a well known NP-complete problem from theoretical computer science. It is generally defined as a propositional logic over a set of boolean variables, joined with the three operators:

- $\wedge$ (conjunction)
- $\vee$ (disjunction)
- $\neg$ (negation)

In general a SAT formula can be composed in an arbitrary way with these three operators and any amount of variables. For example: $A = (u \wedge (v \vee \neg w) \wedge x \wedge (y \vee \neg z)) \vee \neg (z \vee \neg x)$. Assigning truth values to the variables u through z will evaluate the formula to either true or false. If any such assignment exists such that the formula evaluates to true, the formula is said to be satisfiable. If all valuations yield false, the formula is unsatisfiable.

The SAT problem is to determine whether a given formula is satisfiable.

However, typically the problem is restricted to 3-SAT, which consists of a formula in 3-CNF (conjunctive normal form): A set of conjoined clauses (i.e. “and”ed), each clause consisting of exactly three disjoint (i.e. “or”ed), possibly negated, variables. For example: $B = (u \vee v \vee \neg w) \wedge (v \vee \neg u \vee y) \wedge (\neg v \vee \neg z \vee y)$.

In contrast, we present Binary-SAT: An arbitrary formula consisting of any nesting of negations, conjunctions, and disjunctions. The latter operators, however, accepting exactly two arguments, for example: $C = (u \vee v) \vee (\neg w \wedge (v \vee \neg (u \vee y)))$.

Formally, we define Binary-SAT recursively:

- Every boolean variable v is an expression
- If e is an expression so is $\neg e$
- If e_1 and e_2 are expressions, so are $e_1 \vee e_2$ and $e_1 \wedge e_2$

1.2 Normal Polish Notation (NPN)

Instead of the infix notation $(a \times b)$, operations can be noted in NPN as $\times ab$. The semantics are equivalent, but parentheses are unnecessary, as the representation is unambiguous if the number of arguments to an operator is fixed.

For clarifications:
Julian Büning <julian.buening@comsys.rwth-aachen.de>

2 Write a SAT solver

We will write a solver for NPN denoted Binary-SAT queries that are passed on the command line. We will denote conjunctions as `&`, disjunctions as `|` and negations as `!`. Variables are from `[a-zA-Z]`. For instance the Binary-SAT formula C above would be denoted as `||uv&!w|v!|uy`.

Write a program that can be invoked as follows:

```
$ ./sat '||uv&!w|v!|uy'
```

Decide whether the query is satisfiable or unsatisfiable. Reject any invalid query.

- In case the query is unsatisfiable, exit with a corresponding message.
- If the query is satisfiable, print a corresponding message and output a satisfying model, i.e. a valuation for all occurring variables that satisfied the given query. In the given query, for instance, a valuation of 0 to all variables is a valid output.

It is fully acceptable to turn in a brute force solution, nobody expects the development of a state-of-the-art SAT solver.
Of course, a fast solver is more likely to impress...

Solve this task using the C++ programming language. We will compile your test program with something like the following:

```
$ g++-11 -std=c++17 -O3 -flto *.cpp -Wall -Wextra -pedantic -Wnull-dereference \
-Wfloat-equal -Wundef -Wcast-qual -Wcast-align -Wodr -Wold-style-cast \
-Wmissing-include-dirs -Wswitch-enum -Wswitch-bool -Wmissing-declarations \
-Wimplicit-fallthrough=5 -Wdouble-promotion -Wstrict-overflow=5 -Wformat=2 \
-Wformat-overflow=2
```

For clarifications:
Julian Büning <julian.buening@comsys.rwth-aachen.de>