

Lab classes - Week 10

In the lab-classes this week we experiment with *SAT solvers*.

Update the environment: In your existing directory CS-242-Algorithms:

```
git checkout master
git pull git://github.com/OKullmann/CS-242-Algorithms.git master
```

Or if this directory (repository) doesn't exist (anymore), resp. you want to create a new repository, create a new clone by

```
git clone git://github.com/OKullmann/CS-242-Algorithms.git
```

In case you created a new clone, by (just) `git pull` in the future you will pull from OKullmann's Github repository (by default).

Now change to the subdirectory for week 10:

```
cd 200910/Week10/
```

Basic setup:

1. This time we compile a SAT-solver source-code package. You might want to have a look into it, but this would take (quite) some time.
2. We create small examples of SAT problems and run the solver.

Compilation is done by

```
Week10> make
```

Just one executable is produced:

```
minisat2
```

Comments on the code

1. `minisat2` is quite a well-known SAT solver, a so-called "conflict-driven clause-learning SAT-solver".
2. In a sense this class of algorithms combines the backtracking algorithm with dynamic programming.
3. <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/> is the project homepage.

Input format As variables you have to use the natural numbers $1, 2, \dots$, while the negated variables just use arithmetic negation. For example, the input formula

$$(a \vee b) \wedge (\neg a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

uses then the variables $a \mapsto 1, b \mapsto 2$, which results in the clause-set

$$\left\{ \{1, 2\}, \{-1, 2\}, \{1, -2\}, \{-1, -2\} \right\}.$$

Now the (standard) input-format of a SAT-solver, the *DIMACS format*, adds a header line showing the maximal index of a variable and the number of clauses, and finishes a clause with “0”. So the above clause-set yields

```
p cnf 2 4
1 2 0
-1 2 0
1 -2 0
-1 -2 0
```

You can either store this in a file (by the way, the DIMACS format also allows leading comment lines, starting with “c”), and then use `./minisat2 file`. Alternatively, `minisat2` also reads from standard input (recall that then your input is completed by `Ctrl-D`); however likely it’s best you use files (so it is easier to re-use inputs).

First examples

1. Determine (un)satisfiability of the above example, and also run `minisat2` on it.
2. Translate the three examples from the script (the casting-example, the colouring examples, and the example used to illustrate the simplest backtracking algorithm) into the Dimacs format, and then run the solver on it. Understand the results (that is, satisfiability or unsatisfiability, not the various statistics).
3. Show the resulting four Dimacs-files to the postgrads.

Further examples

1. Determine (un)satisfiability of

$$F := \left\{ \{c, b, \bar{e}\}, \{\bar{c}, d, a\}, \{\bar{b}, g\}, \{\bar{d}, f\}, \{c, \bar{b}, \bar{g}\}, \{\bar{c}, \bar{d}, \bar{f}\}, \{\bar{d}, f\}, \{d, \bar{a}\}, \{b, e\} \right\}$$

by hand, and then run the solver on it.

2. We said that the chromatic number of K_3 (the triangle) is 3. Verify this with the SAT solver, that is, translate the 2-colourability and the 3-colourability problem for this graph into SAT problems, and run the solver on them.

Here you have to adopt the example given in the script appropriately.

3. Show your notes and the resulting three files to the postgrads.