IN3070
IN4070

Autumn 2023

Exercises for the Course

Logic

Obligatory Exercises 3

Deadline: 1.11.2022, 23:59

Exercise O3.1

(Program a SAT solver)

In your favourite programming language, write a program that checks the satisfiability of a propositional formula in clause form.

The program should indicate if the clause set is satisfiable or not, and if it is satisfiable, give a satisfying interpretation.

IN3070: You may use any of the calculi we covered in the course. (But don't just build a truth table!)

IN4070: You should use a calculus that uses some form of simplification or unit propagation as discussed in the DPLL lecture.

Check you program on the clause set consisting of all 8 clauses you can build from 3 propositional variables.

$$\{\{p, q, r\},\ \{p, q, \neg r\},\ \{p, \neg q, r\}, \ldots\}$$

This clause set should be identified as unsatisfiable. If you delete one of the clauses, the resulting set of 7 clauses is satisfiable.

See what happens with the 2^n clauses from n propositional variables for $n > 3$. How large n can your program cope with?

Please deliver your program, as well as a PDF explaining your code and your results.

Hints:

- *Building up a whole proof tree will be complicated and slow and may take a lot of memory. Try to write your program so it works on one branch at a time. Only keep track of the leaf sequent you are working on. One way of doing this is using recursion. Here's some pseudo code:*

```
Result prove(Sequent s) {
   if (s is axiom) {
      return "unsatisfiable"
   else if (no more rule applications possible on s) {
      return literals in s as satisfying interpretation
   }
   else {
      pick a possible rule application
      List<Sequent> prems = premisses from that rule application
      for p in prems {
         answer = prove(s)
```

```
            if (answer is a satisfying interpretation I) {
               return I
            }
         }
         // the proofs for all premisses were closed, so...
         return "unsatisfiable";
      }
   }
```

It may be easier to keep the literals and the remaining clauses separate in the sequent `s`*, i.e. pass around two arguments.*

- *You can represent a clause set as a list of lists or an array of arrays, etc., depending on what is most natural for the programming language you choose.*
- *For the literals you can use* `p` *and* `not(p)` *in Prolog, but you could also use integers, so that* 1 *is a propositional varible and* -1 *its negation. Take care not to use* 0 *in that case…Again, it's up to you what is easiest in your programming language.*
- *Normal resolution won't easily give you a satisfying interpretation when it fails, so don't base your program on that.*
- ***And please:*** *We need to understand what your program does. So please add enough documentation and use sensible function/method/predicate/variable names.*