INF3170 / INF4171
Oblig 1

Deadline: 14 October 2016, 23:59

Delivery

This course uses Devilry (devilry.ifi.uio.no) for handing in solutions. Hand in your solutions in the form of a single PDF or plaintext (.txt) file with answers to problems 1 and 2, along with a .zip or .tgz archive with source code (and possibly documentation) for problem 3.

Problem 1: Natural Deduction

Give Natural Deduction proofs or deductions for the following:

a) $\vdash P \to (Q \to P)$.

b) $\vdash P \to \neg\neg P$.

c) $\neg\neg P \vdash P$.

d) $A \to (B \to C) \vdash (A \land B) \to C$.

e) $A \to C, B \to C \vdash (A \lor B) \to C$.

Problem 2: Sequent Calculus

Use the (classical) sequent calculus to give proofs of the sequents from problem 1. Also, use sequent calculus to find counter models to the following sequents:

f) $\vdash P \to (P \to Q)$.

g) $A \wedge B \vdash A \wedge C$.

h) $A \vee B \vdash A \vee C$.

Problem 3: Implement a Theorem Prover

In this problem you are to implement the sequent calculus for formulas containing $\wedge$ and $\neg$. You can use any programming language you want, but you may not be able to get help if you chose an esoteric programming language.

Your program only needs to output whether the input sequent is provable or not, but bonus point will be awarded if you provide counter models for non-provable sequents.

Input format

The input will be fed to the Standard Input (`stdin`), and must be strings from the following set IN:

- All lower case letter are in IN, i.e., $a, b, c, \ldots, x, y, z \in IN$.
- If $s, t \in IN$, then $-s \in IN$ and $\&st \in IN$.

The strings are interpreted in the following way:

- Any string in IN is a formula.
- $-s$ is the negation of the formula s.
- $\&st$ is the conjunction of s and t.
- Whitespace is not interpreted, and can be added for readability.

For example, the formula $P \vee Q$ must first be rewritten as $\neg(\neg P \wedge \neg Q)$. This is then represented as `-&-p-q`. A more readable form is `- & -p -q`.

The notation used for the input is commonly knowned as prefix notation or polish notation.

Problem 3+: Implement full Sequent Calculus (optional)

Same as problem 3, but with all the rules of sequent calculus. Use | (ASCII pipe) for $\vee$ and $>$ (ASCII greater than) for $\rightarrow$.

Suggestions and hints

Testing material

For testing, you can use the sequents from problem 1 and 2. The formulas must be converted into a form with only $\wedge$ and $\neg$. You can use the following rules:

$$\begin{aligned} A \vee B &\quad\Leftrightarrow\quad \neg(\neg A \wedge \neg B) \\ A \rightarrow B &\quad\Leftrightarrow\quad \neg(A \wedge \neg B). \end{aligned}$$

Data structure

The algorithm or data structure should have a recursive nature, since you may have to split the proof tree.

If you want to avoid recursion, you can alternatively keep a dynamic array of proof leaf nodes. Since we never do backtracking, there is no need to keep sequents that are not leaves. If a leaf is an axiom, it can be deleted from the array. Thus the array contains all active searches for a counter model. Should the array be empty, there is no counter model.

Interpretation as a recursive function

Note that we can define $F : IN \rightarrow FORM$ as

- $f(x) = x$ for lower case letters. x is an atomic formula.
- $f(-s) = \neg f(s)$.
- $f(\&st) = f(s) \wedge f(t)$.