

Computational Logic
Assignment 3: programming
Due: **13 April 2010**, 1.30 PM

Send your solution to `martinen@elet.polimi.it` in one Prolog file named `lastnames.pl`, using max. 4 letters per last name. The file must contain your full name(s) as a comment in the first line. Aim at *correctness* and *compactness*.

Consider the calculus for propositional logic by Jan Łukasiewicz consisting of the Modus Ponens rule (MP) and the three following axiom schemata, where P, Q, R range over propositional formulas (with only \neg and \Rightarrow):

$ax_1: P \Rightarrow (Q \Rightarrow P)$

$ax_2: (P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$

$ax_3: (\neg P \Rightarrow \neg Q) \Rightarrow (Q \Rightarrow P)$

Part I - maximum 4 clauses

Implement in Prolog a predicate `prove/2` that takes as input a formula F and outputs a proof for F in the calculus, if F is a theorem.

F should be represented in prefix notation. Example: represent $((\neg p) \Rightarrow q)$ as the term `imp(not(p),q)`. No parser is required.

The output proof is a term that represents all proof steps. For instance, an application of MP that concludes q from p and $p \Rightarrow q$ is indicated as `mp(q,p,imp(p,q))`; the application of axiom ax_1 $p \Rightarrow (q \Rightarrow p)$ is indicated as `ax1(imp(p,imp(q,p)))`. Similarly for axioms ax_2 and ax_3 .

Example: the goal `?- prove(imp(q,imp(p,imp(q,p))),Proof)` should return an answer similar to the following one

```
Proof = mp(imp(q, imp(p, imp(q, p))), ax1(imp(p, imp(q, p))), ax1(imp(imp(p,
imp(q, p)), imp(q, imp(p, imp(q, p)))))
```

Part II - maximum 6 clauses

Implement now `prove/3`, in which the third argument reports the number of applications of MP in the proof. Represent the number *à la* Peano, i.e., zero is 0 and the successor of n is $s(n)$. For instance, 3 is `s(s(s(0)))`.

Part III - maximum 3 clauses

You may have noticed that your program may loop trying to apply MP indefinitely, especially when asking for more solutions. Define now `prfid/2`, with the same signature as `prove/2`, that uses a *first iterative deepening* strategy on the number of applications of MP: first try to find a proof with 0 MP applications; on failure, try with 1; then with 2; and so on.

Remarks:

- There is no need to use the cut operator `!` to implement your solution.
- On certain queries, the Prolog system may loop instead of returning an answer; do not expect to be able to prove complex theorems with the `prove` or `prfid` predicates, which certainly are not designed for efficiency.
- Some of the answers returned by the Prolog system may be incorrect due to the unsound implementation of unification in Prolog, that unifies, e.g., `X` with `f(X)`. Of course, you are not expected to correct this problem.
- You can implement your solution under any Prolog system you like (SWI, Eclipse, or Ciao).