Constructive Logic (15-317), Fall 2016
Assignment 6: Logic Programming and Quantifiers

Contact: Evan Cavallo (ecavallo@cs.cmu.edu)

Due Tuesday, October 25, 2016

This assignment is due at the beginning of class on the above date and must be submitted electronically via autolab. Submit your homework as a **tar** archive containing a **.kyt** file for each KeYmaeraI task and a **hw6.pl** file with your solutions to the Prolog tasks. There is no written portion of this assigment, so you do not have to submit a pdf file.

After submitting via autolab, please check the submission's contents to ensure it contains what you expect. No points can be given to a submission that isn't there.

1 Quantifiers

In this section, you will use KeYmaeraI to prove theorems with quantifiers in the sequent calculus. For each specification `foo.kyx`, submit your solution tactic as `foo.kyt`.

Task 1 (4 points)**.** `uniform.kyx`:

$$(\exists y \forall x P(x, y)) \supset (\forall x \exists y P(x, y))$$

Task 2 (4 points)**.** `per.kyx`:

$$\begin{aligned} & (\forall x \forall y P(x, y) \supset P(y, x)) \\ \supset \; & (\forall x \forall y \forall z (P(x, y) \land P(y, z) \supset P(x, z))) \\ \supset \; & (\forall u \forall v (P(u, v) \supset P(u, u) \land P(v, v))) \end{aligned}$$

Task 3 (4 points). `russell.kyx`

$$(\exists s \forall x (E(x,s) \Leftrightarrow \neg E(x,x))) \supset \bot$$

(If you read the terms as sets and the predicate $E(x,y)$ as $x \in y$, this is the statement that there is no "Russell set" $\{x \mid x \notin x\}$.)

Task 4 (4 points). `yoneda.kyx`

$$\forall x \forall y ((P(x) \supset P(y)) \Leftrightarrow \forall z ((P(y) \supset P(z)) \supset (P(x) \supset P(z))))$$

2 Prolog Programming

For this part of the assignment, you will write and test some simple Prolog predicates, adhering to the following standards:

- Don't use cut, conditional, or any other control operator.
- Don't use negation-as-failure or disequality.
- You may use any of the list processing predicates or arithmetic predicates provided by gprolog, except `maplist` or any of the sorting predicates.
- Backtracking must not result in nontermination or in wrong answers. That is, when the user types `;` in response to a query result, execution must terminate with a correct answer or `no`.
- Backtracking may produce redundant correct answers.
- When your code is loaded and compiled, no warnings should be printed.

You can run Prolog on an Andrew machine via

```
/afs/andrew/course/15/317/bin/runprolog
```

Alternatively, you can download and install a copy locally from `http://www.gprolog.org`. You can load a file `foo.pl` at the Prolog prompt by typing

```
?- [foo].
```

Issue queries by typing predicates at the prompt. If Prolog offers more solutions, you can see them by typing `;` and ignore them by pressing enter.

In your submission for the following tasks, you can (and in some cases should!) define helper predicates.

2.1 Mergesort

Task 5 (12 points)**.** In this task, we'll implement a predicate which can be used to perform mergesort. Let `L1@L2` indicate the concatenation of the lists `L1` and `L2`.

1. Implement a predicate `split(L,L1,L2)` which holds exactly when `L1` and `L2` *evenly partition* `L`, that is, when `L1@L2` is a permutation of `L`, and `L1` and `L2` differ in length by at most one.

2. Implement a predicate `merge(L1,L2,L)` for sorted lists of integers `L1`, `L2`, and `L`, which holds exactly when `L` is a sorted permutation of `L1@L2`.

3. Implement a predicate `mergesort(L1,L2)` operating over two lists of integers. Your predicate should use the aforementioned primitives to implement mergesort; `mergesort(L1,L2)` should hold exactly when `L2` is a sorted permutation of `L1`.

2.2 *n*-rooks

Task 6 (12 points)**.** Implement a predicate `nrooks(A)` which holds for a two-dimensional list `A` when `A` contains exactly one `rook` in each row and column, with all other entries `empty`. For example, your solution should respond to the following queries like so, modulo order of responses:

```
| ?- nrooks([[empty,rook],[rook,empty]]).

true ? ;

no
| ?- nrooks([[A00,A01],[A10,A11]]).

A00 = empty
```

```
A01 = rook
A10 = rook
A11 = empty ? ;

A00 = rook
A01 = empty
A10 = empty
A11 = rook ? ;

no
| ?- nrooks([[A,B],[B,empty]]).

A = empty
B = rook ? ;

no
| ?- nrooks([[A,B],[A,empty]]).

no
```