Constructive Logic (15-317), Fall 2015
Assignment 9: Forward-looking **Pro**log

Contact: Vincent Huang (vincenth@andrew.cmu.edu)
Recitation FILL ME IN

Due Tuesday, November 24, 2015

Think you're getting good at Prolog? Ha-ha! Try this!

Think you're really good at Prolog now? Ha-ha! Prolog doesn't do forward reasoning. . .

The Prolog portion of your work should be submitted electronically by placing `hw09.pl` in your handin directory (`/afs/andrew/course/15/317/submit/andrewid/hw9`).

The written portion of your work should be submitted at the beginning of class. If you are familiar with LaTeX, you are encouraged to use this document as a template for typesetting your solutions, but you may alternatively write your solutions *neatly* by hand.

1 Type inference (25 points)

We can implement symbolic algorithms such as type checking and evaluation in Prolog almost as easily as we can specify them on paper. Consider the proof term assignment for natural deduction that we have been using all semester. In Prolog, we could specify the syntax of terms as follows:

```
% Infix notation
:- op(840, xfy, =>).   % implies, right assoc
:- op(830, xfy, \/).   % or, right assoc
:- op(820, xfy, /\).   % and, right assoc
:- op(800, fy, :).     % has-type, prefix
:- op(800, fy, ?).     % variable, prefix
```

Figure 1: Prolog starter code for type inference.

```
term(?X).
term(unit).
term(lam(?X,M))
  :- term(M).
term(app(M,N))
  :- term(M), term(N).
term(pair(M,N))
  :- term(M), term(N).
term(fst(M))
  :- term(M).
term(snd(M))
  :- term(M).
term(inl(M))
  :- term(M).
term(inr(M))
  :- term(M).
term(case(M,?X,N,?Y,P))
  :- term(M), term(N), term(P).
```

Here we use `?x` to represent a variable x, `lam(?x.M)` to represent the term $\lambda x.\ M$, and `case(M,?x,N,?y,P)` to represent case($M, x.\ N, y.\ P$). We represent conjunction by /\, disjunction by \/, implication by =>, and truth by `top`.

When we looked at proof terms in class, we annotated lambdas by the type of the bound variable; in practice, we can actually infer possible types for the variable by looking at the rest of the term. The term $\lambda x.\ (\pi_1 x)$ is encoded `lam(?x, fst(?x))`. It must be assigned a type which unifies with `(A /\ B) => A`; that is, it could have type `(A /\ B) => A`, but it would also validly typecheck as `(tt /\ tt) => tt`.

Task 1 (20 points)**.** Implement a predicate `infer(G, M, A)` that holds whenever term `M` has type `A` under context `G`. (Hint: Make sure your implementation is robust with respect to alpha-renaming!)

The file hw09.pl contains some infix operator declarations relevant to the assignment (see Figure 1).

Task 2 (5 points)**.** Given a term `M` with no free variables, how may we invoke `infer` to infer the type of `M`? Use this to infer the type of $\lambda x.\ x\ x$, and explain the behaviour you see in terms of what happens in Prologs proof search. (Put your answers in a comment.)

2 Gattai! (4 points)

Find the most general unifier of the following terms, or explain why it is not possible to find one.

Task 3 (2 points). $f(x, x, g(x))$ and $f(y, h(z, a), w)$

Task 4 (2 points). $f(x, x, g(x))$ and $f(y, h(z, a), z)$

3 Towering over the river (11 points)

You are probably familiar with the Tower of Hanoi puzzle. Recall that all disks start on a single peg with the smallest on top, and the goal is to move them all to a different peg. Along the way, you can move only the topmost disk of a peg, and you cannot place a larger disk on top of a smaller disk.

You might be familiar with the standard recursive solution to the puzzle. You might be less familiar with an iterative solution to the problem, but at least one exists (see figure 2).

1. Number the pegs 1 – 3. Treat them like they're in a circle (so if you're moving a disk clockwise from peg 3, you move it to peg 1).
2. Number the disks $1 - n$. Assign a disk the clockwise direction if its number is even, and counterclockwise if its number is odd.
3. At every point, there should be only one disk that can be legally moved. Move the disk in accordance with the above. Do not move a disk that has just been moved.
4. Stop when there are no legal moves.

Figure 2: Iterative algorithm to solve the Tower of Hanoi puzzle

Task 5 (6 points). Write predicates and rules that allow you to represent a Tower of Hanoi puzzle, and, when executed as a forward logic program, implement the iterative algorithm stated in figure 2.

Task 6 (2 points). Represent the starting state where there are 3 disks on peg 1 using the rules and predicates you gave in task 5.

Task 7 (3 points). Now apply your rules on the start state you gave in task 6 to saturation, showing *every step*. Read off the Tower of Hanoi solution you get and write that down in plain English in a separate paragraph.