CS 164 Programming Languages and Compilers **Spring 2019**

Discussion Week of 3/18: Prolog and Type Inference

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

The goal of this section is to expose you to logic programming and type inference.

1 Prime Numbers and List Reversal in Prolog

1. **Prime Numbers.** We'll start by looking at prime numbers. Write a function `prime(X)` in Prolog that takes a number `X` and returns true if it is prime and false if not. You may find the starter code file `primes.pl` on the course website useful.
2. **List Reversal.** Write a list reversal predicate in Prolog. We are intentionally not giving you the function signature. Come up with your own.

2 Type System for a Toy Language

Recall the `typeof` predicate from lecture.

```
defn(I, T, [def(I, T) | _]).
defn(I, T, [def(I1, _) | R]) :- dif(I, I1), defn(I, T, R).
typeof(X, T, Env) :- defn(X, T, Env).
```

1. Translate these Prolog lines into English. Be precise.

Now we will gradually build up a subset of the following grammar with typing rules. Then we'll experiment with type inference with this grammar.

```
e :  ID | INT
  | [ ((e ,)* e)?  ] /* list */
  | lambda ( ID , e )
  | e '+' e
  | e '<<' e
```

```
| e '//' e
| cast(e,e)
```

Begin with two typing rules. The starter code is also available online.

```
typeof(X, int, _) :- integer(X).
typeof(X, T, Env) :- defn(X, T, Env).
```

2. Write a typing rule such that the type of an empty list is unbounded (e.g `[_]`).
3. Write a typing rule such that the type of a list is `[T]`, where all list elements are of type `T`.
4. Write a typing rule such that the type of a lambda is `T1->T2`, where `T2` is the return type and `T1` is the type `X` is bound to within the body of the lambda.
5. Write a typing rule such that the type of `L + R` is `int` when `L` and `R` are both of type `int`.
6. Write a typing rule such that the type of `L // R` is `[T]` when `L` and `R` are both of type `[T]`.
7. Write a typing rule such that the type of `L << R` is `T2` when `L` is of type `T1 -> T2` and `R` is of type `T1`.
8. Write a typing rule such that the type of `cast(L,R)` is `T1 -> T2` when `L` is of type `T1` and R is of type `T2`.

   At this point we could enter our rules into Prolog and have it infer types for our programs. For the next few questions, try solving for `T` manually.

   (a) `typeof(f << g, T, [def(f, int->[int]), def(g, int)]).`

   (b) `typeof(lambda(x, x // x) << [1], T, []).`

   (c) `typeof(lambda(x, x + x) << [1], T, []).`

   (d) `typeof(lambda(x, x + x) << 1, T, []).`

   (e) `typeof(lambda(x, x + x) + 1, T, []).`

   (f) `typeof(lambda(x, x // x) << [lambda(x, x // x)], T, []).`

   (g) `typeof(lambda(x,lambda(y, cast(x,y))) << [lambda(x,lambda(y, cast(x,y)))], T, []).`