CS334 HW 9: Judgments

Due Thursday, April 23, at 9:55 am

Remember to write down how long the assignment takes you to complete

1 Type Judgments (25 points)

Consider the following small language:

$$
\begin{aligned}
\mathit{type} &::= \texttt{bool} \mid \texttt{num} \mid \mathit{type} \rightarrow \mathit{type} && (1)\\
\mathit{lambda} &::= \texttt{(}\ \lambda\ \texttt{(}\,\mathit{id}\ \texttt{:}\ \mathit{type}\ \texttt{)}\ \texttt{:}\ \mathit{type}\ \mathit{exp}\ \texttt{)} && (2)\\
\mathit{app} &::= \texttt{(}\ \mathit{exp}\ \mathit{exp}\ \texttt{)} && (3)\\
\mathit{exp} &::= \mathit{lambda} \mid \mathit{app} \mid \mathit{bool} \mid \mathit{num} && (4)
\end{aligned}
$$

1. If there was a `sqrt` procedure defined in the library, what would its type be?

2. Give type judgements for all expressions in the language, e.g., T-num, T-app, etc.

3. Assume that `even?` is defined and has type *num* → *bool*. Prove that the expression `((λ (x:num) (even?  x)) 7)` has type *bool*.

4. Assume that two-argument + is a procedure in the standard library of this language, which only allows single-argument procedures. What must the type of + be? (This is a puzzle.)

5. Note that *type* is not a subset of *exp*. Why might it be a bad idea to extend *exp* to include *type*? (This is a puzzle.)

6. I've heard people casually mention in class that "Scheme has no types." That is not true. Explain how types must work in Scheme, based on your knowledge of the language, type judgments, and experimentation with Dr. Scheme.

2 Language Design (25 points)

Read Steele, "Growing a Language", *HOSC* 1999. Steele was a key designer in both Scheme and Java (among other languages). You're experienced with both of those. Write an essay of length less than one page that covers the following:

1. The main point of the paper.

2. How does the policy advocated in this paper apply to Scheme and Java?

3. Does the paper gives good advice? Why/not?

4. Either how you can generalize the advice to design of programs (not just languages), or what a better policy would be (for language or program design).

3 Challenge

If we remove num and bool from the above grammar, we have a typed lambda calculus. In that version of lambda calculus, what is the type of a Church boolean? What is the type of `NOT`, defined on Church booleans? (These are *really* challenging to answer based only on Thursday's lecture–if it takes you less than five minutes to answer, you probably haven't thought about it carefully enough.)