Name:

CS 421 — Type Semantics Activity (Monotype Version)

Mattox Beckman

The Rules

0.1 The Language

L	::=	$\lambda x.L$	abstractions
	|	$L\ L$	applications
	|	`let` $x = L$ `in` L	Let expressions
	|	`if` L `then` L `else` L	If expressions
	|	E	expressions
E	::=	x	variables
	|	n	integers
	|	b	booleans
	|	$E \oplus E$	integer operations
	|	$E \sim E$	integer comparisons
	|	$E \,\&\&\, E$	boolean and
	|	$E \,\|\|\, E$	boolean or

0.2 The Type Rules

Arithmetic $$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \oplus e_2 : \texttt{int}}$$

Relations $$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \sim e_2 : \texttt{bool}}$$

Booleans $$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \texttt{and}\ e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \texttt{or}\ e_2 : \texttt{bool}}$$

If $$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \alpha \quad \Gamma \vdash e_3 : \alpha}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \alpha}$$

Application $$\frac{\Gamma \vdash e_1 : \alpha_2 \to \alpha \quad \Gamma \vdash e_2 : \alpha_2}{\Gamma \vdash e_1\ e_2\ : \alpha}$$

Abstraction $$\frac{\Gamma \cup \{x : \alpha_1\} \vdash e : \alpha_2}{\Gamma \vdash \lambda x.e : \alpha_1 \to \alpha_2}$$

Let $$\frac{\Gamma \vdash e_1 : \alpha_1 \quad \Gamma \cup \{x : \alpha_1\} \vdash e_2 : \alpha_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2\ : \alpha_2}$$

Reductions

Reduce the following programs according to the semantic rules given.

Problem 1)

$\{\texttt{x:Int,y:Int}\} \vdash \texttt{if}\ x * y > 2\ \texttt{then}\ x\ \texttt{else}\ y\ :\ Int$

Problem 2)

$\{\texttt{x:Int,y:Int}\} \vdash \texttt{let}\ m = x * y\ \texttt{in}\ m - x\ :\ Int$

Problem 3)

$\{\} \vdash (\lambda f.\lambda x.f\ x)\ (\lambda x.x)\ 10\ :\ Int$

Make your own rules!

Problem 4)

Try to write the type rules for HASKELL's `head` and `tail` functions.

Problem 5)

The logical rule for *Modus Pones* looks like this:

$$\frac{A \to B \quad A}{B}$$

Is there a programming language equivalent to this?[1] Talk to a neighbor and see if you can find a semantic rule that mirrors this.

Problem 6) What happens when you try to type check this code? Try to derive α.

`{y:Int,z:String}` $\vdash (\lambda f.(fy, fz))\ (\lambda x.x) : \alpha$

[1] Hint, the answer is "yes".