CMSC631 (Practice) Exam (v2)
Spring, 2014

Consider the following syntax of the programming language $\mathcal{PB}$ ("Peanut Butter"):

$$\begin{array}{llll} \textit{Exp} & e & ::= & \textit{True} \mid \textit{False} \\ & & \mid & \textit{Nil} \\ & & \mid & \textit{Pair}(e, e) \\ & & \mid & \textit{Proj}_L(e) \\ & & \mid & \textit{Proj}_R(e) \\ & & \mid & \textit{Cond}(e, e, e) \end{array}$$

The Peanut Butter language contains pairs, nil, and booleans. The behavior of $\mathcal{PB}$ programs is a bit quirky; we describe how $\mathcal{PB}$ programs work informally below:

- Values in $\mathcal{PB}$ include booleans, nil, and pairs of values (note: this is a recursive definition); pairs are constructed with *Pair*.
- Pair values are deconstructed with $\textit{Proj}_L$ and $\textit{Proj}_R$, which project out the left and right component of a pair, respectively. So for example, $\textit{Proj}_L(\textit{Pair}(e_1, e_2)) = e_1$. Applying $\textit{Proj}_L$ or $\textit{Proj}_R$ to non-pair values is an error.
- $\textit{Cond}(e_1, e_2, e_3)$ is a conditional form, which selects e_2 to evaluate whenever e_1 evaluates to a *truish* value, and selects e_2 to evaluate otherwise. A truish value is any value that is not *False*.

  So for example, $\textit{Cond}(\textit{False}, e_1, e_2) = e_2$, but $\textit{Cond}(\textit{Pair}(\textit{False}, \textit{False}), e_1, e_2) = e_1$.

Problem 1. *Give a formal definition of the set of values in $\mathcal{PB}$.* □

Problem 2. *Define a natural semantics for $\mathcal{PB}$. Show the derivation for evaluating the program:*

$$Proj_L(Cond(Pair(True, False), Pair(False, Nil), Pair(True, Nil)))$$

(Your semantics should only specify the "good" behavior of programs and doesn't need to bother with erroneous programs.) □

It turns out that even though $\mathcal{PB}$ only has pairs, nil, and booleans for values, $\mathcal{PB}$ programmers tend to think in terms of "lists". Lists are either empty or consist of an element paired together with another list. An empty list is represented by *Nil*. So for example, the list of three *True* values could be represented:

$$\mathit{Pair}(\mathit{True}, \mathit{Pair}(\mathit{True}, \mathit{Pair}(\mathit{True}, \mathit{Nil}))).$$

Moreover, $\mathcal{PB}$ programmers think in terms of *homogeneous* lists, i.e. lists of the same kinds of elements. So for example, a $\mathcal{PB}$ programmer thinks in terms of "a list of booleans" or "a list of lists of booleans," etc.

With that in mind we can formalize a notion of types for $\mathcal{PB}$:

$$\begin{array}{llcl} \mathit{Type} & t & ::= & \mathit{Bool} \\ & & \mid & \mathit{List}(t) \end{array}$$

Problem 3. *Define a type judgement relation for $\mathcal{PB}$ programs. Your type system should accept the program given in problem 2 as having type Bool. Give the type derivation for the program in problem 2. Give an example of a program that is ill-typed.* $\square$

After years of use, the $\mathcal{PB}$ language was replaced by it's successor $\mathcal{PB}\&\mathcal{J}$, which added the following features to $\mathcal{PB}$:

- Using the *J*(e) operator, programs could jump to end of evaluation, making the value of e the final result of the computation.
- Programs no longer consisted of single expressions e, but instead consist of any number function definitions followed by an expression that can make use of those definitions. Functions take a single argument and may be (mutually) recursive. Functions are *not* values in $\mathcal{PB}\&\mathcal{J}$.
- Projection operations were replaced by a pattern maching construct: *Let*(x, y, e_1, e_2) which evaluates e_1 to a pair then binds x to the left component and y to the right, within the scope of e_2.

Problem 4. *Give a formal definition of the syntax of $\mathcal{PB}\&\mathcal{J}$ programs.* □

Problem 5. *Define a small step reduction semantics for* $\mathcal{PB}\&\mathcal{J}$. $\square$