Assignment 3 Q&A

15-312 Foundations of Programming Languages
Kevin Watkins ⟨kw@cmu.edu⟩

February 16, 2005

Q1. Student B writes:

> I have a question about stream forcing. In a **case** statement:
>
> $$\mathbf{case}(\mathbf{cons}(x.e_1, x.e_2), e_3, y.z.e_4)$$
>
> [... solution details omitted ...] do I substitute [values] for y, z into e_4, or do I just substitute [non-values] straight into e_4?
>
> In assignment 1, we proved the substitution theorem for substituting expressions. Judging from the assignment 3 write-up, we can only use the value-substitution part of that theorem. So I would guess that you want [something] to be fully evaluated first, before substituting into the body of the **case** statement. [... details omitted ...] Am I right?

A1. See the correction below. This is a good question. Your guess about the value substitution theorem is right—even though we're adding some laziness for the recursive streams, we still want to keep the language as ML-like as we can, so that means we're only going to allow values to be substituted in for variables, not arbitrary expressions. (Lazy languages like Haskell usually do the opposite thing... an expression is never evaluated before being substituted.) This is usually what ML stream libraries (like the one I used in the code for Assignment 2) do, too... in Assignment 2, whenever a stream gets forced, the head and tail get computed down to values before they're returned to whoever did the forcing.

One hint I can give you is that you can invent any stack frames you want. All the stack frames we've seen so far look like an expression with a box somewhere in it, but stack frames don't always have to look that way.

Q2. Student B writes back:

> In class today, Prof. Crary added $\mathbf{cont}(k)$ to be a type of expression, even though $\mathbf{cont}(k)$ is not a expression one would write in MinML, but just something the C Machine to use to store intermediate values....
>
> For **case**, can I invent an intermediate expression $\mathbf{case}'$, and have **case** step into $\mathbf{case}'$ where the parts of the stream would then be evaluated (it would be the same as evaluating **let**)?

A2. Okay... so the first thing is, I led you astray with the last message, because I forgot that this assignment is about the M machine, not the C machine. So defining new stack frames is out.

The second thing is, I feel more comfortable throwing in new expressions just for the purposes of the machine, when the expressions are only in one type. The addition of $\mathbf{cont}(k)$ only adds to the set of expressions having type τ **cont**, not to **int** or **bool** or whatever. Throwing in new *values* not restricted to a single type would be particularly bad, because we'd be apt to break the canonical forms lemma. But your **case′** wouldn't be a value, I'm guessing, so that's not a problem.

That being said, in the M machine it looks like something has to give; either we add another sort of expression, like **case′**, or we reuse an expression like **let** for the purposes of **case**. The underlying problem being that in the M machine, the whole state of the machine at each moment in time has to be coded up as a single expression.

Given these two somewhat distasteful options, I think I like the **case′** idea better. But it wouldn't be *wrong* to use **let**, because the assignment doesn't say you're not allowed to do that.