Homework 7
Running Things

98-317: Hype for Types

Due: 13 March 2018 at 6:30 PM

1 Introduction

In lecture, we saw how a language is evaluated. Rules for dynamics define how one expression is either a value or transitions to another. Given a specification for a dynamics, we may complete an interpreter for a language.

This homework is divided into three parts: Required, Useful, and Fun. You will receive credit for this homework if you turn in something (not necessarily something working) for the "required" portion.

Code solutions to the problems in this assignment will be released next week.

Turning in the Homework Run the command

```
tar cf handin.tar dynamics.sml
```

inside your `hw7` directory, then submit `handin.tar` to Autolab.

2 Lambda++

Remember Lambda++, from a few weeks ago? We recently “shared” the language specification with the CMU [1] PoP group[2] at PLunch [3], and they gave us some great constructive feedback. [4]

One of the comments was that despite having a robust type system, Lambda++ has no meaningful way of being run. The Hype for Types staff was confused as to why Lambda++ should actually have any practical utility, but decided to give it a try anyway. As a refresher, this is the syntax of Lambda++, with explicit type annotations elided.

Type	τ	$::=$	α	type variable
			$\tau \to \tau$	arrow type
			$\tau + \tau$	sum type
			$\tau \times \tau$	product type
Expression	e	$::=$	x	variable
			$\texttt{fn}\ x \Rightarrow e$	lambda
			$e\ e$	application
			$\langle e, e \rangle$	pair
			$\#1\ e$	left projection
			$\#2\ e$	right projection
			$\texttt{INL}\ e$	left injection
			$\texttt{INR}\ e$	right injection
			$\texttt{case}\ e\ \texttt{of INL}\ x \Rightarrow e\ \texttt{\|}\ \texttt{INR}\ x \Rightarrow e$	case analysis

The syntax representation in Standard ML is the same as last week. We’ll start where we left off next week, with our type inference mechanism having inferred all the necessary types. Throughout this implementation, **you are free to assume that the type inference and checker have passed.**

2.1 Dynamics

Under intense pressure to bring Lambda++ up to professional scrutiny, the staff have written some of the rules for the dynamics of the language. However, they need your help to finish them. Their incomplete work is below.

[1] https://www.cmu.edu/

[2] http://www.cs.cmu.edu/Groups/pop/

[3] https://goo.gl/maps/L3bAxENYdTF2

[4] Hype for Types is not sponsored by or affiliated with the PoP group, except for our faculty advisor. The story, all names, characters, and incidents portrayed in this course are fictitious. No identification with actual persons (living or deceased), papers, projects, and languages is intended or should be inferred. No person or entity associated with this course received payment or anything of value, or entered into any agreement, in connection with the depiction of tobacco products. No animals were harmed in the making of this assignment.

$$\frac{}{\texttt{fn}\ x \Rightarrow e\ \mathsf{val}} \qquad \frac{e_1 \longmapsto e_1'}{e_1\ e_2 \longmapsto e_1'\ e_2} \qquad \frac{\texttt{/* TODO: vijay finish this */}}{e_1\ e_2 \longmapsto e_1\ e_2'}$$

$$\frac{e_2\ \mathsf{val}}{(\texttt{fn}\ x \Rightarrow e_1)\ e_2 \longmapsto [e_2/x]e_1}$$

$$\frac{\texttt{\# TODO: charles why}}{\#1\ \langle e_1, e_2\rangle \longmapsto e_1} \qquad \frac{\texttt{\# isn't this}}{\#2\ \langle e_1, e_2\rangle \longmapsto e_2} \qquad \frac{\texttt{\# finished yet?}}{\langle e_1, e_2\rangle\ \mathsf{val}}$$

$$\frac{e\ \mathsf{val}}{\texttt{INL}\ e\ \mathsf{val}} \qquad \frac{e\ \mathsf{val}}{\texttt{INR}\ e\ \mathsf{val}} \qquad \frac{e \longmapsto e'}{\texttt{INL}\ e \longmapsto \texttt{INL}\ e'} \qquad \frac{e \longmapsto e'}{\texttt{INR}\ e \longmapsto \texttt{INR}\ e'}$$

$$\frac{\texttt{<!-- Jeanne could you please help with this one -->}}{\texttt{case}\ e\ \texttt{of INL}\ x_1 \Rightarrow e_1\ \texttt{|}\ \texttt{INR}\ x_2 \Rightarrow e_2 \longmapsto \texttt{case}\ e'\ \texttt{of INL}\ x_1 \Rightarrow e_1\ \texttt{|}\ \texttt{INR}\ x_2 \Rightarrow e_2}$$

$$\frac{}{\texttt{case INL}\ e\ \texttt{of INL}\ x_1 \Rightarrow e_1\ \texttt{|}\ \texttt{INR}\ x_2 \Rightarrow e_2 \longmapsto [e/x_1]e_1}$$

$$\frac{}{\texttt{case (* wait chris *) of INL}\ x_1 \Rightarrow e_1\ \texttt{|}\ \texttt{INR}\ x_2 \Rightarrow e_2 \longmapsto \texttt{(* what goes here *)}}$$

As you can see, the staff were in a real hurry and only left notes for each other to finish the work. When cornered and interrogated about what work was remaining, they mentioned the following:

- "Eager function application. Anything else would be a kludge."
- "Um, the products are lazy."
- "Oh yeah, obviously the cases are exactly like SML. I refuse to do it like Java."
- "We're not French[5], so we evaluate left-to-right.'

Recall that *eager* semantics require that an operand be a value before it can be worked with further. By contrast, *lazy* semantics allow an operand to be operated on even before it is a value. In a lazy language, large chunks of unevaluated expressions can be manipulated before actually being computed.

Looks like we'll have to finish the dynamics for these slackers first.

3 Required

Required Task Complete the missing portions of the dynamics above. Use the completed rules as a guide as to what should be written in each spot. Your solution should have one full rule for each incomplete rule above, with all missing components filled in.

The resulting system must be deterministic. That is, there should never be any ambiguity about which rule to apply.

Hint: it's possible for some slots that they should actually be empty. Which ones?

[5] `https://caml.inria.fr`

4 Useful

The submission deadline for the ACM SIGHYPE 2018 conference is coming up, and the course staff wants to whip up a prototype they can demonstrate. They want the Lambda++ interpreter to have a fully functional dynamics so that we can finally run the language.

Useful Task In `dynamics.sml`, implement the function

```
eval : exp -> exp
```

which uses the given dynamics to evaluate its expression down to a value.

Hints:

- You may find it helpful to write a helper function `step : exp -> exp` which applies just one of the rules at a time. Then, the evaluator needs only to repeatedly step until you're done.
- You need a way to distinguish between non-values and values. Perhaps a datatype could serve this purpose?
- Remember that some of the constructs are lazy, and some are eager. You can't recursively evaluate every expression you see.
- We provide you a function that substitutes expressions for variables in other expressions. It will be very useful.
- Watch out for shadowing and capture. If you substitute e' for x in `fn` $x \Rightarrow e$, we would *not* expect instances of x within e to get substituted. Likewise, if you substitute x for y in `fn` $x \Rightarrow y$, we might end up with `fn` $x \Rightarrow x$, which would be absurd. The provided routine takes care of shadowing but not capture. During testing, we advise using new variable names often to avoid capture. There's a proper solution for capture, [6] but we won't be using it in this assignment.
- You can assume the typechecker succeeds, but if you have inexhaustive matches, feel free to raise `TypeError`.
- Functions are values.

[6] Look up "de Bruijn indices"

5 Fun

Upon arriving at SIGHYPE 2018, the authors are bombarded with Haskell enthusiasts who want to try out Lambda++. One comments that the language is not sufficiently lazy, and that they should adopt a Haskell-like *call-by-need* evaluation strategy. That is, the language should not evaluate a term until it is necessary, but when it becomes necessary, it should be evaluated at most once so that multiple references to that term do not waste time computing it more than once.

Fun Task Adopt Lambda++ evaluation to this *call-by-need* strategy. One popular method is saving partial computations as suspended computations (thunks), then when those thunks are evaluated, replacing them with the result. This transformation only works in a purely functional language with no side effects, of course. (Ironically, an efficient implementation might use a mutable table to store these computations. That's up to you!)