Takehome Final, Written Problems

CS 538, Spring 2020

May 4, 2020

In this problem, we will be adding basic exceptions to the core functional language we saw in the first half of the course. The core language is a lambda calculus with booleans, integers, and strings:

$$
\begin{aligned}
\text{Num } n &::= \mathbb{N} \\
\text{Str } s &::= \text{``letters''} \\
\text{Val } v &::= x \mid \lambda x.\ e \mid \mathit{true} \mid \mathit{false} \mid n \mid s \\
\text{Exp } e &::= x \mid \lambda x.\ e \mid e_1\ e_2 \mid \mathit{true} \mid \mathit{false} \mid n \mid s \mid \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \mid e_1 = e_2 \mid e_1 + e_2
\end{aligned}
$$

In words, we have the following components:

- Numbers n (0, 1, ...)
- Strings s (“foo”, “bar”, ...)
- Variables x
- Values v (42, “foo”, $\lambda x.\ x + 1$, ...)
- Expressions e ($x + 0$, 3×5, ...)

The language is eager (call-by-value), with the following operational semantics:

$$
\frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \frac{e_2 \to e_2'}{(\lambda x.\ e_1)\ e_2 \to (\lambda x.\ e_1)\ e_2'} \qquad \frac{}{(\lambda x.\ e)\ v \to e[x \mapsto v]}
$$

$$
\frac{e_1 \to e_1'}{\mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \to \mathit{if}\ e_1'\ \mathit{then}\ e_2\ \mathit{else}\ e_3} \qquad \frac{}{\mathit{if}\ \mathit{true}\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \to e_2} \qquad \frac{}{\mathit{if}\ \mathit{false}\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \to e_3}
$$

$$
\frac{e_1 \to e_1'}{e_1 = e_2 \to e_1' = e_2} \qquad \frac{e_2 \to e_2'}{v_1 = e_2 \to v_1 = e_2'} \qquad \frac{v_1 \text{ is equal to } v_2}{v_1 = v_2 \to \mathit{true}} \qquad \frac{v_1 \text{ is not equal to } v_2}{v_1 = v_2 \to \mathit{false}}
$$

$$
\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2} \qquad \frac{e_2 \to e_2'}{v_1 + e_2 \to v_1 + e_2'} \qquad \frac{n = n_1 \text{ plus } n_2}{n_1 + n_2 \to n}
$$

1 Raising exceptions

To raise exceptions, we add a new expression:

$$
\text{Exp } e ::= \mathit{raise}(s) \mid \cdots
$$

The expression $\mathit{raise}(s)$ is an exception, and the string s describes the kind of exception. If we hit an exception during evaluation, then we don't evaluate the rest of the expression—we just step to the exception. We can model this behavior using the following operational rules:

$$\frac{}{\mathit{raise}(s)\ e \to \mathit{raise}(s)} \qquad \frac{}{(\lambda x.\ e)\ \mathit{raise}(s) \to \mathit{raise}(s)} \qquad \frac{}{\mathit{if}\ \ \mathit{raise}(s)\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \to \mathit{raise}(s)}$$

$$\frac{}{\mathit{raise}(s) = e \to \mathit{raise}(s)} \qquad \frac{}{v = \mathit{raise}(s) \to \mathit{raise}(s)}$$

Answer the following questions.

1. Write down similar rules for raising exceptions from $e_1 + e_2$. Hint: you should have two new rules, almost the same as the new rules for equality.
2. Write a (lambda calculus) function that takes two arguments x and y. If x is equal to 42 then raise the exception $\mathit{raise}(\text{“1st 42”})$, otherwise if y is equal to 21 then raise the exception $\mathit{raise}(\text{“2nd 21”})$, otherwise return the sum of x and y.

2 Handling exceptions

Most languages with exceptions also have exception handlers, which are used to recover and run error-handling code when an exception is raised. We add the following expression to our language:

$$\text{Exp } e ::= \mathit{try}\ e_1\ \mathit{with}\ e_2 \mid \cdots$$

The idea is that e_1 is a piece of code that may raise an exception, and e_2 is a function that takes a string describing the exception, and then does something (handles it). If e_1 doesn't raise an exception, then e_2 is never run.

To model this behavior, we add the following step rules:

$$\frac{e_1 \to e_1'}{\mathit{try}\ e_1\ \mathit{with}\ e_2 \to \mathit{try}\ e_1'\ \mathit{with}\ e_2} \qquad \frac{}{\mathit{try}\ \mathit{raise}(s)\ \mathit{with}\ e \to e\ s} \qquad \frac{}{\mathit{try}\ v\ \mathit{with}\ e \to v}$$

We have carefully ensured that $\mathit{raise}(s)$ is *not* a regular value v, so the last two rules do not overlap.

Answer the following questions.

1. Suppose that *foo* is a piece of code. Write a program that runs *foo*, handles an exception if it is labeled “1st 42” by returning 0, otherwise re-raises the exception. (Your answer should include “*foo*” somewhere. You do not need to show how this program steps.)
2. Show how your previous answer steps when *foo* is equal to $(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \mathit{raise}(\text{“1st 42”})\ \mathit{else}\ \mathit{raise}(\text{“2nd 21”}))\ 0$.
3. Show how your previous answer steps when *foo* is equal to $(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \mathit{raise}(\text{“1st 42”})\ \mathit{else}\ \mathit{raise}(\text{“2nd 21”}))\ 1$.