Assignment 7
K Machines

15-814: Types and Programming Languages
Frank Pfenning & David M Kahn

Due Thursday, October 28, 2021
70 points

This assignment is due by 11:59pm on the above date and it must be submitted electronically as a PDF file on Gradescope.

1 Typing the K Machine

The properties of progress and preservation have been major themes throughout the course. This emphasis is no different in the setting of K machine. However, statements about K machine make use of slightly different judgments than before, as we saw in lecture. For example, we now use the symbol $\div$ to act similarly to the colon from our previous typing judgments, so that $s \div \sigma$ means that s is a state[1] in the K machine returning a final value of type σ.

Task 1 (5 points) Give the typing rules for $s \div \sigma$. You may refer to the judgment $K \div \tau \Rightarrow \sigma$ from Section L12.6.

Task 2 (5 points) Provide the transition rules and new form(s) of continuations for the unit type 1.

Task 3 (10 points) In analogy to the typing rules for functions in the K machine we gave in lecture, provide typing rules for continuations supporting the following types:

(i) Variadic sums $\sum_{i \in I}(i : \tau_i)$

(ii) Products $\tau_1 \times \tau_2$

(iii) Unit 1

The *composition rules* of the K machine are those with the property that if $k \bowtie e \mapsto k' \bowtie e'$ then $k(e) = k'(e')$. Here $\bowtie$ is a generic notation for either $\triangleright$ or $\triangleleft$. The *critical rules* of the K machine are those that are not composition rules.

Task 4 (25 points)

(i) Write out the statement of preservation for the K machine.

[1] Recall that a state in a K machine is of the form $k \triangleleft e$ or $k \triangleright e$.

(ii) State the form of its proof (that is, by induction or cases, and on which judgment).

(iii) Give one case in the proof of preservation for a composition rule.

(iv) Give one case in the proof of preservation for a critical rule.

Task 5 (10 points)

(i) Give the statement of progress for the K machine.

(ii) State the form of its proof (that is, by induction or cases, and on which judgment). You do not need to show any cases.

2 Modifying the K Machine

Our current K machine evaluation strategy is *call-by-value* because it eagerly evaluates the argument to a function. However, there is an alternative strategy, *call-by-name*, that is preferred in some settings. This strategy does not try to evaluate function arguments prior to application, but substitutes the unevaluated expression into the body of a function. On the pure λ-calculus, call-by-name corresponds to a leftmost/outermost strategy.

There are tradeoffs between *call-by-value* and *call-by-name*. If an argument is never used in the function body, then *call by value* "wastes time" by evaluating the argument (which, the extreme case, may not terminate). However, if an argument is used multiple times, *call by name* "wastes time" by evaluating it separately each time it arises in the function body. And there are further variants beyond these, like *call-by-need* which is like *call-by-name*, but avoids reevaluating arguments, memoizing their value instead.

Task 6 (5 points) Modify the K machine evaluation rules for functions to be *call-by-name*. Feel free to add or remove rules as needed, or to add new K machine constructs so long as they are properly defined. Just make sure all your changes are clear, and that your resulting machine satisfies the usual properties (progress, preservation, and determinacy).

In lecture, we introduced the ability to raise exceptions, with $\mathsf{raise}\ E$ for some exception E. Most languages, however, do not *just* raise exceptions — they also allow you to catch them!

For this task, we introduce the *try-catch* block to our langauge. The idea is that $\mathsf{try}\ e_1\ \mathsf{catch}\ e_2$ attempts to evaluate and return e_1, but if it hits an exception, it will instead evaluate and return e_2. We can express this in our language formally with the following step rules:

$$\frac{e_1 \mapsto e_1'}{\mathsf{try}\ e_1\ \mathsf{catch}\ e_2 \mapsto \mathsf{try}\ e_1'\ \mathsf{catch}\ e_2}\ \mathsf{step/try}_0 \qquad \frac{v_1\ \mathit{value}}{\mathsf{try}\ v_1\ \mathsf{catch}\ e_2 \mapsto v_1}\ \mathsf{step/try/success}$$

$$\frac{}{\mathsf{try}\ (\mathsf{raise}\ E)\ \mathsf{catch}\ e_2 \mapsto e_2}\ \mathsf{step/try/fail}$$

Task 7 (10 points) Modify the transition rules in the K machine so they incorporate try-catch blocks.

Feel free to add or remove rules as needed, or to add new K machine constructs as long as they are properly defined. Just make sure all your changes are clear, and that your resulting machine satisfies the usual properties (progress, as modified to account for exceptions, preservation, and determinacy). You do not need to prove any of these properties.