Lecture Notes on Evaluation

15-417/817: HOT Compilation
Frank Pfenning

Lecture 5
February 28, 2025

1 Introduction

Earlier, we had said that the Sax implementation produces a heap from which we have to read off the large value, but we didn't specify what these large values should be. Now that we have the ND source language, we can plug this whole and specify the meaning of a source expression.

We use a high-level of specification, working directly with a *big-step semantics* that relates a given expression with its final value. We'd also like to preserve nontermination, which in our language just arises from the absence of a value for an expression. Correspondingly, the result of compilation to Sax should have no final configuration.

In lecture, we also covered some aspects of nested pattern matching which is written up in Lecture 4.

2 Semantic Rules

We only evaluate closed and well-typed expressions. Therefore, we don't need any context and define directly

$$e \hookrightarrow V$$

This judgment satisfies *preservation*:

If $\cdot \vdash e : A$ *and* $e \hookrightarrow V$ *then* $V : A$.

Also useful is that values are stable under evaluation:

$$V \hookrightarrow V$$

The *progress* property is more difficult to express since it relates to the process of constructing a derivation of $e \hookrightarrow V$ given e. This is a characteristic of *natural semantics* [Kahn, 1987]. The alternative of *structural operational semantics* [Plotkin, 1981] has some advantages (such as an understanding of progress), but usually has less parallelism. The *substructural operational semantics* [Pfenning, 2004, Simmons, 2012] we used for Sax represents some intermediate point.

As before, we proceed by types.

Pairs. The constructor is straightforward.

$$\frac{e_1 \hookrightarrow V_1 \quad e_2 \hookrightarrow V_2}{(e_1, e_2) \hookrightarrow (V_1, V_2)}$$

Even this is not formal in this form of big-step semantics, it does capture soe aspect of fork/join parallelism: evaluation of e_1 and e_2 can proceed independently and in parallel (the "fork") and the results can be combined to a pair (the "join").

The elimination rule requires substitution.

$$\frac{e \hookrightarrow (V, W) \quad e'(V, W) \hookrightarrow V'}{\mathbf{match}\ e\ ((x, y) \Rightarrow e'(x, y)) \hookrightarrow V'}$$

This does not intrinsically express the futures-based parallelism of Sax, because we assume that V and W are large values and cannot be addresses or contain addresses. These would be foreign to the level of abstraction we are working with in this big-step semantics.

Unit. As usual, this corresponds to a nullary pair.

$$\frac{}{() \hookrightarrow ()} \qquad \frac{e \hookrightarrow () \quad e' \hookrightarrow V'}{\mathbf{match}\ e\ (() \Rightarrow e') \hookrightarrow V'}$$

Sums. Again, fairly intuitive.

$$\frac{e \hookrightarrow V}{k(e) \hookrightarrow k(V)} \qquad \frac{(k \in L)e \hookrightarrow k(V) \quad e'_k(V) \hookrightarrow V'}{\mathbf{match}\ e\ \{\ell(x_\ell) \Rightarrow e'_\ell(x_\ell)\}_{\ell \in L} \hookrightarrow V'}$$

Top-level functions.

$$\frac{e_i \hookrightarrow V_i \quad (\forall i) \quad e'(\overline{V_i}) \hookrightarrow V' \quad F\ \overline{x_i} = e'(\overline{x_i})}{F\ \overline{e_i} \hookrightarrow V'}$$

That's already it! Substitution here is more straightforward than the general case because the values we substitute are closed and therefore no capture of bound variables can occur.

3 Divergence

We can just say that e diverges if there is no V such that $e \hookrightarrow V$. This is important because we want compilation to preserve nontermination: if the original program does not have a value, then the compiled program should not either.

We can only take this specification and turn it into inference rules through a mechanical process of negation. However, the result will be a mixed inductive-coinductive characterization of divergence.

For example, writing $e \Uparrow$ if e diverges, then for the construction of pairs we would have

$$\frac{e_1 \Uparrow}{(e_1, e_2) \Uparrow} \qquad \frac{e_2 \Uparrow}{(e_1, e_2) \Uparrow}$$

For destructions of pairs, we obtain several rules.

$$\frac{e \Uparrow}{\mathbf{match}\ e\ ((x,y) \Rightarrow e'(x,y)) \Uparrow} \qquad \frac{e \hookrightarrow (V,W) \quad e'(V,W) \Uparrow}{\mathbf{match}\ e\ ((x,y) \Rightarrow e'(x,y)) \Uparrow}$$

The second rule is only sufficient if we know that expressions are well-typed. If not, the e might evaluate to a value that is not a pair and the result would not have a value. It is in this rule that divergence calls upon the evaluation judgment.

It is important that the definition of $e \Uparrow$ is interpreted *coinductively*, that is, permitting infinite derivations. I believe a characterization using only rules under the usual inductive interpretation is impossible, because we would then have a decision procedure for termination to solve the halting problem.

The unit is once again boring.

$$\frac{e \Uparrow}{\mathbf{match}\ e\ (() \Rightarrow e') \Uparrow} \qquad \frac{e \hookrightarrow () \quad e' \Uparrow}{\mathbf{match}\ e\ (() \Rightarrow e') \Uparrow}$$

And, for sums:

$$\frac{e \Uparrow}{k(e) \Uparrow}$$

$$\frac{e \Uparrow}{\mathbf{match}\ e\ \{\ell(x_\ell) \Rightarrow e'(x_\ell)\}_{\ell \in L} \Uparrow} \qquad \frac{(k \in L) \quad e \hookrightarrow k(V) \quad e'_k(V) \Uparrow}{\mathbf{match}\ e\ \{\ell(x_\ell) \Rightarrow e'(x_\ell)\}_{\ell \in L} \Uparrow}$$

Again, we need the assumption of typing so that $(k \in L)$ will always be satisfied.

The desired theorem (which we haven't proved) is:

If $\cdot \vdash e : A$ then either $e \hookrightarrow V$ for some V or there is an infinite derivation of $e \Uparrow$.

4 Nested Pattern Matching

There are two principal approaches to adding nested pattern matching to the direct semantics. One is to instrument the typing judgment so it creates an expression with one-level pattern matching only. The other is to give a semantics directly. A good measure of correctness is to ascertain that the two approaches are equivalent. We show here the second approach.

Recall the judgment

$$\Delta \vdash \Omega \rhd K^+ \Longleftarrow C$$

where Ω is a sequence of types, and K^+ is a non-empty collection of branches.

Dynamically, Δ will be empty, and we will have a sequence of large values whose types correspond to those prescribed by Ω. That is

$$\frac{V_1 : A_1 \quad \ldots \quad V_n : A_n}{V_1 \cdots V_n : A_1 \cdots A_n}$$

with the evaluation judgment

$$\overline{V} \rhd K^+$$

We have the following rules:

$$\frac{V_1 \cdot V_2 \cdot \overline{W} \rhd (\mathsf{filter}^*\ (\_,\_)\ K^+) \hookrightarrow V'}{(V_1, V_2) \cdot \overline{W} \rhd K^+ \hookrightarrow V'} \qquad \frac{\overline{W} \rhd (\mathsf{filter}^*\ (\,)\ K^+) \hookrightarrow V'}{(\,) \cdot \overline{W} \rhd K^+ \hookrightarrow V'}$$

$$\frac{V \cdot \overline{W} \rhd (\mathsf{filter}^*\ k(\_)\ K^+) \hookrightarrow V'}{k(V) \cdot \overline{W} \rhd K^+ \hookrightarrow V'} \qquad \frac{e' \hookrightarrow V'}{(\cdot) \rhd (\cdot \Rightarrow e') \hookrightarrow V'}$$

$$\frac{(\mathsf{filter}^*\ x\ K^+) = J^+(x) \quad \overline{W} \rhd J^+(V) \hookrightarrow V' \quad x\ \mathsf{fresh}}{V \cdot \overline{W} \rhd K^+ \hookrightarrow V'}$$

The last rules is the most interesting because V can be of arbitrary type. Filtering returns a nonempty sequence of branches J^+ which may (and, in the linear case, *must*) depend on x so we can substitute V in all branches of $J^+(x)$.

References

Gilles Kahn. Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pages 22–39. Springer-Verlag LNCS 247, 1987.

Frank Pfenning. Substructural operational semantics and linear destination-passing style. In W.-N. Chin, editor, *Proceedings of the 2nd Asian Symposium on Programming Languages and Systems (APLAS'04)*, page 196, Taipei, Taiwan, November 2004. Springer-Verlag LNCS 3302. Abstract of invited talk.

Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981.

Robert J. Simmons. *Substructural Logical Specifications*. PhD thesis, Carnegie Mellon University, November 2012. Available as Technical Report CMU-CS-12-142.