Lecture Notes on
Substructural Operational Semantics

15-816: Linear Logic
Frank Pfenning

Lecture 12
February 27, 2012

In this lecture we view the operational semantics of a programming language as a stateful system with some transitions. This suggests we should be able to describe it in linear logic in a way that is analogous to the early examples used in this class, like blocks world. Since we use linear logic to describe operational semantics, we could use the term *linear operational semantics* to describe this technique. It turns out that other substructural logics, like ordered logic [PS09] or affine logic are often more appropriate, so we refer to it more generally as *substructural operational semantics* [Pfe04] (SSOS).

The particular semantics that we will describe is a transcription of the semantics via the translation to the session-typed π-calculus presented in the last lecture.

1 General Principles

The general idea is that we describe the state of a (potentially nondeterministic) abstract machine by propositions in linear logic. These are of the following forms:

Evaluation: $\mathsf{eval}(M, x)$ means that we would like to evaluate M and communicate the value along channel x. In the terminology of SSOS, x is called a *destination*.

Return: $\mathsf{retn}(V, x)$ means that we return value V to destination x.

Continuation: $\mathsf{cont}(x, F, w)$ means that we wait for a value on x to carry out F and pass the result to w.

Some flexibility will be required to interpret these correctly in each case. The overall idea is that we model state transitions so that

$$\mathsf{eval}(M, x) \longrightarrow^* \mathsf{retn}(V, x)$$

if and only if V is the value of M. Because of the frame property for linear contexts, we also get

$$\Delta, \mathsf{eval}(M, x) \longrightarrow^* \Delta, \mathsf{retn}(V, x)$$

State transitions, the way we discussed them at the very beginning of the class, are most directly modeled in the focusing calculus, where $\Delta \longrightarrow^* \Delta'$ if

$$\begin{array}{c} \Delta' \vdash C \\ \vdots \\ \Delta \vdash C \end{array}$$

where each step in the proof construction consists of focusing on a formula in the (omitted) Γ and playing the focusing phase together with the following inversion phase through to its conclusion. For example, focusing on an (unrestriction) $\mathsf{d} \multimap \mathsf{n} \otimes \mathsf{n}$ corresponds to the derived rule

$$\frac{\Delta, \mathsf{n}, \mathsf{n} \vdash C}{\Delta, \mathsf{d} \vdash C}$$

when we interpret all atoms as being positive. This interpretation will be discussed further in Lecture 13 on *Forward Chaining*.

2 Additive Pairs

Because of their relative simplicity, we start with additive pairs $\langle M, N\rangle$. Since we are trying to implement a concurrent semantics, here are the translations from the last lectures:

$$\begin{array}{l} [\langle M, N\rangle]^x = x.\mathsf{case}([M]^x, [N]^x) \\ [\pi_1 M]^w = (\nu x)([M]^x \mid x.\mathsf{inl}; [x \leftrightarrow w]) \\ [\pi_2 M]^w = (\nu x)([M]^x \mid x.\mathsf{inr}; [x \leftrightarrow w]) \end{array}$$

Let's consider $[\pi_1 M]^w$. The right-hand side means we have to create a new destination x and then evaluate M destination x. Meanwhile, we wait for a

pair to be returned on x to we can extract the first component and forward its value to w.

$$\mathsf{eval}(\pi_1 M, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \pi_1\_, w)$$

Let's take this apart a bit. The free variables in this rule, namely M and w are implicitly universally quantified outside the proposition, just like we do not make quantifiers explicit when writing inference rules. The existential quantifier on the right-hand side of the linear implication will add a new parameter to the term context. Its type would be that of a destination, which could be either a new separate type, or just be the same as the type of terms. In order to see why this introduces a new parameter we have to study focusing in the presence of quantification, which we postpone to another lecture. Briefly, it is because the left rule for existentials (which incidentally is invertible),

$$\frac{\Psi, n{:}\tau\ ;\ \Gamma\ ;\ \Delta, A\{n/x\} \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta, \exists x{:}\tau.A \to C}\ \exists L$$

adds a fresh n to the context Ψ of term parameters.

We obtain a symmetric form for the second projection.

$$\begin{array}{c}\mathsf{eval}(\pi_1 M, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \pi_1\_, w) \\ \mathsf{eval}(\pi_2 M, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \pi_2\_, w)\end{array}$$

A pair just waits for the first or second projection to be applied, so it functions as a value. That means it simply returns the pair to the given destination.

$$\mathsf{eval}(\langle M, N\rangle, x) \multimap \mathsf{retn}(\langle M, N\rangle, x)$$

Finally, the interaction of a pair with a projection takes place when a returned value meets its continuation.

$$\mathsf{retn}(\langle M, N\rangle, x) \otimes \mathsf{cont}(x, \pi_1\_, w) \multimap ??$$

To see what it has to transition to, let's recall the rules of translation to the session-typed π-calculus:

$$\begin{array}{l}[\langle M, N\rangle]^x = x.\mathsf{case}([M]^x, [N]^x) \\ [\pi_1 M]^w = (\nu x)([M]^x \mid x.\mathsf{inl}; [x \leftrightarrow w])\end{array}$$

When the inl is received by the case, both sides reduce: one to the first component of the pair $[M]^x$, the other to a channel forwarding $[x \leftrightarrow w]$. We therefore have

$$\mathsf{retn}(\langle M, N\rangle, x) \otimes \mathsf{cont}(x, \pi_1\_, w) \multimap \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \_, w)$$

where channel forwarding is implemented simply by a continuation which performs no other work. This forwarding needs to be implemented by a separate rule

$$\mathsf{retn}(V, x) \otimes \mathsf{cont}(x, \_, w) \multimap \mathsf{retn}(V, w)$$

It is not a priori obvious that we do not need the symmetric form:

$$\mathsf{retn}(V, w) \otimes \mathsf{cont}(x, \_, w) \multimap \mathsf{retn}(V, x) \quad ?$$

We should check at the end of the development.

Here is the summary of the substructural operational semantics, starting with specific rules for pairs, followed by a generic rule for forwarding.

$$\begin{array}{l}
\mathsf{eval}(\pi_1 M, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \pi_1_, w) \\
\mathsf{eval}(\pi_2 M, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \pi_2_, w) \\
\\
\mathsf{eval}(\langle M, N\rangle, x) \multimap \mathsf{retn}(\langle M, N\rangle, x) \\
\\
\mathsf{retn}(\langle M, N\rangle, x) \otimes \mathsf{cont}(x, \pi_1_, w) \multimap \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, _, w) \\
\\
\mathsf{retn}(V, x) \otimes \mathsf{cont}(x, _, w) \multimap \mathsf{retn}(V, w)
\end{array}$$

3 Functions

We recall the translation of functions from the last lecture.

$$\begin{array}{lcl}
[\lambda y.\, M]^x & = & x(y).[M]^x_y \\
[M\, N]^w & = & (\nu x)([M]^x \mid (\nu y)\overline{x}\langle y\rangle.([N]^y \mid [x \leftrightarrow w])) \\
[x]^w & = & [x \leftrightarrow w]
\end{array}$$

It is instructive to start with the elimination form. To evaluate $M\, N$ we immediately start the evaluation of M. We also create a process that would like send a new y on x, but has to wait until M is prepared to receive. So the second part of the translation is wrapped up in a continuation.

$$\mathsf{eval}(M\, N, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \_\, N, w)$$

λ-expressions do not immediately spawn evaluation, since they would like to receive an argument. So functions are returned as values.

$$\mathsf{eval}(\lambda y.\, M_y, x) \multimap \mathsf{retn}(\lambda y.\, M_y, x)$$

When the returned function meets its continuation, we have to create the appropriate processes that would be active after the interaction embodied in the translation of $[M\ N]^w$. We obtain:

$$\begin{array}{l}\mathsf{retn}(\lambda y.\, M_y, x) \otimes \mathsf{cont}(x, \_\ N, w) \\ \quad \multimap \exists z.\, \mathsf{eval}(M\{z/y\}_z, x) \otimes \mathsf{eval}(N, z) \otimes \mathsf{cont}(x, \_, w)\end{array}$$

To clarify the communication we have called the new variable z rather than y. It may be possible to optimize this rule, for example, by folding in the forwarding operation like so:

$$\begin{array}{l}\mathsf{retn}(\lambda y.\, M_y, x) \otimes \mathsf{cont}(x, \_\ N, w) \\ \quad \multimap \exists z.\, \mathsf{eval}(M\{z/y\}_z, w) \otimes \mathsf{eval}(N, z)\end{array}$$

Generally, we avoid such optimizations in order to remain as faithful as possible to the concurrent semantics we have already given.

When we encounter a variable we have to wait for its value, forwarding it to the target destination.

$$\mathsf{eval}(x, w) \multimap \mathsf{cont}(x, \_, w)$$

4 Example

With the constructs we already have, we can illustrate this operational semantics at work. We start in a state with a single destination w and the goal to evaluate $(\lambda y.\, \pi_1\langle y, y\rangle)V$, where V is an unspecified value we have initially. We would expect

$$\mathsf{eval}((\lambda y.\, \pi_1\langle y, y\rangle)\, V, w) \longrightarrow^* \mathsf{retn}(V, w)$$

Below, we show the available parameters on the left and evolve the state starting at the first line.

$$
\begin{array}{lcl}
w & ; & \mathsf{eval}((\lambda y.\, \pi_1\langle y, y\rangle)\, V, w) \\
w, x & ; & \mathsf{eval}(\lambda y.\, \pi_1\langle y, y\rangle, x), \mathsf{cont}(x, \_\, V, w) \\
w, x & ; & \mathsf{retn}(\lambda y.\, \pi_1\langle y, y\rangle, x), \mathsf{cont}(x, \_\, V, w) \\
w, x, z & ; & \mathsf{eval}(\pi_1\langle z, z\rangle, x), \mathsf{eval}(V, z), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{eval}(\langle z, z\rangle, y), \mathsf{cont}(y, \pi_1 \_, x), \mathsf{retn}(V, z), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{retn}(\langle z, z\rangle, y), \mathsf{cont}(y, \pi_1 \_, x), \mathsf{retn}(V, z), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{eval}(z, y), \mathsf{cont}(y, \_, x), \mathsf{retn}(V, z), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{cont}(z, \_, y), \mathsf{cont}(y, \_, x), \mathsf{retn}(V, z), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{retn}(V, y), \mathsf{cont}(y, \_, x), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{retn}(V, x), \mathsf{cont}(x, \_, w) \\
w, x, z, y & ; & \mathsf{retn}(V, w)
\end{array}
$$

In the step from line 4 to line 5, two different evaluations could proceed, so we carried them out simultaneously. At the end we arrived on the expected return, except that a number of destinations created along the way have accumulated. They could now be garbage-collected, leaving only the initial w.

5 Unrestricted Variables

Unrestricted variables are tied to persistent resources in the sequent calculus and the $!A$ exponential connective. From the last lecture:

$$
\begin{array}{lcl}
[u]^x & = & (\nu y)\overline{u}\langle y\rangle.[y \leftrightarrow x] \\
[!M]^x & = & !x(y).[M]^y \\
[\mathsf{let}\ !u = M\ \mathsf{in}\ N_u]^w & = & (\nu x)([M]^x \mid x/u.[N]^w_u)
\end{array}
$$

Again, let's start with the elimination form. We immediately proceed with the evaluation of M and wait for a communication opportunity along x so we can promote the linear x to a persistent u.

$$\mathsf{eval}(\mathsf{let}\ !u = M\ \mathsf{in}\ N_u, w) \multimap \exists x.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \mathsf{let}\ !u = \_\ \mathsf{in}\ N_u, w)$$

A term $!M$ is a value directly, representing a replicating input.

$$\mathsf{eval}(!M, x) \multimap \mathsf{retn}(!M, x)$$

When a return of $!M$ meets its continuation, the variable x is promoted, but M itself is not yet a candidate for evaluation. It's only at uses of u in the

body N_u that a new copy of M is scheduled for evaluation.

$$\mathsf{retn}(!M, x) \otimes \mathsf{cont}(x, \mathsf{let}\ !u = \_\ \mathsf{in}\ N_u, w) \multimap \exists u.\, !\mathsf{retn}(M, u) \otimes \mathsf{eval}(N_u, w)$$

Notice that M is *persistently* returned to u. This is needed because u may be used several times in the body N_u. When u is encountered in a context where the result is supposed to be passed to x, we span a new evaluation of M with destination y, which is forwarded to x.

$$\mathsf{eval}(u, x) \multimap \mathsf{cont}(u, \_, x)$$
$$!\mathsf{retn}(M, u) \otimes \mathsf{cont}(u, \_, x) \multimap \exists y.\, \mathsf{eval}(M, y) \otimes \mathsf{cont}(y, \_, x)$$

Again, we may be able to optimize and eliminate the creating of y, evaluating M directly with destination x.

It may be more intuitive to think of u as guarding a *persistent continuation* with a dynamically determined destination. Then we might write something like this:

$$\mathsf{retn}(!M, x) \otimes \mathsf{cont}(x, \mathsf{let}\ !u = \_\ \mathsf{in}\ N_u, w) \multimap \exists u.\, !\mathsf{cont}(u, M, \_) \otimes \mathsf{eval}(N_u, w)$$
$$\mathsf{eval}(u, x) \multimap \mathsf{retn}(x, u)$$
$$\mathsf{retn}(x, u) \otimes !\mathsf{cont}(u, M, \_) \multimap \exists y.\, \mathsf{eval}(M, y) \otimes \mathsf{cont}(y, \_, x)$$

Exercises

Exercise 1 Extend the substructural operational semantics of the linear λ-calculus by adding:

(i) Multiplicative pairs $A \otimes B$.

(ii) Multiplicative unit $\mathbf{1}$.

(iii) Disjunctions $A \oplus B$.

(iv) Contradiction $\mathbf{0}$.

(v) Additive unit $\top$.

References

[Pfe04] 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.

[PS09] Frank Pfenning and Robert J. Simmons. Substructural operational semantics as ordered logic programming. In *Proceedings of the 24th Annual Symposium on Logic in Computer Science (LICS 2009)*, pages 101–110, Los Angeles, California, August 2009. IEEE Computer Society Press.