Lecture Notes on Functional Computation

15-816: Linear Logic
Frank Pfenning

Lecture 11
February 22, 2012

In the linear λ-calculus from last lecture, we interpreted proof reductions as term reductions in an underlying language of proof terms. This is the original insight behind the Curry-Howard isomorphism [How69], albeit on an natural deduction for (non-linear) intuitionistic logic and simply-typed (non-linear) λ-terms. But there is still a significant step between proof term reduction and an operational semantics. For example, in functional programming languages such as ML or Haskell, we do not evaluate under λ-abstractions, and we impose a specific order of evaluation for functions such as call-by-value or call-by-name. Finding compelling logical underpinnnings for these is still an active area of research.

In this lecture we investigate a specific possibility for analyzing the computational content of linear natural deductions, using two pieces already in place: the translation from natural deduction to sequent calculus, and the interpretation of sequent proofs as concurrent processes. Combining these two will give us a concurrent operational semantics for the linear λ-calculus. More details and additional material on this translation can be found in Toninho et al. [TCP12]. This is definitely not the end of the story, however, because other interpretations are possible. We may return to them later in the course.

1 From Natural Deduction to Sequent Calculus

In the last lecture we stated that every natural deduction can be translated to a sequent calculus proof. Annotating the resulting sequent calculus proofs with their process interpretations will therefore provide us with a compilation of linear λ-terms to the session-typed π-calculus. The realization of this idea is remarkably similar to Milner's interpretation of functions as processes [Mil92], even though Milner's work was rooted in the untyped λ-calculus and π-calculus.

Recall that linear hypothetical judgments of natural deduction have the form

$$\Gamma \,;\, \Delta \Vdash M : A$$

where Γ consists of unrestricted variable declarations $u_j{:}B_j$ while Δ consists of linear variable declarations $x_i{:}A_i$. Similarly, sequents annotated with process terms have the form

$$\Gamma \,;\, \Delta \vdash P :: x : A$$

where Γ consists of shared channels $u_j{:}B_j$, Δ consists of linear channels $x_i{:}A_i$, and $x{:}A$ is also a linear channel.

Therefore, our goal is to translate proof terms M to processes P that preserves typing. We see that the processes use one additional channel, along which they offer a service, so the translation will have to take this as a parameter. We write

$$[M]^x = P$$

and expect the theorem

From Functions to Processes.
If $\Gamma \,;\, \Delta \Vdash M : A$ then $\Gamma \,;\, \Delta \vdash [M]^x :: x : A$.

We also expect some connection between reductions on M, as shown in the last lecture, and process reduction on $[M]^x$, but we will not try to anticipate it and instead extract it from the translation itself. We now proceed construct by construct to see what we obtain.

2 Variables

Variables are translated to channels, and even in the statement of our desired theorem we did not formally distinguish between them. We restate the case in the proof, now with proof terms.

$$\frac{}{x{:}A \Vdash x : A}\ \mathsf{hyp}$$

We construct:

$$\dfrac{}{x{:}A \vdash [x \leftrightarrow w] :: w : A}\ \mathsf{id}_A$$

This means

$$[x]^w \quad = \quad [x \leftrightarrow w]$$

3 Additive Pairs

Next, we consider additive pairs, typed by $A \mathbin{\&} B$. First, the introduction rule:

$$\dfrac{\Delta \Vdash M : A \quad \Delta \Vdash N : B}{\Delta \Vdash \langle M, N \rangle : A \mathbin{\&} B}\ \&I$$

The introduction rules correspond directly to right rules and we obtain

$$\dfrac{\Delta \vdash [M]^x :: x : A \quad \Delta \vdash [N]^x :: x : B}{\Delta \vdash x.\mathsf{case}([M]^x, [N]^x) :: x : A \mathbin{\&} B}\ \&R$$

So $[\langle M, N\rangle]^x = x.\mathsf{case}([M]^x, [N]^x)$. A pair therefore offers a choice between its components along the channel x. The client should be able to select the first or second component.

The elimination rules use cut in the target of the translation, so we can transport the formula to which we apply the elimination to the left-hand side of a sequent. The rule

$$\dfrac{\Delta \Vdash M : A \mathbin{\&} B}{\Delta \Vdash \pi_1 M : A}\ \&E_1$$

therefore becomes

$$\dfrac{\Delta \vdash A \mathbin{\&} B \quad \dfrac{\dfrac{}{A \vdash A}\ \mathsf{id}_A}{A \mathbin{\&} B \vdash A}\ \&L_1}{\Delta \vdash A}\ \mathsf{cut}_{A\&B}$$

Now we annotate this with process terms:

$$\dfrac{\Delta \vdash [M]^x :: x : A \mathbin{\&} B \quad \dfrac{\dfrac{}{x{:}A \vdash [x \leftrightarrow w] :: w : A}\ \mathsf{id}_A}{x{:}A \mathbin{\&} B \vdash x.\mathsf{inl}; [x \leftrightarrow w] :: w : A}\ \&L_1}{\Delta \vdash (\nu x)([M]^x \mid x.\mathsf{inl}; [x \leftrightarrow w]) :: w : A}\ \mathsf{cut}_{A\&B}$$

So, indeed, the client of a pair $[M]^x$ selects the first component by sending inl along x and forwarding the result. Summarizing the three cases for pairs:

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

Now we should examine the reduction that arises when a destructor (such as π_1 or π_2) meets a constructor (such as $\langle -, -\rangle$). We obtain:

$$\begin{array}{lcll} [\pi_1\langle M, N\rangle]^w & = & (\nu x)([\langle M, N\rangle]^x \mid x.\mathsf{inl}; [x \leftrightarrow w]) & \text{defn. of } []^w \\ & = & (\nu x)(x.\mathsf{case}([M]^x, [N]^x) \mid x.\mathsf{inl}; [x \leftrightarrow w]) & \text{defn. of } []^x \\ & \longrightarrow & (\nu x)([M]^x \mid [x \leftrightarrow w]) & \text{process reduction} \\ & \longrightarrow & [M]^w & \text{structural reduction} \end{array}$$

The last reduction here comes from a cut where the first premise is annotated with $[M]^x$ and the second premise is the identity. This reduces to the first premise under some renaming to make sure the offer takes place along the correct channel.

We can see from this that the induced operational semantics on natural deduction proof terms is lazy for pairs of type $A \,\&\, B$, since the $x.\mathsf{case}(-, -)$ prefix protects it from reduction. As predicted, we can project out the desired component. It also corresponds directly to the term reduction

$$\pi_1\langle M, N\rangle \longrightarrow_R M$$

4 Multiplicative Pairs

Recall

$$\frac{\Delta \Vdash M : A \quad \Delta' \Vdash N : B}{\Delta, \Delta' \Vdash M \otimes N : A \otimes B} \otimes I$$

Under the translation, and writing in process terms immediately, we obtain

$$\frac{\Delta \vdash [M]^y :: y : A \quad \Delta' \vdash [N]^x :: x : B}{\Delta, \Delta' \vdash (\nu y)\overline{x}\langle y\rangle.([M]^y \mid [N]^x) :: x : A \otimes B} \otimes R$$

At this point it is not clear what to make of this. In the synchronous π-calculus the output prefix means that neither $[M]^y$ nor $[N]^x$ can reduce. So this form of pair also appears to be lazy.

Perhaps the elimination rule will shed more light on the situation. We index terms here with the explicitly mentioned variables or channels that they may depend on, because it clarifies the communication patterns. We have:

$$\frac{\Delta \Vdash M : A \otimes B \quad \Delta', y{:}A, x{:}B \Vdash N_{y,x} : C}{\Delta, \Delta' \Vdash \mathsf{let}\ y \otimes x = M\ \mathsf{in}\ N_{y,x} : C}\ \otimes E$$

Here is the translation, first without process terms:

$$\dfrac{\Delta \vdash A \otimes B \quad \dfrac{\Delta', A, B \vdash C}{\Delta', A \otimes B \vdash C}\ \otimes L}{\Delta, \Delta' \vdash C}\ \mathsf{cut}_{A \otimes B}$$

Annotating this now with process terms we get:

$$\dfrac{\Delta \vdash [M]^x :: x : A \otimes B \quad \dfrac{\Delta', y{:}A, x{:}B \vdash [N]^w_{y,x} :: w : C}{\Delta', x{:}A \otimes B \vdash x(y).[N]^w_{y,x} :: w : C}\ \otimes L}{\Delta, \Delta' \vdash (\nu x)([M]^x \mid x(y).[N]^w_{y,x}) :: w : C}\ \mathsf{cut}_{A \otimes B}$$

In summary:

$$\begin{array}{lcl} [M \otimes N]^x & = & (\nu y)\overline{x}\langle y\rangle.([M]^y \mid [N]^x) \\ [\mathsf{let}\ y \otimes x = M\ \mathsf{in}\ N]^w & = & (\nu x)([M]^x \mid x(y).[N]^w_{y,x}) \end{array}$$

Checking out the reduction between the two:

$$\begin{array}{lcl} [\mathsf{let}\ y \otimes x = M_1 \otimes M_2\ \mathsf{in}\ N]^w & = & (\nu x)([M_1 \otimes M_2]^x \mid x(y).[N]^w_{y,x}) \\ & = & (\nu x)((\nu y)\overline{x}\langle y\rangle.([M_1]^y \mid [M_2]^x) \mid x(y).[N]^w_{y,x}) \\ & \longrightarrow & (\nu x)(\nu y)([M_1]^y \mid [M_2]^x \mid [N]^w_{y,x}) \end{array}$$

This reduction results in three parallel processes: $[M_1]^y$, communicating along y, $[M_2]^x$, communicating along x, and the body of the let-term $[N]^w_{y,x}$ which depends on both y and x. Even though the multiplicative pairs themselves have some aspect of lazy evaluation, they actually evaluate in parallel with the body, synchronizing on the variables y and x along which they communicate. For example, when the variable y is needed during the computation of N, it will have to wait until M_1 offers communication (either input or output, depending on the type of y).

5 Functions

Now we turn our attention to functions, keeping in mind that they must use their argument exactly once. First, the introduction rule corresponds to a right rule, as usual.

$$\frac{\Delta, y{:}A \Vdash M_y : B}{\Delta \Vdash \lambda y.\, M_y : A \multimap B} \multimap I$$

Translated (already with the proof terms):

$$\frac{\Delta, y{:}A \vdash [M]^x_y :: x : B}{\Delta \vdash x(y).[M]^x_y :: x : A \multimap B} \multimap R$$

We see that the the translation of a function $[\lambda y.M]^x = x(y).[M]^x_y$ first receives the function argument along x and then evaluates the body. However, the "function argument" here is not a value, but the name of a channel along which we can communicate with the argument. Let's look at the evaluation of function application.

$$\frac{\Delta \Vdash M : A \multimap B \quad \Delta' \Vdash N : A}{\Delta, \Delta' \Vdash M\,N : B} \multimap E$$

Here is what we construct, first without process terms.

$$\frac{\Delta \vdash A \multimap B \quad \dfrac{\Delta' \vdash A \quad \dfrac{}{B \vdash B}\,\mathsf{id}_B}{\Delta', A \multimap B \vdash B} \multimap L}{\Delta, \Delta' \vdash B} \mathsf{cut}_{A \multimap B}$$

Filling in process terms, we get:

$$\frac{\Delta \vdash [M]^x :: x : A \multimap B \quad \dfrac{\Delta' \vdash [N]^y :: y : A \quad \dfrac{}{x{:}B \vdash [x \leftrightarrow w] :: w : B}\,\mathsf{id}_B}{\Delta', x{:}A \multimap B \vdash (\nu y)\overline{x}\langle y\rangle.([N]^y \mid [x \leftrightarrow w]) :: w : B} \multimap L}{\Delta, \Delta' \vdash (\nu x)([M]^x \mid (\nu y)\overline{x}\langle y\rangle.([N]^y \mid [x \leftrightarrow w])) :: w : B} \mathsf{cut}_{A \multimap B}$$

We can see that the evaluation of the argument N is blocked until M is ready to receive an input. This will be the case when it has reduced to the translation of a function. Let's calculate:

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

We see that the function body is evaluated in parallel with the function argument, synchronizing on the uses of the variable y. Writing the natural deduction reduction

$$(\lambda y.\, M)\, N \longrightarrow [N/y]M$$

we see that instead of carrying out a substitution over the term M we tie the computation of N to y, now viewed as a channel. In general, when a term is typed as

$$\Delta \Vdash M : A$$

then $[M]^x$ will compute in an environment where each channel $x_i{:}A_i$ declared in Δ will be tied to process communicating along it, implementing the concurrent evaluation of a process $[N_i]^{x_i}$.

To relate reduction to the translation, we expect some relation

$$[[N/y]M]^w \simeq (\nu y)([N]^y \mid [M]^w_y)$$

Of course, this does not hold has an equality, since N may be deeply embedded in $[N/y]M$ while it is available at the top-level of the process on the right. Still, the process term on the right (logically a cut) can be seen as an implementation of the term on the left (logically a substitution).

The evaluation strategy that this corresponds to is *futures* [Hal85]: we evaluate the argument at the same time as the function body, synchronizing on accesses to variables. This can be specialized into call-by-value and call-by-need. If we always schedule reduction on the argument (called N above) first, then it becomes call-by-value. If we always schedule reduction on the body first, then it becomes call-by-need. Linearity of the variable ensures not only that it will be needed, but that its value is needed only once.

6 Persistence

So far, all variables and corresponding channels have been linear. How do unrestricted variables (which correspond to shared channels) fit into the picture?

First, the unrestricted hypothesis rule.

$$\frac{u{:}A \in \Gamma}{\Gamma\ ;\ \cdot \Vdash u : A}\ \mathsf{uhyp}$$

Without process terms, we construct the sequent proof as follows:

$$\frac{\dfrac{}{\Gamma\ ;\ A \vdash A}\ \mathsf{id} \qquad A \in \Gamma}{\Gamma\ ;\ \cdot \vdash A}\ \mathsf{copy}$$

Assigning proof terms:

$$\frac{\dfrac{}{\Gamma\ ;\ y{:}A \vdash [y \leftrightarrow x] :: x : A}\ \mathsf{id} \qquad u{:}A \in \Gamma}{\Gamma\ ;\ \cdot \vdash (\nu y)\overline{u}\langle y\rangle.[y \leftrightarrow x] :: x : A}\ \mathsf{copy}$$

So the translation of an unrestricted variable appearing as a term, $[u]^x$, creates a new channel and sends it to u, tying the result to x.

Translating the $!I$ rule should tell us which form of process is listening on this channel.

$$\frac{\Gamma\ ;\ \cdot \vdash M : A}{\Gamma\ ;\ \cdot \vdash\ !M :\ !A}\ !I$$

Translating the proof

$$\frac{\Gamma\ ;\ \cdot \vdash [M]^y :: y : A}{\Gamma\ ;\ \cdot \vdash\ !x(y).[M]^y :: x :\ !A}\ !R$$

reveals that the process is a replicating input that can create arbitrarily many copies of a term $[M]^y$.

The elimination rule will set up the communcation channel along u.

$$\frac{\Gamma\ ;\ \Delta \Vdash M :\ !A \qquad \Gamma, u{:}A\ ;\ \Delta' \Vdash N : C}{\Gamma\ ;\ \Delta, \Delta' \Vdash \mathsf{let}\ !u = M\ \mathsf{in}\ N : C}\ !E$$

Translated as

$$\frac{\Gamma\ ;\ \Delta \vdash\ !A \qquad \dfrac{\Gamma, A\ ;\ \Delta' \vdash C}{\Gamma\ ;\ \Delta', !A \vdash C}\ !L}{\Gamma\ ;\ \Delta, \Delta' \vdash C}\ \mathsf{cut}_{!A}$$

and with process terms:

$$\dfrac{\Gamma\ ;\ \Delta \vdash [M]^x :: x : !A \qquad \dfrac{\Gamma, u{:}A\ ;\ \Delta' \vdash [N]^w_u :: w : C}{\Gamma\ ;\ \Delta', x{:}!A \vdash x/u.[N]^w_u :: w : C}\ !L}{\Gamma\ ;\ \Delta, \Delta' \vdash (\nu x)([M]^x \mid x/u.[N]^w_u) :: w : C}\ \mathsf{cut}_{!A}$$

Let's summarize these translations:

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

Again, we calculate the reductions to obtain a reading for the operational semantics under the process interpretation.

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

We see that u is tied to a replicating input along u, as expected, and that the body of the let will evaluate. This will happen as soon as the argument was prepared to receive along x, that is, had been reduced to a replicating input along x.

Comparing this to the term reduction

$$\mathsf{let}\ !u = !M\ \mathsf{in}\ N \quad \longrightarrow \quad [M/u]N$$

we see that substitution for unrestricted variables is implemented differently than linear substitution. We expect some relation

$$[[M/u]N]^w \simeq (\nu u)(!u(y).[M]^y \mid [N]^w_u)$$

where the left-hand side is unrestricted substitution and the right-hand side is cut!.

Together, these reductions give us a copying interpretation for $!A$. We compute a value $!M$, where a fresh copy of M is evaluated every time it is used. This is like a call-by-name interpretation of function calls, except that here it is tied to a let-elimination (see Exercise 3).

For now, we just have the preservation of types across the translation.

Theorem 1 (From Functions to Processes)
If $\Gamma\ ;\ \Delta \Vdash M : A$ *then* $\Gamma\ ;\ \Delta \vdash [M]^x :: x : A$.

Proof: By induction on the structure of the given deduction. The reasoning in each case is contained in the proof translations constructed in this lecture.
□

Exercises

Exercise 1 Play through the computational interpretation of disjunction $A \oplus B$ via an interpretation into the session-typed π-calculus in the manner of this lecture.

Exercise 2 Play through the computation interpretation of the multiplicative unit $\mathbf{1}$ via an interpretation into the session-typed π-calculus in the manner of this lecture.

Exercise 3 A simple embedding of (unrestricted) functions into the linear functional language is by defining $A \supset B = (!A) \multimap B$. Explore this definition. Specifically:

(i) Give derived term assignments for introduction and elimination rules for $A \supset B$ under the above definition.

(ii) Translate the derived term assignments into process terms for the sequent calculus.

(iii) Explain the evaluation strategy that corresponds to this embedding of unrestricted functions into the linear λ-calculus.

References

[Hal85] Robert H. Halstead. Multilisp: A language for parallel symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–539, October 1985.

[How69] W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980), 1969.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[TCP12] Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In L. Birkedal, editor, *15th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS'12, Tallinn, Estonia, March 2012. Springer LNCS. To appear.