Lecture Notes on Forward Chaining

15-816: Linear Logic
Frank Pfenning

Lecture 13
February 29, 2012

In this lecture we start discussing proof search as a basic computational mechanism; so far, it has only been proof reduction. Computation via proof search is the technique underlying *logic programming*. It provides a fundamentally different way of combining programming with reasoning, by using reasoning itself as the source of computation.

The particular instance of this idea we introduce in this lecture is *linear forward chaining*, also called *linear bottom-up logic programming*. Linear forward chaining is supported in the Celf implementation [SNS08][1] of the CLF logical framework [WCPW02, CPWW02], and also in LolliMon [LPPW05].

Our goal for this lecture is to provide enough intuition regarding forward chaining and its implementation that we can write Celf code implementing substructural operational semantics. Some of this will remain mysterious since CLF provides a dependent type theory that we haven't covered yet in this course.

1 Derived Rules via Focusing

We previously discussed that we can go from certain propositions back to derived rules via the process of focusing. Let's reconsider this and then add quantifiers.

Assume we have an inference rule

$$\frac{\mathsf{q}}{\mathsf{d} \quad \mathsf{d} \quad \mathsf{n}}$$

[1] More information on the Software page for this course

When we explained linear inference we used this rule for the purpose of effecting a *state transition*. For example,

$$\mathsf{q}, \mathsf{n} \longrightarrow \mathsf{d}, \mathsf{d}, \mathsf{n}, \mathsf{n}$$

In general, applying linear rules of inference allowed transitions

$$\Delta \longrightarrow \Delta'$$

to be described by inference rules. We allowed the state to contain persistent as well as ephemeral propositions, so with our new notation we might rewrite this as

$$(\Gamma \ ; \ \Delta) \longrightarrow (\Gamma' \ ; \ \Delta')$$

where $\Gamma \subseteq \Gamma'$.

When we switched from linear inference to a sequent calculus (since we were unable to represent nested hypothetical judgments in a satisfactory way), we turned our sample rule from above into a *persistent proposition*

$$\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}$$

In which way does this now represent a possible state transition? Assume we have the proposition above as part of Γ and a stable sequent

$$\Gamma \ ; \ \Delta \to C$$

Recall that a sequent if stable if all propositions in Δ are negative or positive atoms, and C is either positive or a negative atom.

At this point we can potentially focus on any proposition in Γ or Δ (except for positive atoms), or on C (unless it is a negative atom). Consider the derivation if we focus on $\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \in \Gamma$:

$$\dfrac{\dfrac{\Gamma \ ; \ \Delta_1 \to [\mathsf{q}] \quad \dfrac{\vdots}{\Gamma \ ; \ \Delta_2, \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \to C}}{\Gamma \ ; \ \Delta_2, [\mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}\ \mathsf{blur}L}{\Gamma \ ; \ \Delta, [\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}\ {\multimap}L}{\Gamma \ ; \ \Delta \to C}\ \mathsf{focus}L!$$

where $\Delta = (\Delta_1, \Delta_2)$. Note that after the first focusing step, all the rules are forced.

Let's now fix the polarity of all atoms to be *positive*. In that case, the first remaining subgoal can only succeed if Γ contains q or $\Delta_1 = \mathsf{q}$. Let's assume

we are in a situation where we know that q cannot be in Γ, so $\Delta_1 = \mathsf{q}$ is forced. Completing the first subproof and substituting in our knowledge on the form of Δ, we obtain:

$$\dfrac{\dfrac{}{\Gamma\ ;\ \mathsf{q} \to [\mathsf{q}]}\ \mathsf{id}_{q^+} \quad \dfrac{\vdots}{\dfrac{\Gamma\ ;\ \Delta_2, \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \to C}{\Gamma\ ;\ \Delta_2, [\mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}\ \mathsf{blur}L}}{\dfrac{\Gamma\ ;\ \Delta_2, \mathsf{q}, [\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}{\Gamma\ ;\ \Delta_2, \mathsf{q} \to C}\ \mathsf{focus!}}\ \multimap L$$

Multiplicative conjunction is invertible on the left. So in a full focusing system, the inversion on $\mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}$ is also forced. We arrive at:

$$\dfrac{\dfrac{}{\Gamma\ ;\ \mathsf{q} \to [\mathsf{q}]}\ \mathsf{id}_{q^+} \quad \dfrac{\dfrac{\Gamma\ ;\ \Delta_2, \mathsf{d}, \mathsf{d}, \mathsf{n} \to C}{\Gamma\ ;\ \Delta_2, \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \to C}\ \otimes L \times 2}{\Gamma\ ;\ \Delta_2, [\mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}\ \mathsf{blur}L}}{\dfrac{\Gamma\ ;\ \Delta_2, \mathsf{q}, [\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}] \to C}{\Gamma\ ;\ \Delta_2, \mathsf{q} \to C}\ \mathsf{focus!}}\ \multimap L$$

At this point we again have a stable sequent, since we assumed the original goal sequent was stable. Writing it out explicitly, we obtain the following derived rule:

$$\dfrac{\Gamma\ ;\ \Delta, \mathsf{d}, \mathsf{d}, \mathsf{n} \to C}{\Gamma\ ;\ \Delta, \mathsf{q} \to C}$$

A remarkable observation is that in a system of focusing, this represents the *only* way we can use the persistent assumption $\mathsf{q} \multimap \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n}$. In other words, we could throw away this assumption and just agree to use the above derived rule instead.

Looking at the derived rule we can now see how state change is represented in the sequent calculus. If, under linear inference, we had a transition

$$\Delta \longrightarrow \Delta'$$

then using focusing on the corresponding proposition in Γ we obtain the transition

$$\dfrac{\Gamma\ ;\ \Delta' \to C}{\Gamma\ ;\ \Delta \to C}$$

More generally, if we allow persistent propositions as well, then the context Γ plays two roles: one for the propositions representing rules Γ_r, the other for persistent atomic propositions. Then a transition $(\Gamma\ ;\ \Delta) \longrightarrow (\Gamma'\ ;\ \Delta')$ becomes

$$\frac{\Gamma_r, \Gamma'\ ;\ \Delta' \to C}{\Gamma_r, \Gamma\ ;\ \Delta \to C}$$

Note that the right-hand side C plays no significant role here.

In summary, we can model linear inference using persistent propositions and focusing by assigning all atoms a positive polarity.

2 Focusing on Quantifiers

Our goal for this lecture is to implement substructural operational semantics as an executable forward-chaining logic program. Our discussion of chaining and focusing so far lacks quantifiers, which are necessary for many programs.

The universal quantifier is invertible on the right, so we focus on the left and invert on the right.

$$\frac{\Psi, n{:}\tau\ ;\ \Gamma\ ;\ \Delta \to A\{n/x\}}{\Psi\ ;\ \Gamma\ ;\ \Delta \to \forall x{:}\tau.A}\ \forall R \qquad \frac{\Psi \vdash M : \tau \quad \Psi\ ;\ \Gamma\ ;\ \Delta, [A\{M/x\}] \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta, [\forall x{:}\tau.A] \to C}\ \forall L$$

The right rule introduces a new parameter n into the term context Ψ; the left rule instantiates the quantifier with a term of the correct type.

Conversely, the existential quantifier is invertible on the left, so we focus on the right and invert on the left.

$$\frac{\Psi \vdash M : \tau \quad \Psi\ ;\ \Gamma\ ;\ \Delta \to [A\{M/x\}]}{\Psi\ ;\ \Gamma\ ;\ \Delta \to [\exists x{:}\tau.A]}\ \exists R \qquad \frac{\Psi, n{:}\tau\ ;\ \Gamma\ ;\ \Delta, A\{n/x\} \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta, \exists x{:}\tau.A \to C}\ \exists L$$

It is straightforward to extend the theorems regarding the soundness and completeness of chaining and focusing to the quantifiers. Crucial is the observation that we can consider $A\{M/x\}$ to be strictly smaller than $\exists x{:}\tau.A$ and $\forall x{:}\tau.A$, because $A\{M/x\}$ has fewer quantifiers and connectives.

3 Derived Rules from Quantifiers

Once we add quantifiers, our state transitions have to account for the possibility that new term parameters are introduced, either by the $\exists L$ rule or

the $\forall R$ rule. Then a state transition

$$(\Psi\ ;\ \Gamma\ ;\ \Delta) \longrightarrow (\Psi'\ ;\ \Gamma'\ ;\ \Delta')$$

with $\Psi \subseteq \Psi'$ and $\Gamma \subseteq \Gamma'$ will be modeled by

$$\frac{\Psi_c, \Psi'\ ;\ \Gamma_r, \Gamma'\ ;\ \Delta' \to C}{\Psi_c, \Psi\ ;\ \Gamma_r, \Gamma\ ;\ \Delta \to C}$$

where Ψ_c contains constants from the term language and Γ_r contains the transition rules, rendered as linear logical propositions.

Let's play through an example from substructural operational semantics.

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

Assuming a type tm for terms and dest for destinations:

$$\begin{array}{l}\forall m{:}\mathsf{tm}.\ \forall n{:}\mathsf{tm}.\ \forall w{:}\mathsf{dest}. \\ \quad \mathsf{eval}(m\ n, w) \multimap \exists x{:}\mathsf{dest}.\, \mathsf{eval}(m, x) \otimes \mathsf{cont}(x, \_\ n, w)\end{array}$$

We assign eval and cont a positive polarity. Focusing on this an pursuing three steps:

$$\dfrac{\Psi \vdash M{:}\mathsf{tm} \qquad \dfrac{\Psi \vdash N{:}\mathsf{tm} \qquad \dfrac{\vdots \quad \dfrac{\Psi \vdash W{:}\mathsf{dest} \quad \Psi\ ;\ \Gamma\ ;\ \Delta, [\cdots] \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta, \forall w{:}\mathsf{dest}.\, [\cdots] \to C}\forall L}{\Psi\ ;\ \Gamma\ ;\ \Delta, [\forall n{:}\mathsf{tm}.\ \forall w{:}\mathsf{dest}.\ \cdots] \to C}\forall L}{\Psi\ ;\ \Gamma\ ;\ \Delta, [\forall m{:}\mathsf{tm}.\ \forall n{:}\mathsf{tm}.\ \forall w{:}\mathsf{dest}.\ \cdots] \to C}\forall L$$

At this point the open subproof is forced as follows:

$$\dfrac{\Psi\ ;\ \Gamma\ ;\ \Delta_1 \to [\mathsf{eval}(M\ N, W)] \qquad \dfrac{\vdots \quad \dfrac{\Psi\ ;\ \Gamma\ ;\ \Delta_2, \exists x{:}\mathsf{dest}.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \_\ N, W) \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta_2, [\exists x{:}\mathsf{dest}.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \_\ N, W)] \to C}\mathsf{blur}L}{\Psi\ ;\ \Gamma\ ;\ \Delta, [\mathsf{eval}(M\ N, W) \multimap \exists x{:}\mathsf{dest}.\, \mathsf{eval}(M, x) \otimes \mathsf{cont}(x, \_\ N, W)]}\multimap L$$

where $\Delta = (\Delta_1, \Delta_2)$. We notice that eval is positive, and so $\Delta_1 = \mathsf{eval}(M\ N, W)$ and the first open goal is closed with the positive identity rule. Carrying

out the inversions in the second open derivation, we get:

$$\dfrac{\dfrac{\dfrac{\Psi, x{:}\mathsf{dest}\ ;\ \Gamma\ ;\ \Delta_2, \mathsf{eval}(M,x), \mathsf{cont}(x, \_\ N, W) \to C}{\Psi, x{:}\mathsf{dest}\ ;\ \Gamma\ ;\ \Delta_2, \mathsf{eval}(M,x) \otimes \mathsf{cont}(x, \_\ N, W) \to C}\otimes L}{\Psi\ ;\ \Gamma\ ;\ \Delta_2, \exists x{:}\mathsf{dest}.\, \mathsf{eval}(M,x) \otimes \mathsf{cont}(x, \_\ N, W) \to C}\exists L}{\Psi\ ;\ \Gamma\ ;\ \Delta_2, [\exists x{:}\mathsf{dest}.\, \mathsf{eval}(M,x) \otimes \mathsf{cont}(x, \_\ N, W)] \to C}\ \mathsf{blur}L$$

Putting these all together, and renaming Δ_2 to Δ, we obtain the following derived rule:

$$\dfrac{\begin{array}{l}\Psi \vdash M : \mathsf{tm} \\ \Psi \vdash N : \mathsf{tm} \\ \Psi \vdash W : \mathsf{dest} \\ \Psi, x{:}\mathsf{dest}\ ;\ \Gamma\ ;\ \Delta, \mathsf{eval}(M,x), \mathsf{cont}(x, \_\ N, W) \to C\end{array}}{\Psi\ ;\ \Gamma\ ;\ \Delta, \mathsf{eval}(M\ N, W) \to C}$$

Generally, we assume that any proposition in the context is well-formed, so the fact that a proposition $\mathsf{eval}(M\ N, W)$ is in the context implies the first three premises, and we are left with

$$\dfrac{\Psi, x{:}\mathsf{dest}\ ;\ \Gamma\ ;\ \Delta, \mathsf{eval}(M,x), \mathsf{cont}(x, \_\ N, W) \to C}{\Psi\ ;\ \Gamma\ ;\ \Delta, \mathsf{eval}(M\ N, W) \to C}$$

Let's read this rule. If we have a proposition $\mathsf{eval}(M\ N, W)$ which states that we have to evaluate the application of M to N with destination W, then we introduce a new destination x and replace $\mathsf{eval}(M\ N, W)$ with $\mathsf{eval}(M, x)$ and $\mathsf{cont}(x, \_\ N, W)$. This is precisely how we would like our substructural operational semantics to proceed.

The outermost universally quantified variables in the propositions turn into schematic variables in the derived rule, while existentially quantified variables turn into new parameters in the premise.

4 Forward Chaining

Forward chaining seems to work for a specific class of formulas. We can reverse engineer this class from the previous example. We want to maintain that stable sequents have the form

$$\Psi\ ;\ D_1, \ldots, D_k, Q_1^+, \ldots Q_m^+\ ;\ P_1^+, \ldots, P_n^+ \to G$$

where Q_j^+ and P_i^+ are positive atoms. The following grammar allows us to stay within this fragment.

$$\begin{array}{lrcl} \text{Clauses} & D & ::= & \forall x{:}\tau.D \mid D_1 \mathbin{\&} D_2 \mid \top \mid G \multimap D \mid H \\ \text{Heads} & H & ::= & P^+ \mid H_1 \otimes H_2 \mid \mathbf{1} \mid \exists x{:}\tau.H \mid H_1 \oplus H_2 \mid \mathbf{0} \mid !D \\ \text{Goals} & G & ::= & P^+ \mid G_1 \otimes G_2 \mid \mathbf{1} \mid \exists x{:}\tau.G \mid G_1 \oplus G_2 \mid \mathbf{0} \end{array}$$

Focusing on a *persistent clause* D in a context of all positive atoms will start a chaining phase followed by an inversion phase and can lead only to states with only persistent clauses and positive atoms. The right-hand side will never be involved in any such inference. Goals are slightly more restricted than heads, because we want a sequent

$$\Psi \mathbin{;} \Gamma \mathbin{;} \Delta \to [G]$$

to be quickly and efficiently decidable which can be achieved by making sure the G is purely positive, with no negative subformulas. Such subgoals will arise from a left focus on $[G \multimap D]$. Dually, we lose focus in a sequent $\Psi \mathbin{;} \Gamma \mathbin{;} [H] \to G$ and then can invert all the way down to positive atoms or new unrestricted clauses added to Γ. Methods for finding correct instances for the universal quantifiers in clauses and existential quantifiers in goals will be discussed in a future lecture.

Carrying out repeated focusing in a *don't-care nondeterministic* manner is forward-chaining logic programming. The system terminates when no further focusing steps can be carried out, a situation we call *quiescence*. At this point we can examine the state by focusing on the right-hand side, which again poses a decidable question.

The fact that forward chaining proceeds with don't-care nondeterminism or *committed choice* means that a program is only correct if *all* choices lead to the correct answer or computation. This is unlike our earlier modeling of stateful systems, where we deemed a problem representation correct if a proof existed if and only if a solution to the problem existed. Forward chaining logic programming imposes a much more stringent correctness requirement.

5 Using Celf

The Celf implementation of CLF supports both forward chaining and backward chaining, although in a slightly different style than presented in this lecture. As a result, there will be a few matters of syntax and semantics that will seem mysterious, until we explain them later on in this course.

We aim to directly implement the substructural operational semantics developed in the last lecture. We start with pairs. First we have to define the terms that are typed in the $\Psi \vdash M : \tau$ judgment. Unlike in first-order linear logic, in CLF these terms can also be typed in a linear fashion. We declare:

```
tm : type.

pair : tm & tm -o tm.
pi1 : tm -o tm.
pi2 : tm -o tm.
```

Note that `&` here is the additive conjunction of the framework. The representation function looks like this:

$$\begin{array}{lcl} \ulcorner\langle M, N\rangle\urcorner & = & \texttt{pair <}\ulcorner M\urcorner\texttt{,}\ulcorner N\urcorner\texttt{>} \\ \ulcorner\pi_1 M\urcorner & = & \texttt{pi1}\ulcorner M\urcorner \\ \ulcorner\pi_2 M\urcorner & = & \texttt{pi2}\ulcorner M\urcorner \end{array}$$

Next we need a type of destinations, as well as predicates `eval`, `retn`, and `cont`. These are actually represented as type families, a distinction we can ignore for now.

```
dest : type.

eval : tm -> dest -> type.
retn : tm -> dest -> type.
cont : dest -> (tm -o tm) -> dest -> type.
```

An interesting question is how we represent the frames that are part of continuation. Recall that they are like terms with a hole, where a subterm was extracted for evaluation. A term with a hole can be represented as a linear function from terms to terms, written `tm -o tm` in the declarations above. Now recall the linear specification of the substructural semantics.

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

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

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

We transcribe this in a fairly straightforward way, optimizing slightly by omitting the intermediate continuation frame in the two projection rules, evaluating the component directly with destination w.

```
ev/pair : eval (pair < M, N >) X -o {retn (pair < M, N >) X}.
ev/pi1 : eval (pi1 M) W -o {Exists x. eval M x * cont x (\h. pi1 h) W}.
ev/pi2 : eval (pi2 M) W -o {Exists x. eval M x * cont x (\h. pi2 h) W}.
ev/proj1 : retn (pair < M, N >) X * cont X (\h. pi1 h) W
            -o {eval M W}.
ev/proj2 : retn (pair < M, N >) X * cont X (\h. pi2 h) W
            -o {eval N W}.
```

We note that the all free variables must be capitalized, and are implicitly universally quantified. We also note that the succedents of the forward-chaining linear implication are enclosed in `{` braces `}`. They define a so-called *monad* which is used to control the interaction between forward and backward chaining in CLF. For the purpose of this lecture we can ignore them.

Next we come to functions. We need to extend our representation function, which is somewhat tricky, since the linear λ-abstraction binds a variable. We map this into a corresponding abstraction in CLF.

$$\begin{array}{lcl} \ulcorner \lambda y.\, M \urcorner & = & \texttt{lam (\y.}\ulcorner M \urcorner\texttt{)} \\ \ulcorner y \urcorner & = & \texttt{y} \\ \ulcorner M\, N \urcorner & = & \texttt{app}\ulcorner M \urcorner \ulcorner N \urcorner \end{array}$$

In addition, we have to substitute a destination for a variable, so we need a corresponding constructor. Note that in a source expression (before evaluation), there should never be a destination in a term.

```
lam : (tm -o tm) -o tm.
app : tm -o tm -o tm.

dst : dest -o tm.
```

Recall the SSOS specification of evaluation.

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

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

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

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

Again, we transcribe this into Celf. We optimize the interaction rule by elimination the new continuation that just forwards from x to w, evaluating $M\{z/y\}$ directly with destination w.

```
ev/lam : eval (lam \y. M y) X -o {retn (lam \y. M y) X}.
ev/app : eval (app M N) W
          -o {Exists x. eval M x * cont x (\h. app h N) W}.
ev/red : retn (lam \y. M y) X * cont X (\h. app h N) W
          -o {Exists z. eval (M (dst z)) W * eval N z}.
ev/var : eval (dst Z) W -o {cont Z (\h. h) W}.
ev/fwd : retn V Z * cont Z (\h. h) W -o {retn V W}.
```

Interesting here is how we represented $M\{z/y\}$. First, we cannot substitute z directly, because z is a destination, not a term. So we substitute `(dst z)` instead, coercing the destination into a term. Secondly, we use the compositionality of the representation function so that

$$\ulcorner M\{z/y\}\urcorner = [\ulcorner z\urcorner/\mathtt{y}]\ulcorner M\urcorner = \mathtt{(\backslash y.\ }\ulcorner M\urcorner\mathtt{)}\ulcorner z\urcorner$$

We will discuss this technique, often called *higher-order abstract syntax*, in more detail in a future lecture.

Finally, we come to unrestriced variables and terms of type $!A$ in the linear λ-calculus. We extend our representation:

$$\begin{array}{lcl} \ulcorner !M\urcorner & = & \mathtt{bang\ (!}\ulcorner M\urcorner\mathtt{)} \\ \ulcorner \mathsf{let}\ !u = M\ \mathsf{in}\ N\urcorner & = & \mathtt{ulet}\ulcorner M\urcorner\mathtt{(\backslash !u.}\ulcorner N\urcorner\mathtt{)} \\ \ulcorner u\urcorner & = & \mathtt{!u} \end{array}$$

Here, `\!u.N` in the framework is a persistent abstraction that has type `!tm -o tm` which can be written equivalently as `tm -> tm`. This leads to the following declarations:

```
bang : tm -> tm.
ulet : tm -o (tm -> tm) -o tm.

udst : dest -> tm.
```

The last declaration, of `udst`, allows us to include persistent destinations in terms, marking them as unrestricted.

Now recall the substructural operational semantics rules:

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

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

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

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

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

We combine the last two rules, avoiding the creation of two extra continuations.

```
ev/bang : eval (bang M) X -o {retn (bang M) X}.
ev/ulet : eval (ulet M (\!u. N !u)) W
          -o {Exists x. eval M x * cont x (\h. ulet h (\!u. N !u)) W}.
ev/bangred : retn (bang M) X * cont X (\h. ulet h (\!u. N !u)) W
          -o {Exists u. !retn M u * eval (N !(udst u)) W}.
ev/uvar : eval (udst U) W * !retn M U -o {eval M W}.
```

Note that `!retn M U` in the clause `ev/uvar` is not actually permitted in the forward-chaining fragment we defined earlier. As noted by a student after lecture, we could equally well write `retn M U` here, although it turns out that the Celf implementation can handle either one.

In lecture, we created a frame `cont U (\h. h) W`, but this is slightly dishonest, since it doesn't in fact pass on what is returned along U, but evaluates it. This clause created some incorrect nondeterminism, since the linear forwarding rule can now apply to unrestricted destinations, which was not intended in the original SSOS specification where we made a stronger distinction between linear and unrestricted destinations.

Exercises

Exercise 1 Extend the proofs of completeness of chaining from Lecture 9 to include quantifiers.

(i) Admissibility of identity (Theorem 3)

(ii) Admissibility of cut (Theorem 4)

(iii) Completeness of chaining (Theorem 2)

Exercise 2 Extend the Celf implementation of the linear λ-calculus to cover

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

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

(iii) Disjunctions $A \oplus B$.

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

(v) Additive unit $\top$.

(See also Exercise L12.1)

References

[CPWW02] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[LPPW05] Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In A.Felty, editor, *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, pages 35–46, Lisbon, Portugal, July 2005. ACM Press.

[SNS08] Anders Schack-Nielsen and Carsten Schürmann. Celf - a logical framework for deductive and concurrent systems. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, pages 320–326, Sydney, Australia, August 2008. Springer LNCS 5195.

[WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.