CS411 Translating LS to SS

A. Demers

8 Mar 2001

These are notes on the translation from large-step to small-step rules discussed in class on 6 March.

1 A Language Fragment

Here is a small language fragment with enough features to illustrate the translation.

Types

$$\tau \ ::= \ \text{int} \mid \mathbf{var}(\tau) \mid \mathbf{fun}(\tau_1)\tau_2$$

There are numbers, variables and function types.

Expressions

$$\begin{aligned} e \ ::= \ & n \mid a^\tau \\ & \mid e_1 \theta e_2 \\ & \mid (\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3) \\ & \mid e_1 \leftarrow e_2 \mid (e_1 \uparrow) \\ & \mid (\mathbf{let}\ x_1 \sim e_1\ \mathbf{in}\ e_2) \mid x \\ & \mid (\lambda\ x : \tau\ .\ e_1) \mid e_1(e_2) \end{aligned}$$

There are numbers, assignable variables, conditionals, let binding and functions.

Intuitively, all this behaves like the language described in Notes 10. In particular, you should look there for the definitions of the set V of *values* and B of *bindable values*.

2 Typing Rules

The typing rules can almost be taken directly from Notes 10.

Constants

$$\frac{}{\pi \vdash n : \ \mathbf{int}} \tag{T.1}$$

$$\frac{}{\pi \vdash (a^{\tau}) : \ \mathbf{var}(\tau)} \tag{T.2}$$

Operators

$$\frac{\pi \vdash e_1 : \ \tau_1 \qquad \pi \vdash e_2 : \ \tau_2}{\pi \vdash (e_1 \theta e_2) : \tau} \qquad \theta \text{ is } \tau_1 \times \tau_2 \to \tau \tag{T.3}$$

Control Structures

$$\frac{\pi \vdash e_1 : \ \text{bool} \qquad \pi \vdash e_2 : \ \tau \qquad \pi \vdash e_3 : \ \tau}{\pi \vdash (\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3) : \tau} \tag{T.4}$$

Assignable Variables

$$\frac{\pi \vdash e_1 : \ \mathbf{var}(\tau) \qquad \pi \vdash e_2 : \ \tau}{\pi \vdash (e_1 \leftarrow e_2) : \tau} \tag{T.5}$$

$$\frac{\pi \vdash e_1 : \ \mathbf{var}(\tau)}{\pi \vdash (e_1 \uparrow) : \tau} \tag{T.6}$$

Let Bindings

$$\frac{\begin{array}{c}\pi \vdash e_1 : \ \tau_1 \\ (\pi \oplus \{x : \tau_1\}) \vdash e_2 : \tau\end{array}}{\pi \vdash \mathbf{let}\ x \sim e_1\ \mathbf{in}\ e_2) : \tau} \tag{T.7}$$

$$\frac{}{\pi \vdash x : \ \tau} \qquad \text{where } x : \tau \in \pi \tag{T.8}$$

Functions

$$\frac{(\pi \oplus \{x : \tau'\}) \vdash e : \ \tau}{\pi \vdash (\textbf{lambda}\ x : \tau'\ \textbf{dot}\ e) : \ \textbf{fun}(\tau')\tau} \tag{T.9}$$

$$\frac{\pi \vdash e_1 : \ \textbf{fun}(\tau_2)\tau \qquad \pi \vdash e_2 : \tau_2}{\pi \vdash e_1(e_2) : \tau} \tag{T.10}$$

3 Large Step Rules

Here are large step rules – the eager evaluation, static scope version. Again these are almost exactly a subset of the rules from Notes 10.

Constants

$$\frac{}{\langle n, \phi, \sigma\rangle \rightarrow \langle n, \sigma\rangle} \tag{LS.1}$$

$$\frac{}{\langle a^\tau, \phi, \sigma\rangle \rightarrow \langle a^\tau, \sigma\rangle} \tag{LS.2}$$

Operators

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \rightarrow \langle v_1, \sigma_1\rangle \\ \langle e_2, \phi, \sigma_1\rangle \rightarrow \langle v_2, \sigma 2\rangle\end{array}}{\langle e_1 \theta e_2, \phi, \sigma\rangle \rightarrow \langle v, \sigma_2\rangle} \qquad \text{where } v = v_1 \theta v_2 \tag{LS.3}$$

Control Structures

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \rightarrow \langle 0, \sigma_1\rangle \\ \langle e_3, \phi, \sigma_1\rangle \rightarrow \langle v_3, \sigma'\rangle\end{array}}{\langle (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3), \phi, \sigma\rangle \rightarrow \langle v_3, \sigma'\rangle} \tag{LS.4}$$

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \rightarrow \langle v, \sigma_1\rangle \\ \langle e_2, \phi, \sigma_1\rangle \rightarrow \langle v_2, \sigma'\rangle\end{array}}{\begin{array}{c}\langle (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3), \phi, \sigma\rangle \rightarrow \langle v_2, \sigma'\rangle \\ \text{where } v \neq 0\end{array}} \tag{LS.5}$$

Assignable Variables

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \to \langle a^\tau, \sigma_1\rangle \\ \langle e_2, \phi, \sigma_1\rangle \to \langle v_2, \sigma_2\rangle\end{array}}{\langle e_1 \leftarrow e_2, \phi, \sigma\rangle \to \langle v_2, \sigma_2 \oplus \{a^\tau \sim v_2\}\rangle} \tag{LS.6}$$

$$\frac{\langle e_1, \phi, \sigma\rangle \ \to \ \langle a^\tau, \sigma_1\rangle}{\langle e_1 \uparrow, \phi, \sigma\rangle \ \to \ \langle v, \sigma_1\rangle} \qquad \text{where } a^\tau \sim v \in \sigma_1 \tag{LS.7}$$

Let Bindings

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \to \langle v_1, \sigma_1\rangle \\ \langle e_2, (\phi \oplus \{x \sim v_1\}), \sigma 1\rangle \to \langle v, \sigma'\rangle\end{array}}{\langle (\textbf{let } x \sim e_1 \textbf{ in } e_1), \phi, \sigma\rangle \to \langle v, \sigma'\rangle} \tag{LS.8}$$

$$\frac{}{\langle x, \phi, \sigma\rangle \to \langle v, \sigma\rangle} \qquad \text{where}(x \sim v) \in \phi \tag{LS.9}$$

Functions

$$\frac{}{\langle (\textbf{lambda } x : \tau \textbf{ dot } e), \phi, \sigma\rangle \to \langle (\textbf{lambda } x : \tau \textbf{ dot } e), \phi, \sigma\rangle} \quad \text{if } FV(e) \subseteq \{x\} \tag{LS.10}$$

$$\frac{\langle (\textbf{lambda } x : \tau \textbf{ dot } (\textbf{let } y \sim v \textbf{ in } e)), \phi, \sigma\rangle \to \langle v', \sigma'\rangle}{\begin{array}{c}\langle (\textbf{lambda } x : \tau \textbf{ dot } e), \phi, \sigma \to \langle v', \sigma'\rangle \\ \text{where } y \text{ is the least element of } FV(e) - \{x\} \\ \text{and } (y \sim v) \in \phi\end{array}} \tag{LS.11}$$

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \to \langle (\textbf{lambda } x : \tau \textbf{ dot } e_3), \sigma_1\rangle \\ \langle e_2, \phi, \sigma_1\rangle \to \langle v_2, \sigma_2\rangle \\ \langle e_3, (\phi \oplus \{x \sim v_2\}), \sigma_2\rangle \to \langle v, \sigma'\rangle\end{array}}{\langle e_1(e_2), \phi, \sigma\rangle \to \langle v, \sigma'\rangle} \tag{LS.12}$$

4 LS to SS Translation

Here is the nearly-mechanical procedure I outlined in class for generating SS rules from LS rules.

Judgements the LS and SS rules have essentially the same form:

$$\langle e, \phi, \sigma \rangle \rightarrow \langle e', \sigma' \rangle$$

except that for the large step rules e' is always a value, while in the small step rules it can be any expression. In particular, judgements in the small step rules do not need to produce an updated environment. Updates to the environment happen in subgoals (hypotheses), just as in the large step rules. This will (I hope) become clear in the sequel.

Our large step rules explicitly state that a value reduces to itself; that is, rules (LS.1), (LS.2) and (LS.10) of the previous section together are equivalent to the single rule

$$\frac{}{\langle v, \phi, \sigma \rangle \rightarrow \langle v, \sigma \rangle}$$

Such rules do not appear in the small step semantics – instead, the notion of a complete evaluation is defined as a sequence of steps terminating in a value.

Of the remaining large step rules, nearly every one has the following form, which we shall call a "type (A) rule" below:

$$\frac{\begin{array}{l}\langle e_1, \phi_1, \sigma_0 \rangle \rightarrow \langle v_1, \sigma_1 \rangle \\ \quad \cdots \\ \langle e_i, \phi_i, \sigma_{i-1} \rangle \rightarrow \langle v_i, \sigma_i \rangle \\ \quad \cdots \\ \langle e_k, \phi_k, \sigma_{k-1} \rangle \rightarrow \langle v_k, \sigma_k \rangle\end{array}}{\langle C[e_1, \ldots, e_k]\phi, \sigma \rangle \rightarrow \langle v, \sigma' \rangle} \qquad \text{(A)}$$

where

$$\begin{array}{l}\sigma = \sigma_0 \qquad \text{and} \qquad \sigma' = S(\sigma_k, \{\ldots v_j \ldots\}) \\ v = F(\phi, \sigma_k, \{\ldots v_j \ldots\}) \\ \phi_i = H(\phi, \sigma_{i-1}, \{v_j | j < i\})\end{array}$$

Here S, F and H are functions with only the explicitly specified dependencies.

In general, rules have *instantiation conditions*, such as

$$\text{where } v = v_1 \theta v_2$$

in (LS.3), or

$$\text{where } (a^\tau \sim v) \in \sigma_1$$

in (LS.7). Well, we did say the translation was *almost* mechanical – it will usually, but not always, be obvious how such conditions should be inherited by the small step rules we generate.

The way the stores σ_i are used in the hypotheses of a type (A) rule – the i^{th} hypothesis evaluates e_i taking the store from σ_{i-1} to σ_i – enforces an evaluation order on the subexpressions. We have written the "constructor"

$$C[e_1, \ldots, e_k]$$

(which is almost always a single constructor of the language syntax) to make this evaluation order explicit. In addition, we require that each expression e_i actually occurs as a subexpression of $C[e_1, \ldots, e_k]$, so that from an instance of the constructor we can uniquely identify the e_i.

Each type (A) large step rule is translated to a collection of small step rules of two forms.

The first form assumes the first $(i-1)$ subexpressions have been completely evaluated, and takes a single step in evaluating the i^{th} subexpression:

$$\frac{\langle e_i, \phi_i, \sigma\rangle \rightarrow_1 \langle e_i', \sigma_i'\rangle}{\begin{array}{l}\langle C[v_1, \ldots, v_{i-1}, e_i, e_{i+1}, \ldots, e_k], \phi, \sigma\rangle \\ \qquad \rightarrow_1 \langle C[v_1, \ldots, v_{i-1}, e_i', e_{i+1}, \ldots, e_k], \sigma_i'\rangle\end{array}} \tag{A1}$$

Such rules eventually reduce $C[\ldots]$ so that all subexpressions are values. A second form of translated rule applies in this case:

$$\frac{}{\begin{array}{l}\langle C[v_1, \ldots, v_k], \phi, \sigma\rangle \rightarrow_1 \langle v, \sigma'\rangle \\ \qquad \text{where } \sigma' = S(\sigma_k, \{\ldots v_j \ldots\}) \;\text{ and }\; v = F(\phi, \sigma_k, \{\ldots v_j \ldots\})\end{array}} \tag{A2}$$

Such a rule produces a final value v and eliminates the occurrence of the constructor C.

In our language fragment there are only a few rules that depart from the general type (A) form.

First are the rules for conditionals. Consider Rule (LS.4):

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \rightarrow \langle 0, \sigma_1\rangle \\ \langle e_3, \phi, \sigma_1\rangle \rightarrow \langle v_3, \sigma'\rangle\end{array}}{\langle (\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3), \phi, \sigma\rangle \rightarrow \langle v_3, \sigma'\rangle}$$

The **if** statement has three subexpressions, but the rule hypotheses evaluate only two of them. The value of the condition subexpression determines *which*

two (and thus determines whether rule (LS.4) or (LS.5) is applicable). The translation above does not quite work on these rules, because at some point either the **then** clause e_2 or the **else** clause e_3 needs to disappear, and our translation never does this. The problem is not difficult to fix, however. You'll be asked to do so below.

A second departure from type (A) form is the rule (LS.11) for evaluating a **lambda** expression:

$$\frac{\langle(\textbf{lambda}\ x:\tau\ \textbf{dot}\ (\textbf{let}\ y \sim v\ \textbf{in}\ e)), \phi, \sigma\rangle \rightarrow \langle v', \sigma'\rangle}{\langle(\textbf{lambda}\ x:\tau\ \textbf{dot}\ e), \phi, \sigma \rightarrow \langle v', \sigma'\rangle} \quad \begin{array}{c}\text{where } y \text{ is the least element of } FV(e) - \{x\} \\ \text{and } (y \sim v) \in \phi\end{array}$$

Here the hypothesis evaluates an expression that is *not* a subexpression of the conclusion. This can be viewed as an instance of a more general form

$$\frac{\langle G(e, \phi, \sigma), \phi, \sigma\rangle \rightarrow \langle v, \sigma'\rangle}{\langle e, \phi, \sigma\rangle \rightarrow \langle v, \sigma'\rangle} \tag{B}$$

which not surprisingly we shall call a "type (B)" rule. The intuition is that the transformed expression $G(e, \phi, \sigma)$ is completely equivalent to e for evaluation in ϕ and σ, and the transformation from e to $G(e, \phi, \sigma)$ makes "progress." (For the **lambda** expression rule, "progress" means descending in the well founded ordering defined by subset inclusion on the set of free variables of the **lambda** expression.)

To translate a type (B) rule, we simply add a small step rule that performs the transformation from e to $G(e, \phi, \sigma)$ as an evaluation step:

$$\frac{}{\langle e, \phi, \sigma\rangle \rightarrow_1 \langle G(e, \phi, \sigma), \phi, \sigma\rangle} \tag{B1}$$

The final departure from type (A) is rule (LS.12) for applying a **lambda** expression:

$$\frac{\begin{array}{c}\langle e_1, \phi, \sigma\rangle \rightarrow \langle(\textbf{lambda}\ x:\tau\ \textbf{dot}\ e_3), \sigma_1\rangle \\ \langle e_2, \phi, \sigma_1\rangle \rightarrow \langle v_2, \sigma_2\rangle \\ \langle e_3, (\phi \oplus \{x \sim v_2\}), \sigma_2\rangle \rightarrow \langle v, \sigma'\rangle\end{array}}{\langle e_1(e_2), \phi, \sigma\rangle \rightarrow \langle v, \sigma'\rangle}$$

This rule fails to be type (A) for the subtle reason that e_3, the function body, does not appear in the conclusion of the rule, hence cannot occur as a subexpression of $C[e_1, e_2, e_3]$ as required for the translation to be used. We'll deal with this as a special case in the next section.

5 Small Step Rules

Here we present small step rules constructed using (A) and (B) from the previous section. The rule numbering will follow that for the large step rules.

Constants. These rules define values; there are no corresponding rules in the SS semantics.

Operators. Rule (LS.3) is an instance of (A). So by (A1) we generate the rules

$$\frac{\langle e_1, \phi, \sigma\rangle \rightarrow_1 \langle e_1', \sigma'\rangle}{\langle (e_1 \theta e_2), \phi, \sigma\rangle \rightarrow_1 \langle (e_1' \theta e_2), \phi, \sigma'\rangle} \qquad \text{(SS.3.A1.1)}$$

and

$$\frac{\langle e_2, \phi, \sigma\rangle \rightarrow_1 \langle e_2', \sigma'\rangle}{\langle (v_1 \theta e_2), \phi, \sigma\rangle \rightarrow_1 \langle (v_1 \theta e_2'), \phi, \sigma'\rangle} \qquad \text{(SS.3.A1.2)}$$

By (A2) we generate the single rule

$$\frac{}{\langle (v_1 \theta v_2), \phi, \sigma\rangle \rightarrow_1 \langle v, \phi, \sigma\rangle} \quad \text{where } v = (v_1 \theta v_2) \qquad \text{(SS.3.A2.1)}$$

Note the condition $v = (v_1 \theta v_2)$ from (LS.3), which is inherited by this rule but not necessary for rules (SS.3.A1.*).

Control Structures. Rules (LS.4) and (LS.5), do not quite fit form (A), but their translations are not difficult. They are left as exercises.

Assignable Variables. Rule (LS.6) is type (A), and its translation is left as a straightforward exercise. Rule (SS.7) is also type (A), but with an instantiation condition. The translations are

$$\frac{\langle e_1, \phi, \sigma\rangle \rightarrow_1 \langle e_1', \sigma'\rangle}{\langle (e_1 \uparrow), \phi, \sigma\rangle \rightarrow_1 \langle (e_1' \uparrow), \sigma'\rangle} \qquad \text{(SS.7.A1.1)}$$

$$\frac{}{\langle (v_1 \uparrow), \phi, \sigma\rangle \rightarrow_1 \langle v, \sigma'\rangle} \qquad \text{where}(v_1 \sim v) \in \sigma \qquad \text{(SS.7.A2.2)}$$

Note the instantiation condition "where $(v_1 \sim v) \in \sigma$" is inherited from (LS.7) in rule (SS.7.A2.2), but is not relevant to rule (SS.7.A2.1).

Let Bindings. Rule (LS.8) is also type (A):

$$\frac{\langle e_1, \phi, \sigma\rangle \rightarrow_1 \langle e_1', \sigma'\rangle}{\langle (\textbf{let } x \sim \ e_1 \textbf{ in } e_2), \phi, \sigma\rangle \rightarrow_1 \langle (\textbf{let } x \sim \ e_1' \textbf{ in } e_2), \sigma'\rangle} \quad \text{(SS.8.A1.1)}$$

$$\frac{\langle e_2, (\phi \oplus \{x \sim v_1\}), \sigma\rangle \rightarrow_1 \langle e_2', \sigma'\rangle}{\langle (\textbf{let } x \sim \ v_1 \textbf{ in } e_2), \phi, \sigma\rangle \rightarrow_1 \langle (\textbf{let } x \sim \ v_1 \textbf{ in } e_2'), \sigma'\rangle} \quad \text{(SS.8.A1.2)}$$

$$\frac{}{\langle (\textbf{let } x \sim \ v_1 \textbf{ in } v_2), \phi, \sigma\rangle \rightarrow_1 \langle v2, \sigma\rangle} \quad \text{(SS.8.A2.1)}$$

Rule (LS.9) also fits form (A), though this may be a bit counterintuitive. Note there is no subexpression in the constructor – it consists of a single identifier. Think of it as a 0-ary constructor

$$C_x[\,] \ \equiv \ x$$

Embedding the identifier x into the constructor is not any different here from the previous rule, where the constructor

$$C_x[e_1, e_2] \ \equiv \ (\textbf{let } x \sim e_1 \textbf{ in } e_2)$$

also included x.

Since (LS.9) has no subexpressions and no hypotheses, we generate the single rule

$$\frac{}{\langle x, \phi, \sigma\rangle \rightarrow_1 \langle v, \sigma\rangle} \qquad \text{where}(x \sim v) \in \sigma \quad \text{(SS.9.A2.1)}$$

Functions. Rule (LS.10) defines function values; there is no corresponding rule in the SS semantics.

Rule (LS.11) was the motivation for our definition of type (B) above. By (B.1) it gives rise to the small step rule

$$\frac{}{\begin{array}{c}\langle (\textbf{lambda } x : \tau \textbf{ dot } e), \phi, \sigma\rangle \\ \rightarrow_1 \langle (\textbf{lambda } x : \tau \textbf{ dot } (\textbf{let } y \sim v \textbf{ in } e)), \sigma\rangle \\ \text{where } y \text{ is the least element of } FV(e) - \{x\} \\ \text{and } (y \sim v) \in \phi\end{array}} \quad \text{(SS.11.B.1)}$$

Again note the instantiation condition inherited from (LS.11).

Finally we reach rule (LS.12), which defines function application. This rule fails to be type (A) because of the function body, which does not occur in the conclusion of the rule. We shall treat e_1 and e_2 just as if the entire rule fit form (A). Then we'll use the knowledge that e_3 must appear inside the function value v_1 to split the remainder of the function application into a pair of *ad hoc* rules.

$$\frac{\langle e_1, \phi, \sigma\rangle \rightarrow_1 \langle e_1', \sigma'\rangle}{\langle e_1(e_2), \phi, \sigma\rangle \rightarrow_1 \langle e_1'(e_2), \sigma'\rangle} \tag{SS.12.A1.1}$$

$$\frac{\langle e_2, \phi, \sigma\rangle \rightarrow_1 \langle e_2', \sigma'\rangle}{\langle v(e_2), \phi, \sigma\rangle \rightarrow_1 \langle v_1(e_2'), \sigma'\rangle} \tag{SS.12.A1.2}$$

$$\frac{\langle e_3, (\phi \oplus \{x \sim v_2\}), \sigma\rangle \rightarrow_1 \langle e_3', \sigma'\rangle}{\begin{array}{l}\langle(\textbf{lambda}\ x : \tau\ \textbf{dot}\ e_3)(v_2), \phi, \sigma\rangle \\ \qquad \rightarrow_1 \langle(\textbf{lambda}\ x : \tau\ \textbf{dot}\ e_3')(v_2), \sigma'\rangle\end{array}} \tag{SS.12.1}$$

$$\frac{}{\langle(\textbf{lambda}\ x : \tau\ \textbf{dot}\ v_3)(v_2), \phi, \sigma\rangle \rightarrow_1 \langle v_3, \sigma\rangle} \tag{SS.12.2}$$

This completes the small step rule translation.

6 Collected Exercises

Exercise 1: Give the translation of (LS6).

Exercise 2: Give the translation of (LS4).

Exercise 3: Show that the LS and SS semantics agree by evaluating the expression

$$\begin{array}{l}\textbf{let}\ x \sim a_1^{\text{int}}\ \textbf{in} \\ \quad (\textbf{lambda}\ y : \text{int}\ \textbf{dot}\ y + (x\uparrow))(\\ \qquad (\textbf{let}\ x \sim 7\ \textbf{in}\ (a_1^{\text{int}} \leftarrow x)))\end{array}$$

using both sets of rules.