Supplementary Notes on An Abstract Machine

15-312: Foundations of Programming Languages
Frank Pfenning
modified by Jonathan Aldrich

Lecture 8
Sep 18, 2003

In this lecture we introduce a somewhat lower-level semantics for MinML in the form of an *abstract machine* [Ch 11]. In this machine we make the control flow explicit, rather than encoding it in the search rules as in the first operational semantics. Besides getting closer to an actual implementation, it will allow us to easily define constructs to capture the current continuation [Ch. 12].

Abstract machines have recently gained in popularity through the ascendency of the Java programming language. The standard model is that we compile Java source to Java bytecode, which may be transmitted over networks (for example, as an "applet"), and then interpreted via the Java abstract machine. The use of an abstract machine here plays two important roles: (1) the byte code is portable to any architecture with an interpreter, and (2) the received code can be easily checked for illegal operations. This is type-checking of the abstract machine code goes hand in hand with some residual checking that has to go on while the code is interpreted. Note that traditional type-checking as we have discussed it so far needs to be augmented significantly, for example, to prevent the normally type-safe operation of reformatting the hard disk.

The kind of abstract machine we present here is a variant of the C-machine [Ch 11.1] with two kinds of states: those that attempt to evaluate an expression, and those that return a value that has been computed. Its main component, however, is the same: a run-time stack that records what remains to be done after the current subexpression has been fully evaluated. The stack consists of frames which represents the action to be taken by the abstract machine once the current expression has been evaluated.

We treat here the fragment with pairs, functions, and booleans (see [Ch 11.1] for a treatment of primitive operators).

We begin by defining the syntax in the form of (abstract syntax) grammar. As we have seen before, this can also be written in the form of judgments. When we use v we imply that v must be a value.

States	s	$::=$	$k > e$	evaluate e under k
		$\mid$	$k < v$	return v to k
Stacks	k	$::=$	$\bullet$	empty stack
		$\mid$	$k \triangleright f$	stack k with top f
Frames	f	$::=$	$\mathtt{o}(\Box, e_2) \mid \mathtt{o}(v_1, \Box)$	primops
		$\mid$	$\mathtt{pair}(\Box, e_2) \mid \mathtt{pair}(v_1, \Box)$	pairs
		$\mid$	$\mathtt{fst}(\Box) \mid \mathtt{snd}(\Box)$	projections
		$\mid$	$\mathtt{apply}(\Box, e_2) \mid \mathtt{apply}(v_1, \Box)$	applications
		$\mid$	$\mathtt{if}(\Box, e_1, e_2)$	conditional

A hole $\Box$ in the top stack frame is intended to hold the value returned by evaluation of the current expression. It corresponds to the place in an expression where evaluation can take place and thus implements the search rules of the structured operational semantics.

The main judgment defining the abstract machine is

$$s \mapsto_{\mathsf{c}} s'$$

expressing that state s makes a transition to state s' in one step. The initial state of the machine has the form $\bullet > e$, a final state has the form $\bullet < v$. In general, we define our machine so that if

$$e = e_1 \mapsto \cdots \mapsto e_n = v$$

according to our operational semantics then for any stack k which should have

$$k > e \mapsto_{\mathsf{c}} \cdots \mapsto_{\mathsf{c}} k < v$$

As we will see, the operational semantics and the abstract machines do not take the same number of steps. This is because the operational semantics does not step at all for values, while the abstract machine will take some steps to go from $k > v$ to $k < v$.

We now give the transitions, organized by the type structure of the language.

Integers.

$$\begin{array}{lcl}
k > \mathtt{num}(n) & \mapsto_{\mathsf{c}} & k < \mathtt{num}(n) \\
\\
k > \mathtt{o}(e_1, e_2) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{o}(\Box, e_2) > e_1 \\
k \triangleright \mathtt{o}(\Box, e_2) < v_1 & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{o}(v_1, \Box) > e_2 \\
k \triangleright \mathtt{o}(\mathtt{num}(n_1), \Box) < \mathtt{num}(n_2) & \mapsto_{\mathsf{c}} & k < \mathtt{num}(n) \\
 & & \quad (n = f_o(n_1, n_2))
\end{array}$$

Products.

$$\begin{array}{lcl}
k > \mathtt{pair}(e_1, e_2) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{pair}(\Box, e_2) > e_1 \\
k \triangleright \mathtt{pair}(\Box, e_2) < v_1 & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{pair}(v_1, \Box) > e_2 \\
k \triangleright \mathtt{pair}(v_1, \Box) < v_2 & \mapsto_{\mathsf{c}} & k < \mathtt{pair}(v_1, v_2) \\
\\
k > \mathtt{fst}(e) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{fst}(\Box) > e \\
k \triangleright \mathtt{fst}(\Box) < \mathtt{pair}(v_1, v_2) & \mapsto_{\mathsf{c}} & k < v_1 \\
\\
k > \mathtt{snd}(e) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{snd}(\Box) > e \\
k \triangleright \mathtt{snd}(\Box) < \mathtt{pair}(v_1, v_2) & \mapsto_{\mathsf{c}} & k < v_2
\end{array}$$

Functions.

$$\begin{array}{lcl}
k > \mathtt{fn}(\tau, x.e) & \mapsto_{\mathsf{c}} & k < \mathtt{fn}(\tau, x.e) \\
\\
k > \mathtt{apply}(e_1, e_2) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{apply}(\Box, e_2) > e_1 \\
k \triangleright \mathtt{apply}(\Box, e_2) < v_1 & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{apply}(v_1, \Box) > e_2 \\
k \triangleright \mathtt{apply}(v_1, \Box) < v_2 & \mapsto_{\mathsf{c}} & k > \{v_2/x\}e \\
 & & \quad (v_1 = \mathtt{fn}(\tau, x.e))
\end{array}$$

Recursion.

$$k > \mathtt{rec}(\tau, x.e) \quad \mapsto_{\mathsf{c}} \quad k > \{\mathtt{rec}(\tau, x.e)/x\}e$$

Conditionals.

$$\begin{array}{lcl}
k > \mathtt{true} & \mapsto_{\mathsf{c}} & k < \mathtt{true} \\
k > \mathtt{false} & \mapsto_{\mathsf{c}} & k < \mathtt{false} \\
k > \mathtt{if}(e, e_1, e_2) & \mapsto_{\mathsf{c}} & k \triangleright \mathtt{if}(\Box, e_1, e_2) > e \\
k \triangleright \mathtt{if}(\Box, e_1, e_2) < \mathtt{true} & \mapsto_{\mathsf{c}} & k > e_1 \\
k \triangleright \mathtt{if}(\Box, e_1, e_2) < \mathtt{false} & \mapsto_{\mathsf{c}} & k > e_2
\end{array}$$

As an example, consider the evaluation of

```
(fn x:int => x) 0
```

$$\begin{array}{lllll}
 & \bullet & > & \mathtt{apply}(\mathtt{fn}(\mathtt{int}, x.x), \mathtt{num}(0)) \\
\mapsto_{\mathsf{c}} & \bullet \triangleright \mathtt{apply}(\square, \mathtt{num}(0)) & > & \mathtt{fn}(\mathtt{int}, x.x) \\
\mapsto_{\mathsf{c}} & \bullet \triangleright \mathtt{apply}(\square, \mathtt{num}(0)) & < & \mathtt{fn}(\mathtt{int}, x.x) \\
\mapsto_{\mathsf{c}} & \bullet \triangleright \mathtt{apply}(\mathtt{fn}(\mathtt{int}, x.x), \square) & > & \mathtt{num}(0) \\
\mapsto_{\mathsf{c}} & \bullet \triangleright \mathtt{apply}(\mathtt{fn}(\mathtt{int}, x.x), \square) & < & \mathtt{num}(0) \\
\mapsto_{\mathsf{c}} & \bullet & > & \mathtt{num}(0) \\
\mapsto_{\mathsf{c}} & \bullet & < & \mathtt{num}(0)
\end{array}$$

Note that in the second-to-last step, $\{\mathtt{num}(0)/x\}x = \mathtt{num}(0)$

Proving the correctness of the C-machine is complicated by the fact that the two machines step at different rates. We further have to account for the stack. However, in the overall statement of the correctness theorem, these problems may not be apparent. In order to state the theorem, we first define the multi-step versions of the two transition judgments. This is just the reflexive and transitive closure of the single-step relation. We only define this formally for the abstract machine; other transition relations can similarly be extended to multiple steps [Ch. 2].

$s \mapsto_{\mathsf{c}}^{*} s'$ s steps to s' in zero or more steps

$$\frac{}{s \mapsto_{\mathsf{c}}^{*} s}\ \text{refl} \qquad \frac{s \mapsto_{\mathsf{c}} s' \quad s' \mapsto_{\mathsf{c}}^{*} s''}{s \mapsto_{\mathsf{c}}^{*} s''}\ \text{step}$$

We take certain elementary properties of the multi-step transition relation for granted and use them tacitly. We give here only one, as an example.

Theorem 1 (Transitivity)
If $s \mapsto_{\mathsf{c}}^{*} s'$ and $s' \mapsto_{\mathsf{c}}^{*} s''$ then $s \mapsto_{\mathsf{c}}^{*} s''$.

Proof: By straightforward rule induction on the derivation of $s \mapsto_{\mathsf{c}}^{*} s'$. ■

Theorem 2 (Correctness of C-Machine)
$e \mapsto^{*} v$ if and only if $\bullet > e \mapsto_{\mathsf{c}}^{*} \bullet < v$

As usual, we cannot prove this directly, but we need to generalize it. In this case we also need two lemmas.

Lemma 3 (Determinism)
If $s \mapsto_{\mathsf{c}} s'$ and $s \mapsto_{\mathsf{c}} s''$ then $s' = s''$.

Proof: By cases on the two given judgments. This is a degenerate case of rule induction, since the $\mapsto_{\mathsf{c}}$ judgment is defined only by axioms. ■

Lemma 4 (Value Computation)
(i) $k > v \mapsto_{\mathsf{c}}^{*} k < v$

(ii) If $k > v \mapsto_{\mathsf{c}}^{*} \bullet < a$ then the computation decomposes into $k > v \mapsto_{\mathsf{c}}^{*} k < v$ and $k < v \mapsto_{\mathsf{c}}^{*} \bullet < a$

Proof: Part (i) follows by induction on the structure of v.[1] Part (ii) then follows from part (i) by determinism. We show the proof of part (i) in detail.

Cases: $v = \mathtt{num}(n)$, $v = \mathtt{true}$, $v = \mathtt{false}$, or $v = \mathtt{fn}(\tau, x.e)$. Then the result is immediate by a single step of the abstract machine.

Case: $v = \mathtt{pair}(v_1, v_2)$. Then

$k > \mathtt{pair}(v_1, v_2)$	
$\mapsto_{\mathsf{c}} k \triangleright \mathtt{pair}(\Box, v_2) > v_1$	By rule
$\mapsto_{\mathsf{c}}^{*} k \triangleright \mathtt{pair}(\Box, v_2) < v_1$	By i.h. on v_1
$\mapsto_{\mathsf{c}} k \triangleright \mathtt{pair}(v_1, \Box) > v_2$	By rule
$\mapsto_{\mathsf{c}}^{*} k \triangleright \mathtt{pair}(v_1, \Box) < v_2$	By i.h. on v_2
$\mapsto_{\mathsf{c}} k < \mathtt{pair}(v_1, v_2)$	By rule

■

Now we are in a position to prove the generalization that directly relates a single step in the original semantics to possibly several steps in the C-machine. It is difficult to explain how one might arrive at this generalization, except to say "through experience" and by analysing the failure of other attempts. We express that if $e \mapsto e'$, then under any stack k, if the evaluation of e' yields the final answer a, then the evaluation of e also yields the final answer a.

[1]Equivalently, we could say: *By rule induction on the derivation of v* value.

Lemma 5 (Completeness Lemma for the C-Machine)
If $e \mapsto e'$ *and* $k > e' \mapsto_{\mathsf{c}}^{*} \bullet < a$ *then* $k > e \mapsto_{\mathsf{c}}^{*} \bullet < a$.

Proof: The proof is by rule induction on the derivation of $e \mapsto e'$.

Below, when we claim a step follow *"by inversion"* it is because exactly one of the rules could be applied as the first step. Technically, this is an inversion on the definition of $\mapsto_{\mathsf{c}}^{*}$ (rule step must have been applied), followed by an second inversion on the (single) first step that could have been taken.

We show only the cases for products, since all other cases follow a similar pattern.

For the search rules, we apply inversion until we have uncovered a subcomputation of the abstract machine to which we can apply the induction hypothesis. Then we reconstitute the full computation.

For the reduction rules, we directly construct the needed computation, possibly applying to the value computation lemma, part (i).

Case:

$$\frac{e_1 \mapsto e_1'}{\mathtt{pair}(e_1, e_2) \mapsto \mathtt{pair}(e_1', e_2)}$$

$e_1 \mapsto e_1'$	Subderivation
$k > \mathtt{pair}(e_1', e_2) \qquad \mapsto_{\mathsf{c}}^{*} \bullet < a$	Assumption
$k > \mathtt{pair}(e_1', e_2) \mapsto_{\mathsf{c}} k \triangleright \mathtt{pair}(\Box, e_2) > e_1' \mapsto_{\mathsf{c}}^{*} \bullet < a$	By inversion
$k \triangleright \mathtt{pair}(\Box, e_2) > e_1 \mapsto_{\mathsf{c}}^{*} \bullet < a$	By i.h.
$k > \mathtt{pair}(e_1, e_2) \mapsto_{\mathsf{c}} k \triangleright \mathtt{pair}(\Box, e_2) > e_1 \mapsto_{\mathsf{c}}^{*} \bullet < a$	By rule

Case:

$$\frac{v_1\ \mathsf{value} \quad e_2 \mapsto e_2'}{\mathtt{pair}(v_1, e_2) \mapsto \mathtt{pair}(v_1, e_2')}$$

$e_1 \mapsto e_1'$	Subderivation
$k > \mathtt{pair}(v_1, e_2') \mapsto^{*} \bullet < a$	Assumption
$k > \mathtt{pair}(v_1, e_2') \mapsto k \triangleright \mathtt{pair}(\Box, e_2') > v_1 \mapsto^{*} \bullet < a$	By inversion
$k \triangleright \mathtt{pair}(\Box, e_2') > v_1 \mapsto^{*} k \triangleright \mathtt{pair}(\Box, e_2') < v_1 \mapsto^{*} \bullet < a$	By value computation (ii)
$k \triangleright \mathtt{pair}(\Box, e_2') < v_1 \mapsto k \triangleright \mathtt{pair}(v_1, \Box) > e_2' \mapsto^{*} \bullet < a$	By inversion
$k \triangleright \mathtt{pair}(v_1, \Box) > e_2 \mapsto^{*} \bullet < a$	By i.h.
$k \triangleright \mathtt{pair}(\Box, e_2) < v_1 \mapsto^{*} \bullet < a$	By rule

$k \triangleright \texttt{pair}(\Box, e_2) > v_1 \mapsto^* \bullet < a$ By value computation (i)
$k > \texttt{pair}(v_1, e_2) \mapsto k \triangleright \texttt{pair}(\Box, e_2) > v_1 \mapsto^* \bullet < a$ By rule

Case:

$$\frac{e_1 \mapsto e_1'}{\texttt{fst}(e_1) \mapsto \texttt{fst}(e_1')}$$

$e_1 \mapsto e_1'$ Subderivation
$k > \texttt{fst}(e_1') \mapsto_{\mathsf{c}}^* \bullet < a$ Assumption
$k > \texttt{fst}(e_1') \mapsto_{\mathsf{c}} k \triangleright \texttt{fst}(\Box) > e_1' \mapsto_{\mathsf{c}}^* \bullet < a$ By inversion
$k \triangleright \texttt{fst}(\Box) > e_1 \mapsto_{\mathsf{c}}^* \bullet < a$ By i.h.
$k > \texttt{fst}(e_1) \mapsto_{\mathsf{c}} k \triangleright \texttt{fst}(\Box) > e_1 \mapsto_{\mathsf{c}}^* \bullet < a$ By rule

Case:

$$\frac{v_1 \;\mathsf{value} \quad v_2 \;\mathsf{value}}{\texttt{fst}(\texttt{pair}(v_1, v_2)) \mapsto v_1}$$

$k < v_1 \mapsto_{\mathsf{c}}^* \bullet < a$ Assumption

$k > \texttt{fst}(\texttt{pair}(v_1, v_2))$
$\mapsto_{\mathsf{c}} k \triangleright \texttt{fst}(\Box) > \texttt{pair}(v_1, v_2)$ By rule
$\mapsto_{\mathsf{c}}^* k \triangleright \texttt{fst}(\Box) < \texttt{pair}(v_1, v_2)$ By value computation (i)
$\mapsto_{\mathsf{c}} k < v_1$ By rule
$\mapsto_{\mathsf{c}}^* \bullet < a$ By assumption

Case:

$$\frac{v_1 \;\mathsf{value} \quad v_2 \;\mathsf{value}}{\texttt{snd}(\texttt{pair}(v_1, v_2)) \mapsto v_2}$$

$k < v_2 \mapsto_{\mathsf{c}}^* \bullet < a$ Assumption

$k > \texttt{snd}(\texttt{pair}(v_1, v_2))$
$\mapsto_{\mathsf{c}} k \triangleright \texttt{snd}(\Box) > \texttt{pair}(v_1, v_2)$ By rule
$\mapsto_{\mathsf{c}}^* k \triangleright \texttt{snd}(\Box) < \texttt{pair}(v_1, v_2)$ By value computation (i)
$\mapsto_{\mathsf{c}} k < v_2$ By rule
$\mapsto_{\mathsf{c}}^* \bullet < a$ By assumption

■

We do not show the proof in the other direction, which is a minor variant of the one in [Ch 11.1]. We now return to the correctness theorem.

Theorem 6 (Correctness of C-Machine)

(i) If $e \mapsto^* v$ *then* $\bullet > e \mapsto_{\mathsf{c}}^* \bullet < v$.

(ii) If $\bullet > e \mapsto_{\mathsf{c}}^* \bullet < v$ *then* $e \mapsto^* v$.

Proof: We show part (i) and omit part (ii) (see [Ch 11.1]). The proof of part (i) is by induction on the derivation of $e \mapsto^* v$.

Case:

$$\frac{}{v \mapsto^* v}\ \text{refl}$$

$\bullet > v \mapsto_{\mathsf{c}}^* \bullet < v$ By value computation (i)

Case:

$$\frac{e \mapsto e' \quad e' \mapsto^* v}{e \mapsto^* v}\ \text{step}$$

$\bullet > e' \mapsto_{\mathsf{c}}^* \bullet < v$ By i.h.

$\bullet > e \mapsto_{\mathsf{c}}^* \bullet < v$ By completeness lemma

■