Supplementary Notes on Storage Management

15-312: Foundations of Programming Languages
Frank Pfenning

Lecture 27
December 5, 2002

In this lecture we discuss issues of storage managment and garbage collection. The lecture follows [Ch. 31] rather closely, so we concentrate on what is slightly different here, namely the presentation of bisimulation.

In order to talk about garbage collection, we need to formalize the distinction between *small values* and *large values*, where small values can be part of the stack, while large values must be allocated in the heap. For the purpose of this lecture, small values are either integers, booleans, or "pointers" to large values. Functions are large values, as are pairs. We use the judgments v svalue and v lvalue for small and large values, respectively. Pointers are represented by a new kind of value l, standing for locations in memory. Unlike with mutable storage, such locations cannot be changed, but they can be allocated (implicitly) and deallocated (by garbage collection).

$$\frac{}{\mathsf{int}(n)\ \mathsf{svalue}} \qquad \frac{}{l\ \mathsf{svalue}}$$

$$\frac{v_1\ \mathsf{svalue} \quad v_2\ \mathsf{svalue}}{\mathsf{pair}(v_1, v_2)\ \mathsf{lvalue}} \qquad \frac{}{\lambda x.e\ \mathsf{lvalue}}$$

Note that the components of a pair are small values. This means they must either be integers, booleans, or again pointers.

Heaps are simply locations together with their (immutable) values.

$$\text{Heaps} \quad H \quad ::= \quad \cdot \mid H, l{=}v$$

As usual, all location l must be distinct. Furthermore, values stored in the heap must be large values, that is, if $l{=}v$ is part of the heap, then v lvalue.

We describe a heap-based operational semantics using the A-machine, which is an extension of the C-machine to account for the heap. Recall:

$$\begin{array}{rrcl} \text{Stacks} & K & ::= & \bullet \mid K \triangleright \mathsf{apply}(\Box, e_2) \mid K \triangleright \mathsf{apply}(v_1, \Box) \\ & & \mid & K \triangleright \mathsf{pair}(\Box, e_2) \mid K \triangleright \mathsf{pair}(v_1, \Box) \mid K \triangleright \mathsf{fst}(\Box) \mid K \triangleright \mathsf{snd}(\Box) \end{array}$$

$$\begin{array}{rrcl} \text{States} & s & ::= & K > e \mid K < v \end{array}$$

We restrict stacks and states to contain only small values. If an expression e is stored on the stack or in the process of being evaluated it has not yet been turned into a value and therefore does not have to satisfy this criterion. We show only a few rules which formalize this intuition. We ignore issues of typing, since they are largely orthogonal and basically unchanged from the typing of the C-machine.

$$\frac{K\ \mathsf{stack} \quad e\ \mathsf{exp}}{K > e\ \mathsf{state}} \qquad \frac{K\ \mathsf{stack} \quad v\ \mathsf{svalue}}{K > v\ \mathsf{state}}$$

$$\frac{K\ \mathsf{stack} \quad e_2\ \mathsf{exp}}{K \triangleright \mathsf{apply}(\Box, e_2)\ \mathsf{stack}} \qquad \frac{K\ \mathsf{stack} \quad v_1\ \mathsf{svalue}}{K \triangleright \mathsf{apply}(v_1, \Box)\ \mathsf{stack}}$$

$$\frac{}{\cdot\ \mathsf{heap}} \qquad \frac{H\ \mathsf{heap} \quad v\ \mathsf{lvalue} \quad (l \notin \mathrm{dom}(H))}{H, l{=}v\ \mathsf{heap}}$$

A machine state is now extended by a heap, written as $H; s$, where s is either $K > e$ or $K < v$. There are several invariants we will want to maintain of machine states. For example, it should be *self-contained*. If we denote the location defined by a heap with $\mathrm{dom}(H)$ and the free locations in a term (that is, expression, value, stack, or values defined in a heap) by $\mathrm{FL}(\_)$, the $H; s$ is self-contained if $\mathrm{FL}(H) \cup \mathrm{FL}(s) \subseteq \mathrm{dom}(H)$. For other invariants, see [Ch. 31].

The transitions of the A-machine can now be developed in analogy with the C-machine, keeping in mind that we need to maintain the distinction between small and large values.

$$\begin{array}{lcll} H; K > \mathsf{pair}(e_1, e_2) & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{pair}(\Box, e_2) > e_1 & \\ H; K \triangleright \mathsf{pair}(\Box, e_2) < v_1 & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{pair}(v_1, \Box) > e_2 & \\ H; K \triangleright \mathsf{pair}(v_1, \Box) < v_2 & \mapsto_{\mathsf{a}} & H, l{=}\mathsf{pair}(v_1, v_2); K < l & (l \notin \mathrm{dom}(H)) \\ H; K > \mathsf{fst}(e) & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{fst}(\Box) > e & \\ H; K \triangleright \mathsf{fst}(\Box) < l & \mapsto_{\mathsf{a}} & H; K < v_1 & (l{=}\mathsf{pair}(v_1, v_2) \in H) \\ H; K > \mathsf{snd}(e) & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{snd}(\Box) > e & \\ H; K \triangleright \mathsf{snd}(\Box) < l & \mapsto_{\mathsf{a}} & H; K < v_2 & (l{=}\mathsf{pair}(v_1, v_2) \in H) \end{array}$$

It is easy to verify inductively that the value size invariants for heaps and stacks are preserved by these rules. We finish with the rules for functions.

$$
\begin{array}{lcll}
H; K > \mathsf{apply}(e_1, e_2) & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{apply}(\Box, e_2) > e_1 & \\
H; K \triangleright \mathsf{apply}(\Box, e_2) < v_1 & \mapsto_{\mathsf{a}} & H; K \triangleright \mathsf{apply}(v_1, \Box) > e_2 & \\
H; K \triangleright \mathsf{apply}(l_1, \Box) < v_2 & \mapsto_{\mathsf{a}} & H; K > \{v_2/x\}e_1 & (l_1{=}\lambda x.e_1 \in H) \\
H; K > \lambda x.e & \mapsto_{\mathsf{a}} & H, l{=}\lambda x.e; K < l & (l \notin \mathrm{dom}(H))
\end{array}
$$

Next we would like to show the correctness of the A-machine when compared to the C-machine. Interestingly, this becomes a *strong* bisimulation theorem: the two machines execute in lock-step. This requires that we set up a bisimulation relation. Following some of the prior examples we have seen, this amounts to substituting values for location labels l. Also as before, this has to be done recursively, because the values v can again contain references to other values, and so on. The inductive definition of the bisimulation as a judgment is not difficult, but somewhat tedious, so we only show a few cases. We have the judgments

$$
\begin{array}{l}
H; K \sim K' \\
H; e \sim e' \\
H; v \sim v' \\
H; s \sim s'
\end{array}
$$

defined by the following rules:

$$
\frac{H; v \sim v' \quad (l{=}v \in H)}{H; l \sim v'} \qquad \frac{H; K \sim K' \quad H; e \sim e'}{H; K > e \sim K' > e'} \qquad \frac{H; K \sim K' \quad H; v \sim v'}{H; K < v \sim K' < v'}
$$

$$
\frac{}{H; \bullet \sim \bullet} \qquad \frac{H; K \sim K' \quad H; e_2 \sim e_2'}{H; K \triangleright \mathsf{apply}(\Box, e_2) \sim K' \triangleright \mathsf{apply}(\Box, e_2')}
$$

$$
\frac{H; K \sim K' \quad H; v_1 \sim v_1'}{H; K \triangleright \mathsf{apply}(v_1, \Box) \sim K' \triangleright \mathsf{apply}(v_1', \Box)}
$$

$$
\frac{H; e_1 \sim e_1' \quad H; e_2 \sim e_2'}{H; \mathsf{apply}(e_1, e_2) \sim \mathsf{apply}(e_1', e_2')} \qquad \frac{H; e \sim e'}{H; \lambda x.e \sim \lambda x.e'} \qquad \frac{}{H; x \sim x}
$$

These rules are extended to handle pairs and states using straightforward congruence rules for all constructs. The fact this works essentially works like a congruence yields the following lemma. We take exchange for granted: the order of the locations in the heap is irrelevant. We use O and O' to stand for stacks, expressions, values, or states.

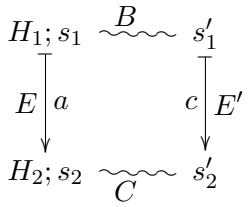
Lemma 1 (Weakening and Substitution)

(i) If $H; O \sim O'$ *then* $H, H'; O \sim O'$

(ii) If $H; v \sim v'$ *and* $H; O \sim O'$ *then* $H; \{v/x\}O \sim \{v'/x\}O'$

Proof: By induction on the structure of the given derivation relating O and O'. ■

Now we can prove strong bisimulation according to the following diagram:



We have to show (1) that if B and E are given, then s'_2, E', and C exist, and (2) that if B and E' are given, then H_2, s_2, E, and C exist.

Theorem 2 (Strong Bisimulation for A- and C-machine)

(i) If $H_1; s_1 \sim s'_1$ *and* $H_1; s_1 \mapsto_{\mathsf{a}} H_2; s_2$ *then there is an* s'_2 *such that* $s'_1 \mapsto_{\mathsf{c}} s'_2$ *and* $H_2; s_2 \sim s'_2$.

(ii) If $H_1; s_1 \sim s'_1$ *and* $s'_1 \mapsto_{\mathsf{c}} s'_2$ *then there is an* H_2 *and* s_2 *such that* $H_1; s_1 \mapsto_{\mathsf{a}} H_2; s_2$ *and* $H_2; s_2 \sim s'_2$.

Proof: In direction (1) we examine the cases for E, applying inversion to B to construct s'_2 and E'. In direction (2) we examine the cases for E', applying inversion to B to construction H_2, s_2 and C. ■

On observable values (i.e. integers, pairs of integers, etc.) the simulation yields the right translation, as can readily be verified.

Once we have heaps, we can formulate garbage collection as a way to trace through the heap and copying all locations accessible in a state. In some ways this inverts the weakening lemma into a *strengthening* property, where we remove part of the heap H' that is not referred to in the stack or expression. Garbage collection admits only weak bisimulation, since the steps of garbage collection are not accounted for in the C-machine. Please see [Ch. 31] for further details and discussion of garbage collection.