28 Control Stacks

Structural dynamics is convenient for proving properties of languages, such as a type safety theorem, but is less convenient as a guide for implementation. A structural dynamics defines a transition relation using rules that determine where to apply the next instruction without spelling out how to find where the instruction lies within an expression. To make this process explicit, we introduce a mechanism, called a *control stack*, that records the work that remains to be done after an instruction is executed. Using a stack eliminates the need for premises on the transition rules so that the transition system defines an *abstract machine* whose steps are determined by information explicit in its state, much as a concrete computer does.

In this chapter, we develop an abstract machine **K** for evaluating expressions in **PCF**. The machine makes explicit the context in which primitive instruction steps are executed, and the process by which the results are propagated to determine the next step of execution. We prove that **K** and **PCF** are equivalent in the sense that both achieve the same outcomes for the same expressions.

28.1 Machine Definition

A state s of the stack machine **K** for **PCF** consists of a *control stack* k and a closed expression e. States take one of two forms:

1. An *evaluation* state of the form $k \triangleright e$ corresponds to the evaluation of a closed expression e on a control stack k.
2. A *return* state of the form $k \triangleleft e$, where e val, corresponds to the evaluation of a stack k on a closed value e.

As an aid to memory, note that the separator "points to" the focal entity of the state, the expression in an evaluation state and the stack in a return state.

The control stack represents the context of evaluation. It records the "current location" of evaluation, the context into which the value of the current expression is returned. Formally, a control stack is a list of *frames*:

$$\frac{}{\epsilon\ \mathsf{stack}} \tag{28.1a}$$

$$\frac{f\ \mathsf{frame} \quad k\ \mathsf{stack}}{k;f\ \mathsf{stack}} \tag{28.1b}$$

The frames of the **K** machine are inductively defined by the following rules:

$$\frac{}{\mathtt{s}(-)\ \mathsf{frame}} \tag{28.2a}$$

$$\frac{}{\mathtt{ifz}\{e_0;x.e_1\}(-)\ \mathsf{frame}} \tag{28.2b}$$

$$\frac{}{\mathtt{ap}(-;e_2)\ \mathsf{frame}} \tag{28.2c}$$

The frames correspond to search rules in the dynamics of **PCF**. Thus, instead of relying on the structure of the transition derivation to keep a record of pending computations, we make an explicit record of them in the form of a frame on the control stack.

The transition judgment between states of the **PCF** machine is inductively defined by a set of inference rules. We begin with the rules for natural numbers, using an eager semantics for the successor.

$$\frac{}{k \triangleright \mathtt{z} \longmapsto k \triangleleft \mathtt{z}} \tag{28.3a}$$

$$\frac{}{k \triangleright \mathtt{s}(e) \longmapsto k;\mathtt{s}(-) \triangleright e} \tag{28.3b}$$

$$\frac{}{k;\mathtt{s}(-) \triangleleft e \longmapsto k \triangleleft \mathtt{s}(e)} \tag{28.3c}$$

To evaluate $\mathtt{z}$, we simply return it. To evaluate $\mathtt{s}(e)$, we push a frame on the stack to record the pending successor and evaluate e; when that returns with e', we return $\mathtt{s}(e')$ to the stack.

Next, we consider the rules for case analysis.

$$\frac{}{k \triangleright \mathtt{ifz}\{e_0;x.e_1\}(e) \longmapsto k;\mathtt{ifz}\{e_0;x.e_1\}(-) \triangleright e} \tag{28.4a}$$

$$\frac{}{k;\mathtt{ifz}\{e_0;x.e_1\}(-) \triangleleft \mathtt{z} \longmapsto k \triangleright e_0} \tag{28.4b}$$

$$\frac{}{k;\mathtt{ifz}\{e_0;x.e_1\}(-) \triangleleft \mathtt{s}(e) \longmapsto k \triangleright [e/x]e_1} \tag{28.4c}$$

The test expression is evaluated, recording the pending case analysis on the stack. Once the value of the test expression is determined, the zero or non-zero branch of the condition is evaluated, substituting the predecessor in the latter case.

Finally, we give the rules for functions, which are evaluated by-name, and the rule for general recursion.

$$\frac{}{k \triangleright \mathtt{lam}\{\tau\}(x.e) \longmapsto k \triangleleft \mathtt{lam}\{\tau\}(x.e)} \tag{28.5a}$$

$$\frac{}{k \triangleright \mathtt{ap}(e_1;e_2) \longmapsto k;\mathtt{ap}(-;e_2) \triangleright e_1} \tag{28.5b}$$

$$\frac{}{k;\mathsf{ap}(-;e_2) \lhd \mathtt{lam}\{\tau\}(x.e) \longmapsto k \rhd [e_2/x]e} \tag{28.5c}$$

$$\frac{}{k \rhd \mathtt{fix}\{\tau\}(x.e) \longmapsto k \rhd [\mathtt{fix}\{\tau\}(x.e)/x]e} \tag{28.5d}$$

It is important that evaluation of a general recursion requires no stack space.

The initial and final states of the **K** machine are defined by the following rules:

$$\frac{}{\epsilon \rhd e \ \mathsf{initial}} \tag{28.6a}$$

$$\frac{e \ \mathsf{val}}{\epsilon \lhd e \ \mathsf{final}} \tag{28.6b}$$

28.2 Safety

To define and prove safety for the **PCF** machine requires that we introduce a new typing judgment, $k \lhd: \tau$, which states that the stack k expects a value of type τ. This judgment is inductively defined by the following rules:

$$\frac{}{\epsilon \lhd: \tau} \tag{28.7a}$$

$$\frac{k \lhd: \tau' \quad f : \tau \rightsquigarrow \tau'}{k;f \lhd: \tau} \tag{28.7b}$$

This definition makes use of an auxiliary judgment, $f : \tau \rightsquigarrow \tau'$, stating that a frame f transforms a value of type τ to a value of type τ'.

$$\frac{}{\mathsf{s}(-) : \mathtt{nat} \rightsquigarrow \mathtt{nat}} \tag{28.8a}$$

$$\frac{e_0 : \tau \quad x : \mathtt{nat} \vdash e_1 : \tau}{\mathtt{ifz}\{e_0; x.e_1\}(-) : \mathtt{nat} \rightsquigarrow \tau} \tag{28.8b}$$

$$\frac{e_2 : \tau_2}{\mathsf{ap}(-;e_2) : \mathtt{parr}(\tau_2;\tau) \rightsquigarrow \tau} \tag{28.8c}$$

The states of the **PCF** machine are well-formed if their stack and expression components match:

$$\frac{k \lhd: \tau \quad e : \tau}{k \rhd e \ \mathsf{ok}} \tag{28.9a}$$

$$\frac{k \lhd: \tau \quad e : \tau \quad e \ \mathsf{val}}{k \lhd e \ \mathsf{ok}} \tag{28.9b}$$

We leave the proof of safety of the **PCF** machine as an exercise.

Theorem 28.1 (Safety)**.** *1. If s ok and $s \longmapsto s'$, then s' ok.*

2. If s ok, then either s final or there exists s' such that $s \longmapsto s'$.

28.3 Correctness of the K Machine

Does evaluation of an expression e using the **K** machine yield the same result as does the structural dynamics of **PCF**? The answer to this question can be derived from the following facts.

Completeness If $e \longmapsto^* e'$, where e' val, then $\epsilon \triangleright e \longmapsto^* \epsilon \triangleleft e'$.
Soundness If $\epsilon \triangleright e \longmapsto^* \epsilon \triangleleft e'$, then $e \longmapsto^* e'$ with e' val.

To prove completeness a plausible first step is to consider a proof by induction on the definition of multi-step transition, which reduces the theorem to the following two lemmas:

1. If e val, then $\epsilon \triangleright e \longmapsto^* \epsilon \triangleleft e$.
2. If $e \longmapsto e'$, then, for every v val, if $\epsilon \triangleright e' \longmapsto^* \epsilon \triangleleft v$, then $\epsilon \triangleright e \longmapsto^* \epsilon \triangleleft v$.

The first can be proved easily by induction on the structure of e. The second requires an inductive analysis of the derivation of $e \longmapsto e'$ that gives rise to two complications. The first complication is that we cannot restrict attention to the empty stack, for if e is, say, $\mathsf{ap}(e_1; e_2)$, then the first step of the **K** machine is

$$\epsilon \triangleright \mathsf{ap}(e_1; e_2) \longmapsto \epsilon;\mathsf{ap}(-; e_2) \triangleright e_1.$$

To handle such situations, we consider the evaluation of e_1 on any stack, not just the empty stack.

Specifically, we prove that if $e \longmapsto e'$ and $k \triangleright e' \longmapsto^* k \triangleleft v$, then $k \triangleright e \longmapsto^* k \triangleleft v$. Reconsider the case $e = \mathsf{ap}(e_1; e_2)$, $e' = \mathsf{ap}(e_1'; e_2)$, with $e_1 \longmapsto e_1'$. We are given that $k \triangleright \mathsf{ap}(e_1'; e_2) \longmapsto^* k \triangleleft v$, and we are to show that $k \triangleright \mathsf{ap}(e_1; e_2) \longmapsto^* k \triangleleft v$. It is easy to show that the first step of the former derivation is

$$k \triangleright \mathsf{ap}(e_1'; e_2) \longmapsto k;\mathsf{ap}(-; e_2) \triangleright e_1'.$$

We would like to apply induction to the derivation of $e_1 \longmapsto e_1'$, but to do so, we need a value v_1 such that $e_1' \longmapsto^* v_1$, which is not at hand.

We therefore consider the value of each sub-expression of an expression. This information is given by the evaluation dynamics described in Chapter 7, which has the property that $e \Downarrow e'$ iff $e \longmapsto^* e'$ and e' val.

Lemma 28.2. *If $e \Downarrow v$, then for every k stack, $k \triangleright e \longmapsto^* k \triangleleft v$.*

The desired result follows by the analog of Theorem 7.2 for **PCF**, which states that $e \Downarrow v$ iff $e \longmapsto^* v$.

To prove soundness, we note that it is awkward to reason inductively about a multi-step transition from $\epsilon \triangleright e \longmapsto^* \epsilon \triangleleft v$. The intermediate steps could involve alternations of evaluation and return states. Instead, we consider a **K** machine state to encode an expression, and show that the machine transitions are simulated by the transitions of the structural dynamics.

To do so, we define a judgment, $s \looparrowright e$, stating that state s "unravels to" expression e. It will turn out that for initial states, $s = \epsilon \rhd e$, and final states, $s = \epsilon \lhd e$, we have $s \looparrowright e$. Then we show that if $s \longmapsto^* s'$, where s' final, $s \looparrowright e$, and $s' \looparrowright e'$, then e' val and $e \longmapsto^* e'$. For this, it is enough to show the following two facts:

1. If $s \looparrowright e$ and s final, then e val.
2. If $s \longmapsto s'$, $s \looparrowright e$, $s' \looparrowright e'$, and $e' \longmapsto^* v$, where v val, then $e \longmapsto^* v$.

The first is quite simple, we need only note that the unraveling of a final state is a value. For the second, it is enough to prove the following lemma.

Lemma 28.3. *If $s \longmapsto s'$, $s \looparrowright e$, and $s' \looparrowright e'$, then $e \longmapsto^* e'$.*

Corollary 28.4. *$e \longmapsto^* \overline{n}$ iff $\epsilon \rhd e \longmapsto^* \epsilon \lhd \overline{n}$.*

28.3.1 Completeness

Proof of Lemma 28.2 The proof is by induction on an evaluation dynamics for **PCF**.

Consider the evaluation rule

$$\frac{e_1 \Downarrow \mathtt{lam}\{\tau_2\}(x.e) \quad [e_2/x]e \Downarrow v}{\mathsf{ap}(e_1; e_2) \Downarrow v} \tag{28.10}$$

For an arbitrary control stack k, we are to show that $k \rhd \mathsf{ap}(e_1; e_2) \longmapsto^* k \lhd v$. Applying both of the inductive hypotheses in succession, interleaved with steps of the **K** machine, we obtain

$$\begin{aligned} k \rhd \mathsf{ap}(e_1; e_2) &\longmapsto k;\mathsf{ap}(-; e_2) \rhd e_1 \\ &\longmapsto^* k;\mathsf{ap}(-; e_2) \lhd \mathtt{lam}\{\tau_2\}(x.e) \\ &\longmapsto k \rhd [e_2/x]e \\ &\longmapsto^* k \lhd v. \end{aligned}$$

The other cases of the proof are handled similarly. □

28.3.2 Soundness

The judgment $s \looparrowright e'$, where s is either $k \rhd e$ or $k \lhd e$, is defined in terms of the auxiliary judgment $k \bowtie e = e'$ by the following rules:

$$\frac{k \bowtie e = e'}{k \rhd e \looparrowright e'} \tag{28.11a}$$

$$\frac{k \bowtie e = e'}{k \lhd e \looparrowright e'} \tag{28.11b}$$

In words, to unravel a state, we wrap the stack around the expression to form a complete program. The unraveling relation is inductively defined by the following rules:

$$\frac{}{\epsilon \bowtie e = e} \tag{28.12a}$$

$$\frac{k \bowtie \mathtt{s}(e) = e'}{k;\mathtt{s}(-) \bowtie e = e'} \tag{28.12b}$$

$$\frac{k \bowtie \mathtt{ifz}\{e_0; x.e_1\}(e) = e'}{k;\mathtt{ifz}\{e_0; x.e_1\}(-) \bowtie e = e'} \tag{28.12c}$$

$$\frac{k \bowtie \mathtt{ap}(e_1; e_2) = e}{k;\mathtt{ap}(-; e_2) \bowtie e_1 = e} \tag{28.12d}$$

These judgments both define total functions.

Lemma 28.5. *The judgment $s \looparrowright e$ relates every state s to a unique expression e, and the judgment $k \bowtie e = e'$ relates every stack k and expression e to a unique expression e'.*

We are therefore justified in writing $k \bowtie e$ for the unique e' such that $k \bowtie e = e'$.

The following lemma is crucial. It states that unraveling preserves the transition relation.

Lemma 28.6. *If $e \longmapsto e'$, $k \bowtie e = d$, $k \bowtie e' = d'$, then $d \longmapsto d'$.*

Proof The proof is by rule induction on the transition $e \longmapsto e'$. The inductive cases, where the transition rule has a premise, follow easily by induction. The base cases, where the transition is an axiom, are proved by an inductive analysis of the stack k.

For an example of an inductive case, suppose that $e = \mathtt{ap}(e_1; e_2)$, $e' = \mathtt{ap}(e_1'; e_2)$, and $e_1 \longmapsto e_1'$. We have $k \bowtie e = d$ and $k \bowtie e' = d'$. It follows from rules (28.12) that $k;\mathtt{ap}(-; e_2) \bowtie e_1 = d$ and $k;\mathtt{ap}(-; e_2) \bowtie e_1' = d'$. So by induction $d \longmapsto d'$, as desired.

For an example of a base case, suppose that $e = \mathtt{ap}(\mathtt{lam}\{\tau_2\}(x.e); e_2)$ and $e' = [e_2/x]e$ with $e \longmapsto e'$ directly. Assume that $k \bowtie e = d$ and $k \bowtie e' = d'$; we are to show that $d \longmapsto d'$. We proceed by an inner induction on the structure of k. If $k = \epsilon$, the result follows immediately. Consider, say, the stack $k = k';\mathtt{ap}(-; c_2)$. It follows from rules (28.12) that $k' \bowtie \mathtt{ap}(e; c_2) = d$ and $k' \bowtie \mathtt{ap}(e'; c_2) = d'$. But by the structural dynamics $\mathtt{ap}(e; c_2) \longmapsto \mathtt{ap}(e'; c_2)$, so by the inner inductive hypothesis we have $d \longmapsto d'$, as desired. □

We may now complete the proof of Lemma 28.3.

Proof of Lemma 28.3 The proof is by case analysis on the transitions of the **K** machine. In each case, after unraveling, the transition will correspond to zero or one transitions of the **PCF** structural dynamics.

Suppose that $s = k \triangleright \mathtt{s}(e)$ and $s' = k;\mathtt{s}(-) \triangleright e$. Note that $k \bowtie \mathtt{s}(e) = e'$ iff $k;\mathtt{s}(-) \bowtie e = e'$, from which the result follows immediately.

Suppose that $s = k;\mathtt{ap}(\mathtt{lam}\{\tau\}(x.e_1); -) \lhd e_2$ and $s' = k \rhd [e_2/x]e_1$. Let e' be such that $k;\mathtt{ap}(\mathtt{lam}\{\tau\}(x.e_1); -) \bowtie e_2 = e'$ and let e'' be such that $k \bowtie [e_2/x]e_1 = e''$. Observe that $k \bowtie \mathtt{ap}(\mathtt{lam}\{\tau\}(x.e_1); e_2) = e'$. The result follows from Lemma 28.6.

□

28.4 Notes

The abstract machine considered here is typical of a wide class of machines that make control flow explicit in the state. The prototype is the SECD machine (Landin, 1965), which is a linearization of a structural operational semantics (Plotkin, 1981). The advantage of a machine model is that the explicit treatment of control is needed for languages that allow the control state to be manipulated (see Chapter 30 for a prime example). The disadvantage is that the control state of the computation must be made explicit, necessitating rules for manipulating it that are left implicit in a structural dynamics.

Exercises

28.1. Give the proof of Theorem 28.1 for conditional expressions.

28.2. Formulate a call-by-value variant of the **PCF** machine.

28.3. Analyze the worst-case asymptotic complexity of executing each instruction of the **K** machine.

28.4. Refine the proof of Lemma 28.2 by bounding the number of machine steps taken for each step of the **PCF** dynamics.