PFPL Supplement: Stack Machines By Name and By Value*

Robert Harper

Fall, 2018

1 Introduction

This note expands on Chapters 28 and 29 of **PFPL** by considering both a by-name and a by-value stack machine, and proving safety for both cases.

2 By-Name Stack Machine

The abstract syntax of the by-name stack machine for **PCF** is given by the following grammar:

Types	τ	$::=$	$\mathtt{nat} \mid \tau_1 \rightharpoonup \tau_2$
Stacks	k	$::=$	$\varepsilon \mid k\,;\,f$
Frames	f	$::=$	$\mathtt{ap}(-\,;\,e_2) \mid \mathtt{ifz}\{\tau\}(-\,;\,e_0\,;\,x\,.\,e_1)$
Expr's	e	$::=$	$x \mid \mathtt{z} \mid \mathtt{s}(e) \mid \mathtt{ifz}\{\tau\}(e\,;\,e_0\,;\,x\,.\,e_1) \mid \lambda\{\tau\}(x\,.\,e) \mid \mathtt{ap}(e_1\,;\,e_2) \mid \mathtt{fix}\{\tau\}(x\,.\,e)$
States	s	$::=$	$k \# e$

The statics for the by-name stack machine is given in Figure 1; the typing rules for **PCF**-by-name expressions are omitted, as they can be found in the text. The dynamics for the by-name stack machine is defined in Figure 2.

The safety theorem for the by-name stack machine is stated according to the familiar pattern.

Theorem 2.1 (Safety)**.** *The by-name stack machine enjoys these properties:*

1. *If $s \longmapsto s'$ then s ok implies s' ok.*
2. *If s ok, then either s final or $s \longmapsto s'$ for some state s'.*

To prove these requires two auxiliary lemmas, the canonical forms lemma and the stack lemma.

Lemma 2.2 (Canonical Forms By Name)**.** *If $e : \tau$ and e val, then*

1. *if $\tau = \mathtt{nat}$, then either $e = \mathtt{z}$ or $e = \mathtt{s}(e')$ for some $e' : \mathtt{nat}$;*
2. *if $\tau = \tau_1 \rightharpoonup \tau_2$, then $e = \lambda\{\tau_1\}(x\,.\,e_2)$ with $x : \tau_1 \vdash e_2 : \tau_2$.*

Lemma 2.3 (Stacks By Name)**.** *If $k \div \tau$ is non-empty, then*

$$\frac{}{\varepsilon \div \tau} \tag{1a}$$

$$\frac{k \div \tau \quad f : \tau' \rightsquigarrow \tau}{k \,;\, f \div \tau'} \tag{1b}$$

$$\frac{k \div \tau \quad e_2 : \tau_2}{\mathtt{ap}(-\,;\, e_2\,) : \tau_2 \rightharpoonup \tau \rightsquigarrow \tau} \tag{1c}$$

$$\frac{e_0 : \tau \quad x : \mathtt{nat} \vdash e_1 : \tau}{\mathtt{ifz}\{\tau\}(-\,;\, e_0\,;\, x\,.\,e_1\,) : \mathtt{nat} \rightsquigarrow \tau} \tag{1d}$$

$$\frac{k \div \tau \quad e : \tau}{k \mathbin{\#} e \;\mathsf{ok}} \tag{1e}$$

Figure 1: Statics of By-Name Stack Machine

$$\frac{}{\varepsilon \mathbin{\#} e \;\mathsf{initial}} \tag{2}$$

$$\frac{e \;\mathsf{val}}{\varepsilon \mathbin{\#} e \;\mathsf{final}} \tag{3}$$

$$\frac{}{k \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0\,;\, x\,.\,e_1\,) \mathbin{\#} \mathtt{z} \longmapsto k \mathbin{\#} e_0} \tag{4a}$$

$$\frac{}{k \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0\,;\, x\,.\,e_1\,) \mathbin{\#} \mathtt{s}(\,e\,) \longmapsto k \mathbin{\#} [e/x]e_1} \tag{4b}$$

$$\frac{}{k \mathbin{\#} \mathtt{ifz}\{\tau\}(\,e\,;\, e_0\,;\, x\,.\,e_1\,) \longmapsto k \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0\,;\, x\,.\,e_1\,) \mathbin{\#} e} \tag{4c}$$

$$\frac{}{k \,;\, \mathtt{ap}(-\,;\, e_2\,) \mathbin{\#} \lambda\{\tau_2\}(\,x\,.\,e\,) \longmapsto k \mathbin{\#} [e_2/x]e} \tag{4d}$$

$$\frac{}{k \mathbin{\#} \mathtt{ap}(\,e_1\,;\, e_2\,) \longmapsto k \,;\, \mathtt{ap}(-\,;\, e_2\,) \mathbin{\#} e_1} \tag{4e}$$

$$\frac{}{k \mathbin{\#} \mathtt{fix}\{\tau\}(\,x\,.\,e\,) \longmapsto k \mathbin{\#} [\mathtt{fix}\{\tau\}(\,x\,.\,e\,)/x]e} \tag{4f}$$

Figure 2: Dynamics of By-Name Stack Machine

1. *if* $\tau = \mathtt{nat}$, *then* $k = k' \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0 \,;\, x \,.\, e_1)$ *for some* $k' \div \tau$, $e_0 : \tau$, *and* $x : \mathtt{nat} \vdash e_1 : \tau$.

2. *if* $\tau = \tau_2 \rightharpoonup \tau'$, *then* $k = k' \,;\, \mathtt{ap}(-\,;\, e_2)$ *for some* $k' \div \tau_2 \rightharpoonup \tau'$ *and* $e_2 : \tau_2$.

Both of these lemmas are proved by induction on the relevant typing judgments. Now, on to the proof the safety theorem.

Proof. 1. Preservation is proved by induction on the transitions of the state machine. Because the transitions are defined by rules free of premises, the induction reduces to a case analysis on each of the rules, with no appeal to any inductive hypothesis. Assume that s ok, and that $s \longmapsto s'$ for some s', with the intention to show that s' ok. By inversion on the first assumption, $s = k \# e$ for some k and e such that $k \div \sigma$ and $e : \sigma$ for some mediating type σ. Consider the possible transitions from s:

   (a) Rule (4a): The mediating type σ must be $\mathtt{nat}$, and so $k' \div \tau$ where $e_0 : \tau$ and $x : \mathtt{nat} \vdash e_1 : \tau$. But then $k' \# e_0$ ok, taking the mediating type to be τ.

   (b) Rule (4b) Similar, using inversion to obtain the type of the predecessor.

   (c) Rule (4c): The mediating type σ is the type τ of the conditional, and so $k \div \tau$, $e_0 : \tau$, and $x : \mathtt{nat} \vdash e_1 : \tau$. But then $k \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0 \,;\, x \,.\, e_1) \div \mathtt{nat}$, which is enough to ensure $k \,;\, \mathtt{ifz}\{\tau\}(-\,;\, e_0 \,;\, x \,.\, e_1) \# e$ ok.

   (d) Rules (4d) and Rule (4e): Similar.

   (e) Rule (4f): immediate by inversion of typing.

2. Progress is proved by analysis of the assumption s ok. By inversion s must be $k \# e$ fo some k and e such that $k \div \sigma$ and $e : \sigma$ for some mediating type σ. Proceed by cases on whether e is a value or not.

   (a) If e val, then consider whether k is empty or not. If $k = \varepsilon$, then s final, which completes the proof. If k is non-empty, then the stack lemma applies, according to the mediating type σ.

      i. If $\sigma = \mathtt{nat}$, then by the stack lemma $k = k' \,;\, \mathtt{ifz}\{\tau'\}(-\,;\, e_0 \,;\, x \,.\, e_1)$, where $k' \div \tau'$, $e_0 : \tau'$, and $x : \mathtt{nat} \vdash e_1 : \tau'$. By the canonical forms lemma either $e = \mathtt{z}$ or $e = \mathtt{s}(e')$ with $e' : \mathtt{nat}$. In the former case, apply Rule (4a), in the latter apply Rule (4b).

      ii. If $\sigma = \sigma_1 \rightharpoonup \sigma_2$, then by canonical forms $e = \lambda\{\sigma_1\}(x \,.\, e_2)$, and by the stack lemma $k = k' \,;\, \mathtt{ap}(-\,;\, e_2)$, where $k' \div \sigma_2$ and $x : \sigma_1 \vdash e_2 : \tau_2$. Then apply Rule (4e) to ensure progress.

   (b) If e is not a value, then either $e = \mathtt{ap}(e_1 \,;\, e_2)$ or $e = \mathtt{ifz}\{\tau\}(e' \,;\, e_0 \,;\, x \,.\, e_1)$ or $e = \mathtt{fix}\{\tau\}(x \,.\, e)$. Rules (4c), (4e), and $(4f)$ ensure progress in the respective cases.

□

The proof is not all that different from the usual one for **PCF**, but for the explicit manipulation of the stack.

3 By-Value Stack Machine

The by-value stack machine is defined in **PFPL** using states of the form $k \triangleright e$ and $k \triangleleft e$ where e val. The by-value dynamics is given in Figure 3 for convenient reference. Observe that the successor operation is strict (evaluates its argument) and that applications evaluate their argument before calling the specified (recursive) function.

The proof of the safety theorem for the by-value stack machine proceeds along similar lines to the proof for the by-name case, with the significant difference being the statements of the canonical forms and stack lemmas. With regard to stacks, the type of the stack no longer uniquely determines the topmost frame, because of the eager successor and call-by-value application.

Lemma 3.1 (Canonical Forms)**.** *If $e : \tau$ and e val, then*

1. *if $\tau = \mathtt{nat}$, then $e = \overline{n}$ for some $n \in \mathbb{N}$.*
2. *if $\tau = \tau_1 \rightharpoonup \tau_2$, then $e = \mathtt{fun}\{\tau_1\,;\tau_2\}(\,f\,.\,x\,.\,e_2\,)$ with $f : \tau_1 \rightharpoonup \tau_2, x : \tau_1 \vdash e_2 : \tau_2$.*

Lemma 3.2 (Stacks)**.** *If $k \div \tau$ for some non-empty stack $k = k'\,;f$, then*

1. *if $\tau = \mathtt{nat}$, then $k' \div \tau'$, and either*
   (a) *$f = \mathtt{ifz}\{\tau'\}(-\,;e_0\,;x\,.\,e_1\,)$ with $e_0 : \tau'$ and $x : \mathtt{nat} \vdash e_1 : \tau'$, or*
   (b) *$f = \mathtt{s}(-\,)$ and $\tau' = \mathtt{nat}$, or*
   (c) *$f = \mathtt{ap}(\,e_1\,;-\,)$ with e_1 val and $e_1 : \mathtt{nat} \rightharpoonup \tau'$.*
2. *if $\tau = \tau_2 \rightharpoonup \tau'$, then $k' \div \tau'$ and either*
   (a) *$f = \mathtt{ap}(-\,;e_2\,)$ with $e_2 : \tau_2$, or*
   (b) *$f = \mathtt{ap}(\,e_1\,;-\,)$ with e_1 val and $e_1 : \tau \rightharpoonup \tau'$, or*
   (c) *$f = \mathtt{ifz}\{\tau'\}(-\,;e_0\,;x\,.\,e_1\,)$ with $e_0 : \tau'$ and $x : \mathtt{nat} \vdash e_1 : \tau'$.*

The by-value dynamics introduces many more possibilities for a stack of a given type! This complicates the safety proof by requiring further case analyses according to the stacks lemma, applying the appropriate rule in each case to established progress.

4 Stack Machine for PCFv

The language **PCFv** defined in Harper (2019b) refines the statics to distinguish between values and computations, and defines a type of unevaluated computations whose elimination form sequences the evaluation of one computation prior to another. In the presence of sequencing the principal arguments of elimination forms may be required to be open values, off-loading responsibility for evaluating these arguments to the programmer (or compiler). **PCFv** is further extended to account for excceptions in Harper (2019a) by generalizing the sequentialization to account for both "normal" and "exceptional" completion of a computation. Correspondingly, the dynamics for **PCFv** is simplified by removing the obligation to evaluate the arguments to elimination forms prior to performing the elimination itself. The stack machine for **PCFv** is similarly simplified to consider only one form of stack frame corresponding to the deferred evaluation of the second component of a sequential binding.

$$\overline{k \triangleright \mathtt{z} \longmapsto k \triangleleft \mathtt{z}} \tag{5a}$$

$$\overline{k \triangleright \mathtt{s}(\, e\,) \longmapsto k\,;\mathtt{s}(\,-\,) \triangleright e} \tag{5b}$$

$$\overline{k\,;\mathtt{s}(\,-\,) \triangleleft \overline{n} \longmapsto k \triangleleft \overline{\overline{n+1}}} \tag{5c}$$

$$\overline{k \triangleright \mathtt{ifz}\{\tau\}(\, e\,;e_0\,;x\,.\,e_1\,) \longmapsto k\,;\mathtt{ifz}\{\tau\}(\,-\,;e_0\,;x\,.\,e_1\,) \triangleright e} \tag{5d}$$

$$\overline{k\,;\mathtt{ifz}\{\tau\}(\,-\,;e_0\,;x\,.\,e_1\,) \triangleleft \overline{0} \longmapsto k \triangleright e_0} \tag{5e}$$

$$\overline{k\,;\mathtt{ifz}\{\tau\}(\,-\,;e_0\,;x\,.\,e_1\,) \triangleleft \overline{n+1} \longmapsto k \triangleright [\overline{n}/x]e_1} \tag{5f}$$

$$\overline{k \triangleright \mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,) \longmapsto k \triangleleft \mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,)} \tag{5g}$$

$$\overline{k \triangleright \mathtt{ap}(\, e_1\,;e_2\,) \longmapsto k\,;\mathtt{ap}(\,-\,;e_2\,) \triangleright e_1} \tag{5h}$$

$$\overline{k\,;\mathtt{ap}(\,-\,;e_2\,) \triangleleft \mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,) \longmapsto k\,;\mathtt{ap}(\,\mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,)\,;-\,) \triangleright e_2} \tag{5i}$$

$$\overline{k\,;\mathtt{ap}(\,\mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,)\,;-\,) \triangleleft e_2 \longmapsto k \triangleright [\mathtt{fun}\{\tau_2\,;\tau\}(\, f\,.\,x\,.\,e\,), e_2/f, x]e} \tag{5j}$$

Figure 3: Dynamics of the By-Value Stack Machine

Thus, the structural dynamics for sequential binding given in Harper (2019b),

$$\frac{e \longmapsto e'}{\mathtt{bnd}(\,\mathtt{comp}(\,e\,)\,;x\,.\,e_2\,) \longmapsto \mathtt{bnd}(\,\mathtt{comp}(\,e'\,)\,;x\,.\,e_2\,)} \tag{6a}$$

$$\frac{e\ \mathsf{val}}{\mathtt{bnd}(\,\mathtt{comp}(\,e\,)\,;x\,.\,e_2\,) \longmapsto [e/x]e_2} \tag{6b}$$

are expressed by the following rules for the **PCFv** stack machine:

$$\overline{k \triangleright \mathtt{bnd}(\,\mathtt{comp}(\,e\,)\,;x\,.\,e_2\,) \longmapsto k\,;x\,.\,e_2 \triangleright e} \tag{7a}$$

$$\frac{e\ \mathsf{val}}{k\,;x\,.\,e_2 \triangleleft e \longmapsto k \triangleright [e/x]e_2} \tag{7b}$$

Now the *only* form of stack frame is the abstractor, $x.e_2$, which represents $\mathtt{bnd}(\,\mathtt{comp}(\,-\,)\,;x.e_2\,)$, with the "hole" being the computation site. No other form of frame is required, because the arguments to elimination forms are restricted to be values.

To relate the two formulations of the by-value stack machine, it is helpful to trace out an evaluation of the extended form of application,

$$\mathtt{ap}^*(\,e_1\,;e_2\,) \triangleq \mathtt{bnd}(\,\mathtt{comp}(\,e_1\,)\,;x_1\,.\,\mathtt{bnd}(\,\mathtt{comp}(\,e_2\,)\,;x_2\,.\,\mathtt{ap}(\,x_1\,;x_2\,)\,)\,)$$

on some stack k:

$$\begin{aligned}
k \triangleright \mathtt{ap}^*(\,e_1\,;e_2\,) &\longmapsto k\,;x_1\,.\,\mathtt{bnd}(\,\mathtt{comp}(\,e_2\,)\,;x_2\,.\,\mathtt{ap}(\,x_1\,;x_2\,)\,) \triangleright e_1\\
&\longmapsto^* k\,;x_1\,.\,\mathtt{bnd}(\,\mathtt{comp}(\,e_2\,)\,;x_2\,.\,\mathtt{ap}(\,x_1\,;x_2\,)\,) \triangleleft v_1\\
&\longmapsto k\,;x_2\,.\,\mathtt{ap}(\,v_1\,;x_2\,) \triangleright e_2\\
&\longmapsto^* k\,;x_2\,.\,\mathtt{ap}(\,v_1\,;x_2\,) \triangleleft v_2\\
&\longmapsto k \triangleright \mathtt{ap}(\,v_1\,;v_2\,)\\
&\longmapsto k \triangleright [v_1, v_2/f, x]e
\end{aligned}$$

where $v_1 \triangleq \mathtt{fun}\{\tau_2\,;\tau\}(\,f\,.\,x\,.\,e\,)$ is the value of e_1 and v_2 is the value of e_2. The abstractor

$$x_1\,.\,\mathtt{bnd}(\,\mathtt{comp}(\,e_2\,)\,;x_2\,.\,\mathtt{ap}(\,x_1\,;x_2\,)\,)$$

pushed onto the stack during evaluation of e_1 corresponds to the frame $\mathtt{ap}(\,-;e_2\,)$, and the abstractor $x_2\,.\,\mathtt{ap}(\,v_1\,;x_2\,)$ pushed onto the stack during evaluation of e_2 corresponds to the frame $\mathtt{ap}(\,v_1\,;-\,)$.

References

Robert Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, Cambridge, England, Second edition, 2016.

Robert Harper. Exceptions: Control and data. Supplement to Harper (2016), Fall 2019a. URL https://www.cs.cmu.edu/~rwh/pfpl/supplements/exceptions.pdf.

Robert Harper. PCF-By-Value. Supplement to Harper (2016), Fall 2019b. URL https://www.cs.cmu.edu/~rwh/pfpl/supplements/pcfv.pdf.