PFPL Supplement: By-Name as By-Value*

Robert Harper

Spring, 2023

1 Introduction

The by-name interpretation of PCF regards variables as ranging over *unevaluated computations*, rather than fully evaluated values. Moreover, the successor operation is interpreted lazily, which is consistent with the dynamics of the conditional, which binds a variable to the predecessor in the successor case. The by-name interpretation supports *fixed points* at any type; in particular, it permits the definition of the divergent computation, Ω_τ, for any type τ, as $\mathtt{fix}[\tau](x\,.\,x)$, which steps to itself. But then how can such an expression, which has no value, be substituted for a variable? Moreover, using a fixed point, it is possible to form a "self-referential natural number" such as the infinite stack of successors. But what is such a thing? It is surely not a natural number.

The situation can be clarified by interpreting the by-name version of PCF into the by-value version by making explicit when computations are suspended. To model this we make use of the self-referential suspension types, $\mathtt{self}(\tau)$, whose *values* are (possibly self-referential) unevaluated computations of type τ, $\mathtt{self}[\tau](x\,.\,e)$. Such computations are activated by the elimination form, $\mathtt{unroll}[\tau](e)$. When self-reference is not needed, the type $\mathtt{self}(\tau)$ may be written $\mathtt{susp}(\tau)$, with introductions $\mathtt{susp}[\tau](e)$ being $\mathtt{self}[\tau](\_\,.\,e)$, and elimination $\mathtt{force}[\tau](e)$ being $\mathtt{unroll}[\tau](e)$.

2 Translation of Types from By-Name to By-Value

The main idea is to define a translation of types under the by-name intepretation into types under the by-value interpretation that makes explicit use of self-referential suspension types, and to define a corresponding translation of expressions that is consistent with this type interpretation.

The following equations define the types $|\tau|$ and $||\tau||$ in such a way as to reflect their by-name meaning in a by-value setting:

$$|\tau| \triangleq \mathtt{self}(||\tau||)$$

$$||\mathtt{nat}|| \triangleq \mathtt{lnat}$$

$$||\tau_1 \rightharpoonup \tau_2|| \triangleq |\tau_1| \rightharpoonup |\tau_2|$$

where $\mathtt{lnat} \triangleq \mathtt{rec}(t.1 + \mathtt{self}(t))$ is the recursive type of lazy natural numbers in the by-value setting. Observe that the type $|\mathtt{nat}|$ unfolds to the type $1 + |\mathtt{nat}|$, reflecting that a computation of this type,

when forced, indicates whether it is zero or non-zero, and in the latter case provides a computation of its predecessor. Moreover, function types are interpreted as computations of functions taking computations to computations, a concise summary of the "call-by-name" dynamics.

Expressions are translated from by-name to by-value according to the following specification:

$$\text{if } \Gamma \vdash_{\mathsf{bn}} e : \tau, \text{ then } |\Gamma| \vdash_{\mathsf{bv}} |e| : |\tau|,$$

where $|\Gamma|(x) = |\Gamma(x)|$ for each x in Γ. The translations are given by induction on the statics, translating well-typed expressions in the by-name setting to well-typed expressions of translated type in the by-value setting.

First, variables are translated as themselves, $|x| \triangleq x$, ranging as they do over unevaluated computations of their type.

$$\text{VAR}\ \frac{}{\Gamma, x : |\tau| \vdash_{\mathsf{bv}} x : |\tau|}$$

Second, fixed points $\mathtt{fix}(x\,.\,e)$ are translated as self-referential values:

$$\text{FIX}\ \frac{|\Gamma|, x : |\tau| \vdash_{\mathsf{bv}} |e| : |\tau|}{\Gamma \vdash_{\mathsf{bv}} \mathtt{self}(x\,.\,\mathtt{unroll}(|e|)) : |\tau|}$$

Thus, in particular, the divergent expression $\mathtt{fix}(x\,.\,x)$ is translated to the value $\mathtt{self}(x\,.\,\mathtt{unroll}(x))$ that, when unrolled, continues to unroll itself.

Considering the type of natural numbers, the translations of $\mathtt{zero}$, $\mathtt{succ}(e)$, and $\mathtt{ifz}\ e\,\{\,e_0 \mid x.e_1\}$ are given as follows:

$$\text{ZERO}\ \frac{}{|\Gamma| \vdash_{\mathsf{bv}} \mathtt{susp}(\mathtt{fold}(1 \cdot \mathtt{zero})) : |\mathtt{nat}|} \qquad \text{SUCC}\ \frac{|\Gamma| \vdash_{\mathsf{bv}} |e| : |\mathtt{nat}|}{|\Gamma| \vdash_{\mathsf{bv}} \mathtt{susp}(\mathtt{fold}(r \cdot \mathtt{succ}(|e|))) : |\mathtt{nat}|}$$

$$\text{IFZ}\ \frac{|\Gamma| \vdash_{\mathsf{bv}} |e| : |\mathtt{nat}| \qquad |\Gamma| \vdash_{\mathsf{bv}} |e_0| : |\tau| \qquad |\Gamma|, x : |\mathtt{nat}| \vdash_{\mathsf{bv}} |e_1| : |\tau|}{|\Gamma| \vdash_{\mathsf{bv}} \mathtt{ifz}\ \mathtt{unroll}(|e|)\,\{\,|e_0| \mid x\,.\,|e_1|\} : |\tau|}$$

Observe that, in combination with fixed points, it is possible to form the "natural number" consisting of an infinite iteration of successors! That is why it is not appropriate, contrary to practice, to call this type $\mathtt{nat}$ in the lazy setting. As an exercise, work out the translation of $\mathtt{fix}(x\,.\,\mathtt{succ}(x))$ to see how this situation is represented under the present interpretation.

The translation of an application $\mathtt{ap}(e\ ;\ e_2)$ is given by a combination of unrolling and by-value application:

$$\text{APP}\ \frac{|\Gamma| \vdash_{\mathsf{bv}} |e| : |\tau_2 \to \tau| \qquad |\Gamma| \vdash_{\mathsf{bv}} |e_2| : |\tau_2|}{|\Gamma| \vdash_{\mathsf{bv}} \mathtt{ap}(\mathtt{unroll}(|e|)\ ;\ |e_2|) : \tau}$$

The "trick" of passing divergence as an argument is achieved by ensuring that divergent expressions are translated to values representing unevaluated computations.

The translation of λ-abstractions given by the rule

$$\text{ABS}\ \frac{|\Gamma|, x : |\tau_1| \vdash_{\mathsf{bv}} |e_2| : |\tau_2|}{|\Gamma| \vdash_{\mathsf{bv}} \mathtt{susp}(\lambda(x\,.\,|e_2|)) : \mathtt{self}(|\tau_1| \rightharpoonup |\tau_2|)}$$

Thus, abstractions are translated as (non-self-referential) suspensions, which are values.

As an exercise, extend the above translation to product and sum types, observing carefully where suspensions are required. In particular examine the translation of the type $1 + 1$, which looks as though it ought to be the type of Booleans—but it is not! As with any type, one value of this type is a suspended divergent computation, which is a value, ruining the inductive character of the Booleans as the least type containing true and false. Going a bit further, extend the translation of recursive types, and compare the translation of $\mathtt{rec}(t \,.\, 1 + t)$ to the translation of `nat` given above: they are not the same.

It is also worth contemplating why there can be no "reverse" translation of by-value into by-name. There is a serious loss of expressive power in the by-name setting relative to the by-value setting that cannot be recovered by translation. In short, *by-name is a mode of use of by-value, and not conversely.* Put in other terms, the by-value variant of **PCF** is *strictly more expressive* than the by-name variant.

References

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