Reasoning with Continuations II: Full Abstraction for Models of Control

Two Corrections

Dorai Sitaram Matthias Felleisen

Department of Computer Science
Rice University
Houston, TX 77251-1892

July 29, 1992

1 Environments for PCF_v

The original paper [1] defines the domain of environments for call-by-value PCF as type-respecting maps from the set of variables to domains (cf. [1, page 166]):

$$\rho \in \mathit{Env} \text{ implies } \rho(x^t) \in \mathbb{D}_t.$$

Kurt Sieber [Universität des Saarlandes; personal communication: June 29, 1990] discovered that this definition is wrong as the following example demonstrates. Clearly,

$$(\mathbf{if0}\ x\ \ulcorner 1 \urcorner\ \ulcorner 1 \urcorner) \simeq \ulcorner 1 \urcorner$$

because x must be bound by some surrounding $\lambda\, x.$-abstraction, which in turn implies that it cannot represent a diverging computation. On the other hand, an environment could map x to bottom and then the denotations of the two expressions would differ.

Solution. The correct way to solve this problem is to restrict the set of environments to type-respecting finite maps from variables to non-bottom elements of the appropriate domain:

$$\rho \in \mathit{Env} \text{ implies } \rho(x^t) \in \mathbb{D}_t \setminus \{\perp\}$$

provided x^t is in the domain of ρ. It is easy to check that this set forms a Scott-domain under the usual ordering on finite functions, i.e., environments with the same domain are ordered component-wise and the empty environment is below every other environment.

An alternative solution, employed by Riecke [2], is to change the language and to include a syntactic construct for declaring recursive objects:

$$\mu x.M.$$

It is then possible to bind a diverging computation to a variable.

2 The CPS model for PCF_v-C

The original paper [1] assigns the following semantics to Y (cf. [1, page 170]):

$$\begin{array}{lcl}\mathsf{fix}_{\mathrm{cps}} & : & \mathbb{D}_{((s\to t)\to(s\to t))\to(s\to t)} \\ \mathsf{fix}_{\mathrm{cps}} & = & \underline{\lambda} f,\kappa.\bigsqcup_{i=0}^{\infty}\{a_i\} \\ & & \quad \text{where } a_0 = f\bot\kappa, a_1 = f\bot(\underline{\lambda} n.fn\kappa), a_2 = f\bot(\underline{\lambda} n.fn(\underline{\lambda} n.fn\kappa)), \quad \ldots\end{array}$$

Thomas Streicher [Universität München; personal communication: June 29, 1992] points out that the set $\{a_i\}$ is not necessarily a directed set and, hence, does not necessarily have a least upper bound. For a counter-example take

$$fnk = \begin{cases} 1 & (kX) = 0 \\ 0 & (kX) = 1 \\ \bot & \text{otherwise} \end{cases}$$

for some element X. Then, taking $\kappa = \underline{\lambda} x.0$, we get

$$\begin{array}{lclcl} a_0 & = & f\bot\kappa & = & 1 \\ a_1 & = & f\bot(\underline{\lambda} n.fn\kappa) & = & 0 \\ a_2 & = & f\bot\kappa & = & 1 \\ \ldots \end{array}$$

One procedure that denotes the above function is

```
(Y (lambda f.
     (call/cc (lambda (k)
        (if0 (% (k X))
             (abort 1)
             (if0 (sub1 (% (k X)) (abort 0) Omega))))))
```

A context that creates an appropriate continuation is

```
let x = [ ] in 0
```

The above code shows that the problematic procedure passed to Y is not a true second-order procedure but a procedure that manipulates its continuation in a non-standard way.

Solution 1. If we restrict the input to $\mathsf{fix}_{\mathrm{cps}}$ to functions f such that

$$fx\kappa = \kappa(fx\mathbf{I})$$

for all inputs x and continuations κ, then it is easy to show that the series a_i is ascending:

$$\begin{array}{lcll} a_0 & = & f\bot\kappa & \\ & = & \kappa(f\bot\mathbf{I}) & \\ a_{i+1} & = & f\bot(\underline{\lambda} n.fn\cdots) & \\ & = & (\underline{\lambda} n.fn\cdots)(f\bot\mathbf{I}) & \\ & = & f(f\bot\mathbf{I})\cdots & \quad [\text{continuations for } f \text{ are non-strict}] \\ & \sqsupseteq & \kappa(f\bot\mathbf{I})\cdots & \\ & = & a_i & \end{array}$$

To enforce this solution on the syntactic level, we require that Y can only be applied to procedures of the form $\lambda f.\lambda x.\cdots$. If applied to any other procedure, Y diverges. While this solution is feasible and covers all *relevant* cases, it is overly restrictive.

Solution 2. An alternative solution relies on the syntactic (operational) definition of Y:

$$(Yf) = f(\lambda x.(Yf)x).$$

A direct translation of this equation into a semantic constraint for $\mathsf{fix}_{\mathrm{cps}}$ yields the following:

$$(\mathsf{fix}_{\mathrm{cps}} f) = f(\underline{\lambda} x.(\underline{\lambda}\kappa.(\mathsf{fix}_{\mathrm{cps}} f)(\underline{\lambda} g.(gx)\kappa))),$$

i.e., $\mathsf{fix}_{\mathrm{cps}}$ is the least element M satisfying the equation

$$M = \underline{\lambda} f.f(\underline{\lambda} x.(\underline{\lambda}\kappa.(Mf)(\underline{\lambda} g.(gx)\kappa))).$$

Thus, $\mathsf{fix}_{\mathrm{cps}}$ is determined by a simple fixpoint equation on an ordinary domain. By solving this equation iteratively, we get the desired solution:

$$\mathsf{fix}_{\mathrm{cps}} = \bigsqcup_i ((\underline{\lambda} F.\underline{\lambda} f.f(\underline{\lambda} x.(\underline{\lambda}\kappa.(Ff)(\underline{\lambda} g.(gx)\kappa))))^i \bot)$$

Acknowledgement. We gratefully acknowledge discussions with Amr Sabry and Thomas Streicher on the topic of fixpoints in CPS models.

References

1. Sitaram, D. and M. Felleisen. Reasoning with continuations II: Full abstraction for models of control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, 1990, 161–175.

2. Riecke, J.G.. Fully abstract translations between functional languages. In *Proc. 18th ACM Symposium on Principles of Programming Languages*, 1991, 245-254.