Denotational Semantics of PCF

Natalie Ravenhill

July 11, 2016

1 Denotation of Types

The Denotational Semantics maps the types of PCF to a domain representing that type. We define a function:

$$[\![-]\!] : Type \to Domain$$

that maps a type to a Domain. We have two possible ways to define a type, so there are two domains we use:

1. The type of Natural numbers is the ground type, so they are modelled by a single domain. We use the flat domain of Natural numbers, where $\perp$ represents a term that loops forever.

$$[\![Nat]\!] = \mathbb{N}_\perp$$

2. Function types are formed of other types. We model them using the domain of continuous functions.

$$[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$$

2 Denotation of Typing Contexts

The Denotational Semantics maps the terms of PCF to a domain. We define a function:

$$[\![-]\!]_{Ctx} : Context \to Domain$$

that maps a typing context to a domain. The domain will be a nested tuple, the size of which depends on the number of variables in Γ. We prove separately that products of domains are also domains.

The empty context is given by

$$[\![\cdot]\!]_{Ctx} = \mathbb{1}$$

the single element set. We also prove separately that this is a domain.

Adding a variable to a context Γ gives us the following:

$$[\![\Gamma, x : A]\!]_{Ctx} = [\![\Gamma]\!] \times [\![A]\!]$$

The products of the domains give all combinations of all possible values of each variable. If we want a specific valuation of the variables, we can refer to $\gamma \in [\![\Gamma]\!]_{Ctx}$.

3 Denotation of well typed terms

Given a well typed term $\Gamma \vdash e : A$ we have

$$[\![\Gamma \vdash e : A]\!] \in [\![\Gamma]\!]_{Ctx} \rightarrow [\![A]\!]$$

So $[\![\Gamma \vdash e : A]\!]\gamma$ gives us an element of $[\![A]\!]$. We can define this on each possible value of e individually:

Variables Given a context $\Gamma = x_0 : A_0, \ \ldots \ , x_n : A_n$, $[\![\Gamma]\!]_{Ctx}$ maps a tuple γ in $[\![A_0]\!] \times \cdots \times [\![A_n]\!]$ to a value in $[\![A_i]\!]$:

$$[\![\Gamma \vdash x_i : A_i]\!] = \lambda\gamma \in [\![\Gamma]\!].\pi_i(\gamma)$$

We use the ith projection function to get the value of the ith variable in the context.

Zero z is an element of Nat, the domain of which we have defined to be $\perp$. As z is a constant, we always map it to the same value, which is 0, no matter what γ is:

$$[\![\Gamma \vdash z : Nat]\!]\gamma = 0$$

Successor When $\Gamma \vdash s(e) : Nat$ is a well typed term, then so is $\Gamma \vdash e : Nat$, so we can use $[\![\Gamma \vdash e : Nat]\!]$ in the definition of the denotational semantics for successor. As the domain of e is $\mathbb{N}_\perp$, we must consider the case where e maps to $\perp$, for which we would also have to map $s(e)$ to $\perp$:

$$[\![\Gamma \vdash s(e) : Nat]\!]\gamma = \text{Let } v = [\![\Gamma \vdash e : Nat]\!]\gamma \text{ in}$$

$$\begin{cases} v+1 & \text{if } v \neq \perp \\ \perp & \text{if } v = \perp \end{cases}$$

Case When $\Gamma \vdash case\ (e, z \mapsto e_0, s(x) \mapsto e_S) : C$ is a well typed term, then so is $\Gamma \vdash e : Nat$, so we can use $[\![\Gamma \vdash e : Nat]\!]$ in the definition of the denotational semantics for case:

$$[\![\Gamma \vdash case\ (e, z \mapsto e_0, s(x) \mapsto e_S) : C]\!]\gamma = \text{Let } v = [\![\Gamma \vdash e : Nat]\!]\gamma \text{ in}$$

$$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v = 0 \\ [\![\Gamma \vdash e_S : C]\!]\gamma & \text{if } v = n+1 \\ v+1 & \text{if } v \neq \perp \end{cases}$$

Application In this rule we already have a denotation for the function and for the element we are applying it to. The bottom element of our domain of functions is the function that loops on all inputs, $\lambda x \in X.\perp_Y$. Therefore the value of f will always be a function. Functions on domains can be applied to bottom elements, so we can still have $f(v)$ when $v = \perp$. Therefore there is only one case for function application:

$$\begin{aligned} [\![\Gamma \vdash e\ e' : B]\!]\gamma = \text{Let } f = [\![\Gamma \vdash e : A \to B]\!]\gamma \text{ in} \\ \text{Let } v = [\![\Gamma \vdash e' : A]\!]\gamma \\ \text{in } f(v) \end{aligned}$$

λ abstraction For λ abstraction, by its typing rule, we already have a denotation for $[\![\Gamma, x : A \vdash e : B]\!]\gamma$. This is a function of type $[\![\Gamma]\!] \times [\![A]\!] \to [\![B]\!]$. The function we want to obtain is of type $[\![\Gamma]\!] \to ([\![A \to B]\!])$, so we must return a continuous function. We use currying, with our denotation of $\Gamma, x : A \vdash e : B$. As this is in a different context, we need our function to be in a context where the value of x is our $a \in [\![A]\!]$ that is the argument to our function, which is $(\gamma, a/x)$:

$$[\![\Gamma \vdash \lambda x : A.e : A \to B]\!]\gamma = \lambda a \in [\![A]\!].[\![\Gamma, x : A \vdash e : A]\!](\gamma, a/x)$$

Fixpoint For fixpoint, by its typing rule we already have a denotation for $[\![\Gamma, x : A \vdash e : A]\!]\gamma$ This is a function of type $[\![\Gamma]\!] \times [\![A]\!] \to [\![A]\!]$. The function we want to obtain is of type $[\![\Gamma]\!] \to [\![A]\!]$ To get an element of $[\![A]\!]$, we use the fixpoint function, $fix_{[\![A]\!]}$, which is a continuous function of type $([\![A]\!] \to [\![A]\!]) \to [\![A]\!]$. the function we give to the fixpoint is the one that maps any given $a \in [\![A]\!]$ to the denotation of $\Gamma, x : A \vdash e : A$ in a context where a is the value of x:

$$[\![\Gamma \vdash fix\ x : A.e : A]\!]\gamma = fix_{[\![A]\!]}(\lambda a \in [\![A]\!].[\![\Gamma, x : A \vdash e : A]\!](\gamma, a/x)$$