Denotational Semantics of PCF

Natalie Ravenhill

July 14, 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

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

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 $\Gamma \vdash case\ (e, z \mapsto e_0, s(y) \mapsto e_S) : C$ 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:

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

$$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v = 0 \\ [\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, n/y) & \text{if } v = n+1 \\ \perp & \text{if } v = \perp \end{cases}$$

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:

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

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:

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

4 Theorems

4.1 Substitution Theorem

The following theorem says that given a well typed expression e and another expression e', which is well typed in the context with $x : A$ added, then the denotation of e' with e substituted for x is the same as the denotation of the original expression in the context with $x : A$ added and valuation with the denotation of e as the value of x:

Theorem 1. *If* $\Gamma \vdash e : A$ *and* $\Gamma, x : A \vdash e' : C$ *and* $\gamma \in [\![\Gamma]\!]$*, then* $[\![\Gamma \vdash [e/x]e' : C]\!]\gamma = [\![\Gamma, x : A \vdash e' : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

Proof. We prove this by induction on the value of e':

Variables There are two cases for variables:

1. For a variable $x : C$, C must be equal to A, so we get $[\![\Gamma \vdash [e/x]x : A]\!]\gamma = [\![\Gamma \vdash e : A]\!]\gamma$, from the substitution rule.

   On the right hand side, $[\![\Gamma, x : A \vdash x : A]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x) = \pi_i(\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$. The value of this is the value of x, which is $[\![\Gamma \vdash e : A]\!]\gamma$.

   Therefore $[\![\Gamma \vdash [e/x]x : A]\!]\gamma = [\![\Gamma, x : A \vdash x : A]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x) = [\![\Gamma \vdash e : A]\!]\gamma$

2. For a variable $y : C$, we have $[\![\Gamma \vdash [e/x]y : C]\!]\gamma = [\![\Gamma \vdash y : C]\!]\gamma$, by the substitution rule for variables. This is equal to $\pi_i(\gamma)$, where $y : C$ is the ith element of Γ. If we extend the context Γ with $x : A$ and the valuation γ with $[\![\Gamma \vdash e : A]\!]\gamma/x$, then this does not affect $\pi_i(\gamma)$, as each variable is

independent. Therefore $[\![\Gamma \vdash [e/x]y : C]\!]\gamma = [\![\Gamma, x : A \vdash y : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$.

Zero By the substitution rule for zero, $[\![\Gamma \vdash [e/x]z : Nat]\!]\gamma = [\![\Gamma \vdash z : Nat]\!]\gamma$. As z is a constant, its denotation will be the same for any Γ and γ, so we always get 0. Therefore $[\![\Gamma \vdash [e/x]z : Nat]\!]\gamma = [\![\Gamma, x : A \vdash z : Nat]\!](\gamma, [\![\Gamma, \vdash e : A]\!]\gamma/x) = 0$.

Successor Using the substitution rule, $[\![\Gamma \vdash [e/x]s(e') : Nat]\!]\gamma = [\![\Gamma \vdash s([e/x]e') : Nat]\!]\gamma$. The induction hypothesis is $[\![\Gamma \vdash [e/x]e' : C]\!]\gamma = [\![\Gamma, x : A \vdash e' : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$, so we can use this to rewrite $[\![\Gamma \vdash s([e/x]e') : Nat]\!]\gamma$ as the following function:

Let $v = [\![\Gamma, x : A \vdash e' : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$ in

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

This function is also the definition of $[\![\Gamma, x : A \vdash s(e') : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$.

Therefore $[\![\Gamma \vdash [e/x]s(e') : Nat]\!]\gamma = [\![\Gamma, x : A \vdash s(e') : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

Case Using the substitution rule for case, $[\![\Gamma \vdash [e/x](case\ (e', z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma = [\![\Gamma \vdash (case\ ([e/x]e', z \mapsto [e/x]e_0, s(y) \mapsto [e/x]e_S) : C]\!]\gamma$. We can use induction on all the expressions with substitutions to get the following definition of $[\![\Gamma \vdash (case\ ([e/x]e', z \mapsto [e/x]e_0, s(y) \mapsto [e/x]e_S) : C]\!]\gamma$:

Let $v = [\![\Gamma, x : A \vdash e' : Nat]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$ in

$$\begin{cases} [\![\Gamma, x : A \vdash e_0 : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x) & \text{if } v = 0 \\ [\![\Gamma, y : Nat, x : A \vdash e_S : C]\!](\gamma, n/y, [\![\Gamma \vdash e : A]\!]\gamma/x) & \text{if } v = n+1 \\ \bot & \text{if } v = \bot \end{cases}$$

This function is also the definition of $[\![\Gamma, x : A \vdash [e/x](case\ (e', z \mapsto e_0, s(v) \mapsto e_S)) : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

Therefore $[\![\Gamma \vdash [e/x](case\ (e', z \mapsto e_0, s(v) \mapsto e_S) : C]\!]\gamma = [\![\Gamma, x : A \vdash case\ (e', z \mapsto e_0, s(v) \mapsto e_S) : C]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

Application Using the substitution rule for application, $[\![\Gamma \vdash [e/x](e_0\ e_1) : B]\!]\gamma = [\![\Gamma \vdash [e/x]e_0([e/x]e_1) : B]\!]\gamma$. We can use induction on $[\![\Gamma \vdash [e/x]e_0 : A \to B]\!]$ and $[\![\Gamma \vdash [e/x]e_1 : A]\!]$ to rewrite the denotation as the following:

Let $f = [\![\Gamma, x : A \vdash e_0 : A \to B]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$ in

$$\text{Let } v = [\![\Gamma, x : A \vdash e_1 : A]\!](\gamma[\![\Gamma \vdash e : A]\!]\gamma/x)$$

$$\text{in } f(v)$$

This function is also the definition of $[\![\Gamma, x : A \vdash e_0\ e_1 : B]\!](\gamma[\![\Gamma \vdash e : A]\!]\gamma/x)$.

Therefore, $[\![\Gamma \vdash [e/x](e_0\ e_1) : B]\!]\gamma = [\![\Gamma, x : A \vdash e_0\ e_1 : B]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

λ **abstraction** Using the substitution rule we have $[\![\Gamma \vdash [e/x](\lambda y : A.e') : A \to B]\!]\gamma = [\![\Gamma \vdash \lambda y : A.([e/x]e') : A \to B]\!]\gamma$. We can use induction with $[\![\Gamma \vdash \lambda[e/x]e' : B$, to rewrite the denotation as the following:

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

This is also the definition of $[\![\Gamma, x : A \vdash \lambda y : A.\ e' : A \to B]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$

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

Fixpoint Using the substitution rule for fixpoint, $[\![\Gamma \vdash [e/x](fix\ y : C.e' : C)]\!]\gamma = [\![\Gamma \vdash fix\ y : C.[e/x]e' : C]\!]\gamma$. We can use induction on $[\![\Gamma, y : C \vdash [e/x]e' : C]\!]$ to rewrite the denotation as the following:

$$fix_{[\![C]\!]}(\lambda c \in [\![C]\!].[\![\Gamma, y : C, x : A \vdash e' : C]\!](\gamma, c/y, [\![\Gamma \vdash e : A]\!]\gamma/x)$$

This is also the definition of $[\![\Gamma, x : A \vdash (fix\ y : A.e' : A)]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$.

Therefore $[\![\Gamma \vdash [e/x](fix\ y : C.e' : C)]\!]\gamma = [\![\Gamma, x : A \vdash (fix\ y : C.e' : C)]\!](\gamma, [\![\Gamma \vdash e : A]\!]\gamma/x)$.

Now we have proved the theorem for every case of e'. $\square$

4.2 Correctness

The following theorem says that for a well typed expression e, if it maps to another expression e', then its denotation will be equal to that of the new expression in the same context:

Theorem 2. *If* $\Gamma \vdash e : A$ *and* $e \mapsto e'$ *and* $\gamma \in [\![\Gamma]\!]$, *then* $[\![\Gamma \vdash e : A]\!]\gamma = [\![\Gamma \vdash e' : A]\!]\gamma$

Proof. By induction on $e \mapsto e'$, so there is a case for each evaluation rule:

Variables have no rules, so there are no cases here.

Zero is a value, so it has no evaluation rules. Therefore there are no cases for zero.

Successor We use a congruence rule for successor, so when $s(e) \mapsto s(e')$ we also know that $e \mapsto e'$. From $\Gamma \vdash s(e) : Nat$, we know that $\Gamma \vdash e : Nat$. Therefore we can use induction on this to get $[\![\Gamma \vdash e : A]\!]\gamma = [\![\Gamma \vdash e' : A]\!]\gamma$.

We can use this to rewrite $[\![\Gamma \vdash s(e) : Nat]\!]\gamma$ as:

Let $v = [\![\Gamma \vdash e' : Nat]\!]\gamma$ in

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

Which is the same as $[\![\Gamma \vdash s(e') : Nat]\!]\gamma$

Case There are three cases for case:

1. When e is an expression that can be reduced, we use a congruence rule for case, so when $case\ (e, z \mapsto e_0, s(y) \mapsto e_S) \mapsto case\ (e', z \mapsto e_0, s(y) \mapsto e_S)$ we also know that $e \mapsto e'$. From $\Gamma \vdash case\ (e, z \mapsto e_0, s(y) \mapsto e_S) : C$, we know that $\Gamma \vdash e : Nat$. Therefore we can use induction to get $[\![\Gamma \vdash e : Nat]\!]\gamma = [\![\Gamma \vdash e' : Nat]\!]\gamma$.

   We can use this to rewrite $[\![\Gamma \vdash case\ (e, z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma$ as:

   Let $v = [\![\Gamma \vdash e' : Nat]\!]\gamma$ in

   $$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v = 0 \\ [\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, n/y) & \text{if } v = n+1 \\ \bot & \text{if } v = \bot \end{cases}$$

   Which is the same as $[\![\Gamma \vdash case\ (e', z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma$.

2. When $e = z$, we have $[\![\Gamma \vdash case(z, z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma$ which is:

   Let $v = [\![\Gamma \vdash z : Nat]\!]\gamma$ in

$$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v = 0 \\ [\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, n/y) & \text{if } v = n+1 \\ \perp & \text{if } v = \perp \end{cases}$$

   As $[\![\Gamma' \vdash z : Nat]\!]\gamma$ is always 0, this can be simplified to $[\![\Gamma \vdash e_0 : C]\!]\gamma$, which is the result of the evaluation rule

3. When $e = s(v)$, we have $[\![\Gamma \vdash case(s(v), z \mapsto e_0, s(y) \mapsto e_S) : C]\!]\gamma$ which is:

   Let $v' = [\![\Gamma \vdash s(v) : Nat]\!]\gamma$ in

$$\begin{cases} [\![\Gamma \vdash e_0 : C]\!]\gamma & \text{if } v' = 0 \\ [\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, n/y) & \text{if } v' = v+1 \\ \perp & \text{if } v' = \perp \end{cases}$$

   where $n = [\![\Gamma \vdash v : Nat]\!]\gamma$.

   There are two possibilities for the value of v'.

   (a) If $v' = \perp$ then the function will return $\perp$

   (b) Otherwise $v' = n + 1$, where $n = [\![\Gamma \vdash v : Nat]\!]\gamma$. With this we can simplify the definition of the expression to $[\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, n/y)$

   This is the same as:

$$[\![\Gamma, y : Nat \vdash e_S : C]\!](\gamma, [\![\Gamma \vdash v : Nat]\!]\gamma/y)$$

   Using substitution, we get $[\![\Gamma \vdash [v/y]e_S]\!]\gamma$.

Application There are two cases for application:

1. We use a congruence rule for function application, so when $e_0\ e_1 \mapsto e_0'\ e_1$ we also know that $e \mapsto e'$. From $\Gamma \vdash e_0\ e_1 : B$, we know that $\Gamma \vdash e_0 : A \to B$. Therefore we can use induction on this to get $[\![\Gamma \vdash e_0\ e_1 : A]\!]\gamma = [\![\Gamma' \vdash e_0'\ e_1 : A]\!]\gamma$.

   We can use this to rewrite $[\![\Gamma \vdash e_0\ e_1 : B]\!]\gamma$ as:

   Let $f = [\![\Gamma \vdash e_0' : A \to B]\!]\gamma$ in

   $$\text{Let } v = [\![\Gamma \vdash e_1 : A]\!]\gamma$$
   $$\text{in } f(v)$$

   which is the same as $[\![\Gamma \vdash e_0'\ e_1 : B]\!]\gamma$

2. When $e = \lambda x : A.e$, it is a value, so cannot be reduced further by the congruence rule. We use the semantic rule:

   $$\frac{}{(\lambda x : A.\ e)\ e' \mapsto [e'/x]e}$$

   so we need a denotation $[\![\Gamma \vdash [e'/x]e]\!]\gamma$.

   As we have the denotation of $[\![\Gamma \vdash (\lambda x : A.e)\ e' : B]\!]\gamma$, we have $f = [\![\Gamma \vdash \lambda x : A.e : A \to B]\!]\gamma$ and $v = [\![\Gamma \vdash e' : A]\!]\gamma$.

   $f = \lambda a \in [\![A]\!].[\![\Gamma, x : A \vdash e : A]\!](\gamma, a/x)$, so $f\ v$ is $[\![\Gamma, x : A \vdash e : A]\!](\gamma, [\![\Gamma \vdash e' : A]\!]\gamma/x)$

   By substitution, this is the same as $[\![\Gamma \vdash [e'/x]e]\!]\gamma$

λ abstraction is a value, so has no evaluation rules. Therefore there are no cases.

Fixpoint As $[\![\Gamma \vdash fix\ x : A.e : C]\!]\gamma$ is a fixpoint operator, we know that $f(fix(f)) = fix(f)$, so we can rewrite it as:

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

which is equal to:

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

which is equal to:

$$[\![\Gamma, x : A \vdash e : C]\!](\gamma, [\![\Gamma \vdash fix\ x : A.e : C]\!]\gamma/x)$$

Using the substitution lemma, this is the same as

$$[\![\Gamma \vdash [fix\ x : A.e/x]e : C]\!]\gamma$$

The evaluation rule is

$$\frac{}{fix\ x : A.e \mapsto [fix\ x : A.e/x]e}$$

so this is the denotation we need. $\square$