# Formal Programming Language Semantics note 13

## A simple functional language

In this note we introduce a simple functional language which corresponds to a certain "core" fragment of languages such as Standard ML and Haskell. This language (and minor variants thereof) is known in the research literature as **PCF** (Programming language for Computable Functions), and was first studied by Scott, Plotkin and Milner in the '70s. We will first give a denotational semantics using CPOs, and later provide an operational semantics to match. This is true to the way in which the ideas actually developed. Indeed, the language **PCF** was largely inspired by the CPO model; the operational semantics came later, and the ideas had a major influence on the design of Standard ML.

**Syntax and static semantics of PCF.** The *types* $\tau$ of **PCF** are given by:

$$\tau \ ::= \ \texttt{int} \mid \texttt{bool} \mid \tau_0 \texttt{ -> } \tau_1.$$

We think of $\tau_0$ `->` $\tau_1$ as the type of *functions* from type $\tau_0$ to type $\tau_1$.

We assume we have an infinite supply of *variables* `x`, `y`, `z`, ..., and use $x$ as a metavariable ranging over these (note the difference in typeface!). We also retain our earlier conventions that $n$ ranges over the set $\mathbb{Z}$ of integers, and $t$ over the set $\mathbb{T}$ of truth values.

The syntax of *expressions* $e$ of **PCF** is given as follows:

$$\begin{array}{rl} e ::= & n \mid t \mid e_0 \texttt{ - } e_1 \mid e_0 \texttt{ = } e_1 \mid \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 \\ & \mid x \mid e_0\, e_1 \mid \texttt{fn } x \texttt{ : } \tau \texttt{ => } e \mid \texttt{fix (} x \texttt{ : } \tau \texttt{ = } e \texttt{)} \end{array}$$

Apart from the `fix` construct, we have used ML syntax here. (As before, we will not be too fussy about fine syntactic details, adding brackets whenever we feel like it.) Note that the `if` construct is now a form of expressions rather than commands; it is essentially the conditional expression form $e_0 ? e_1 : e_2$ of C or Java. The expression form $e_0\, e_1$ represents *function application*; here $e_0$ must be a function of an appropriate type, and $e_1$ is its argument. The form `fn` $x \texttt{ : } \tau \texttt{ => } e$ is called *abstraction* — it can be read as "the function which takes a value $x$ of type $\tau$ to $e$", where the expression $e$ typically involves $x$. (In lambda calculus notation, this would be written as $\lambda x : \tau.\, e$.) The form `fix (`$x \texttt{ : } \tau \texttt{ = } e$`)` can be read as "the least solution to the recursion equation $x = e$" (where again $e$ typically involves $x$); this form is inspired by the fact that we know how to solve such equations in the world of CPOs.

To make the language feel more like ML, we may also add a layer of *declarations* $d$ of the form

$$d ::= \texttt{val } x \texttt{ = } e$$

allowing a variable $x$ to be bound to an expression $e$ for later use. We may then regard ML-style function declarations like `fun f` $x$ `=` $e$ (where $e$ may involve `f` as well as $x$) as syntactic sugar for

$$\texttt{val f = fix (f:}\tau_0\texttt{ ->}\tau_1\texttt{ = fn }x\texttt{:}\tau_0\texttt{=>}e\texttt{)}$$

(for suitable types $\tau_0$ and $\tau_1$). Similarly for functions of several variables. Many other ML constructs, such as *pattern matching* in function declarations, can also be seen as sugar for suitable **PCF** expressions.

Not all the expressions generated by the above grammar are well-typed. The typing rules below specify which are the well-formed expressions of PCF. As with **IMP**$^b$, we use the notion of a *static environment* $[(x_1,\tau_1),\ldots,(x_r,\tau_r)]$ associating a type to finitely many variables. We use $\Gamma$ to range over static environments, and use the notation $\Gamma(x)$ with the same meaning as in Note 7. The following rules allow us to derive assertions of the form $\Gamma \vdash e : \tau$, meaning "in environment $\Gamma$, $e$ is a well-formed PCF expression of type $\tau$".

$$\frac{}{\Gamma \vdash n : \texttt{int}} \qquad \frac{}{\Gamma \vdash t : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_0 : \texttt{int} \quad \Gamma \vdash e_1 : \texttt{int}}{\Gamma \vdash e_0\ \texttt{-}\ e_1 : \texttt{int}} \qquad \frac{\Gamma \vdash e_0 : \texttt{int} \quad \Gamma \vdash e_1 : \texttt{int}}{\Gamma \vdash e_0\ \texttt{=}\ e_1 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_0 : \texttt{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau} \qquad \frac{}{\Gamma \vdash x : \tau}\ \Gamma(x) = \tau$$

$$\frac{\Gamma \vdash e_0 : \tau\ \texttt{->}\ \tau' \quad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0\, e_1 : \tau'} \qquad \frac{\Gamma;(x:\tau) \vdash e : \tau'}{\Gamma \vdash \texttt{fn}\ x\texttt{:}\tau\texttt{=>}e\ \texttt{:}\tau\ \texttt{->}\ \tau'}$$

$$\frac{\Gamma;(x:\tau) \vdash e : \tau}{\Gamma \vdash \texttt{fix (}x\texttt{:}\tau\texttt{ = }e\texttt{)} : \tau}$$

Finally, the elaboration of a declaration in a certain static environment has the effect of "returning" an extended static environment:

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{val}\ x\ \texttt{=}\ e \Rightarrow \Gamma;(x,\tau)}$$

**Denotational semantics of PCF.** We now give a denotational semantics for **PCF** using CPOs. This will not be too difficult, since the language was in a sense designed precisely to be a convenient notation for denoting elements of CPOs. In our interpretation, every type $\tau$ will be interpreted as a CPO $[\![\tau]\!]$, and every (closed) expression $e : \tau$ will be interpreted as an element $[\![e]\!]$ of $[\![\tau]\!]$.

We first define the interpretation of types. For `int` and `bool`, we will use the CPOs $\mathbb{Z}_\perp$ and $\mathbb{T}_\perp$ defined in Note 11.

$$[\![\texttt{int}]\!] = \mathbb{Z}_\perp, \qquad [\![\texttt{bool}]\!] = \mathbb{T}_\perp.$$

For function types, we would naturally like $[\![\tau_0\ \texttt{->}\ \tau_1]\!]$ to be some set of functions from $[\![\tau_0]\!]$ to $[\![\tau_1]\!]$. The following fact about CPOs gives us what we need:

**Proposition 1** *Let $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$ be CPOs. Define $D \Rightarrow E$ to be the set of all* continuous *functions $f : D \to E$, and let $\sqsubseteq$ be the relation on $D \Rightarrow E$ defined by*

$$f \sqsubseteq g \iff \forall x \in D.\ f(x) \sqsubseteq_E g(x).$$

*Then $(D \Rightarrow E, \sqsubseteq)$ is itself a CPO. Furthermore, if $E$ has a least element $\bot$, then $D \Rightarrow E$ has least element $\Lambda x.\bot$ (which we also write as $\bot$).*

We may therefore complete the interpretation of PCF types by defining

$$[\![\, \tau_0 \texttt{->} \tau_1 \,]\!] \;=\; [\![\, \tau_0 \,]\!] \Rightarrow [\![\, \tau_1 \,]\!].$$

Next, we define the denotation of an environment $\Gamma = [(x_1, \tau_1), \ldots, (x_r, \tau_r)]$ to be

$$[\![\, \Gamma \,]\!] \;=\; [\![\, \tau_1 \,]\!] \times \cdots \times [\![\, \tau_r \,]\!].$$

We have not officially defined what we mean by a product of CPOs, but it is fairly obvious. [Exercise: fill in this gap!] Elements of $[\![\, \Gamma \,]\!]$ play a role somewhat analogous to the dynamic environments or states in Note 7 — they assign actual denotations (rather than just types) to the variables in $\Gamma$. We will use $\vec{z}$ or $(z_1, \ldots, z_r)$ to range over $[\![\, \Gamma \,]\!]$.

A closed term $e : \tau$ (one with no free variables) will simply denote an element of $[\![\, \tau \,]\!]$. However, if $e : \tau$ involves free variables drawn from $\Gamma$, the meaning of $e$ will clearly depend on the values assigned to the free variables. So we will define the denotation of $e$ in environment $\Gamma$ to be a certain continuous function

$$[\![\, e \,]\!]_\Gamma \;:\; [\![\, \Gamma \,]\!] \to [\![\, \tau \,]\!].$$

We will make use of the following auxiliary functions:

$$\mathit{minus} \;: \mathbb{Z}_\bot \times \mathbb{Z}_\bot \to \mathbb{Z}_\bot, \qquad (x, y) \mapsto \begin{cases} x - y & \text{if } x, y \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases}$$

$$\mathit{equals} \;: \mathbb{Z}_\bot \times \mathbb{Z}_\bot \to \mathbb{T}_\bot, \qquad (x, y) \mapsto \begin{cases} \mathit{true} & \text{if } x, y \in \mathbb{Z} \text{ and } x = y \\ \mathit{false} & \text{if } x, y \in \mathbb{Z} \text{ and } x \neq y \\ \bot & \text{otherwise} \end{cases}$$

$$\mathit{cond}_\tau \;: \mathbb{T}_\bot \times [\![\, \tau \,]\!] \times [\![\, \tau \,]\!] \to [\![\, \tau \,]\!], \quad (b, x, y) \mapsto \begin{cases} x & \text{if } b = \mathit{true} \\ y & \text{if } b = \mathit{false} \\ \bot & \text{if } b = \bot \end{cases}$$

We also define $\mathit{Fix}_\tau : [\![\, \tau \texttt{->} \tau \,]\!] \to [\![\, \tau \,]\!]$ to be the operation which maps a continuous function $f : [\![\, \tau \,]\!] \to [\![\, \tau \,]\!]$ to its least fixed point $x \in [\![\, \tau \,]\!]$. The definition of $[\![\, e \,]\!]_\Gamma$ can now be given by induction on the structure of $e$:[1]

[1] Strictly speaking, we ought to verify that the right hand side of each clause defines a *continuous* function. This will ensure that the function we are attempting to apply $\mathit{Fix}_\tau$ to in the last clause is always continuous.

$$
\begin{aligned}
[\![ n ]\!]_\Gamma(\vec{z}) &= n \\
[\![ t ]\!]_\Gamma(\vec{z}) &= t \\
[\![ e_0 \texttt{ - } e_1 ]\!]_\Gamma(\vec{z}) &= \mathit{minus}\,([\![ e_0 ]\!]_\Gamma(\vec{z}), [\![ e_1 ]\!]_\Gamma(\vec{z})) \\
[\![ e_0 \texttt{ = } e_1 ]\!]_\Gamma(\vec{z}) &= \mathit{equals}\,([\![ e_0 ]\!]_\Gamma(\vec{z}), [\![ e_1 ]\!]_\Gamma(\vec{z})) \\
[\![ \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 ]\!]_\Gamma(\vec{z}) &= \mathit{cond}_\tau((([\![ e_0 ]\!]_\Gamma(\vec{z}), [\![ e_1 ]\!]_\Gamma(\vec{z}), [\![ e_2 ]\!]_\Gamma(\vec{z})) \\
[\![ x ]\!]_\Gamma(\vec{z}) &= z_j, \text{where } j \text{ is largest s.t. } x_j \equiv x \\
[\![ e_0\, e_1 ]\!]_\Gamma(\vec{z}) &= [\![ e_0 ]\!]_\Gamma(\vec{z})([\![ e_1 ]\!]_\Gamma(\vec{z})) \\
[\![ \texttt{fn } x : \tau \texttt{ => } e ]\!]_\Gamma(\vec{z}) &= \Lambda v \in [\![ \tau ]\!].\, [\![ e ]\!]_{\Gamma;(x,\tau)}(\vec{z}, v) \\
[\![ \texttt{fix (} x : \tau \texttt{ = } e \texttt{)} ]\!]_\Gamma(\vec{z}) &= \mathit{Fix}_\tau\,(\Lambda v \in [\![ \tau ]\!].\, [\![ e ]\!]_{\Gamma;(x,\tau)}(\vec{z}, v))
\end{aligned}
$$

Finally, if $d$ is a declaration such that $\Gamma \vdash d \Rightarrow \Gamma'$, it is natural to define its denotation $[\![ d ]\!]_\Gamma$ to be a continuous function $[\![ \Gamma ]\!] \to [\![ \Gamma' ]\!]$ as follows:

$$[\![ \texttt{val } x = e ]\!]_\Gamma\,(z_1, \ldots, z_r) \;=\; (z_1, \ldots, z_r, [\![ e ]\!]_\Gamma)$$

**Another example of fixed points.** As a brief digression, let us revisit something from Note 4 — the idea of defining an evaluation relation from a set of rule instances — and show how this gives another example of CPOs and fixed points. First, for any set $A$, consider the *powerset* $\mathbb{P}(A)$ (that is, the set of all subsets of $A$), endowed with the usual subset ordering $\subseteq$. It is clear enough that this is a poset, and that it is complete: given any chain $X_0 \subseteq X_1 \subseteq \cdots$ of subsets of $A$, their *union* $\bigcup X_i$ is also a subset of $A$, and is clearly the least upper bound of the $X_i$. Thus, $(\mathbb{P}(A), \subseteq)$ is a CPO.

Now suppose $A$ is some set $V$ of "potential evaluation statements", like the $V$ we defined in Note 4. Our problem was to define a subset $E \subseteq V$ consisting of the "true evaluation statements", given some set $\mathbb{I}$ of rule instances. We can approach this problem as follows. Define an operator $J : \mathbb{P}(V) \to \mathbb{P}(V)$ by

$$J(X) \;=\; \{\phi \in V \mid \exists \phi_1, \ldots, \phi_k \in X \;\text{ s.t. }\; \frac{\phi_1 \cdots \phi_k}{\phi} \;\in \mathbb{I}\}$$

In other words, $J(X)$ is the set of things that can be proved from things in $X$ via a single rule instance. (Note that we don't necessarily have $X \subseteq J(X)$.) As an exercise, you might like to check that $J$ is monotone and continuous, so we can take its least fixed point $E \in \mathbb{P}(V)$. You should then satisfy yourself that this coincides exactly with the set $E$ we defined in Note 4.

*John Longley*