Formal Programming Language Semantics note 15

Full abstraction and universality

As we have seen, our CPO model of PCF gives us a good way of proving *observational equivalences* between PCF programs: if we can show that $[\![e]\!] = [\![e']\!]$, we know that e and e' are observationally equivalent. However, it is natural to ask whether *all* observational equivalences can in principle be established in this way. Might there be two programs e, e' which are observationally equivalent but such that $[\![e]\!] \neq [\![e']\!]$? If so, the CPO model could be regarded as somehow deficient, in that it would not offer a "complete" solution to the observational equivalence problem. One might then wonder whether some other kind of denotational model might do better.[1]

This in turn touches on a more general question: "When is a denotational semantics a good one?" — that is, in what ways might a denotational semantics might be said to enjoy a particularly "close fit" with the programming language it is intended to model? In this note we will consider these questions in the context of PCF and closely related languages, though actually the essential ideas can be applied to virtually any language for which we know how to give a denotational semantics.

The general situation we are considering is that one defines some mathematical structure M_τ for each type τ, and interprets (closed) terms $e : \tau$ as elements $[\![e]\!] \in M_\tau$. In this setting we may consider the following notions:[2]

- Such a model is *fully abstract* if for all closed terms $e, e' : \tau$, if e and e' are observationally equivalent then $[\![e]\!] = [\![e']\!]$. (This is the converse to what is given by adequacy; it expresses the idea that the model captures all possible observational equivalences.)
- A model is *universal* if for every element $x \in M_\tau$ there is a closed term $e : \tau$ such that $[\![e]\!] = x$. (This captures the idea that the model contains no "junk" that does not correspond to the behaviour of any possible program.)

Failure of universality. Is our CPO model of (call-by-name) PCF fully abstract and/or universal? It is easy to see that the model as we have defined it is not

[1] We have illustrated one particular flavour of denotational semantics, using CPOs and continuous functions. However, there are many other kinds of denotational models of PCF on the market: these include *stable domain* models, *game* models, *realizability* models, *logical relation* models and *metric space* models.

[2] Actually it is more usual to give slightly more complicated definitions of these notions involving arbitrary terms, not just closed ones. Here we have gone for slightly simpler definitions which are equivalent to the usual ones for most of the models encountered in practice.

universal, since even at type `int -> int` the model contains (uncountably many!) non-computable functions, which cannot be defined by any program. In fact, it seems clear that *any* universal model (in the above sense) will need to have some notion of computability built into it. One can in fact do this with our CPO model, and give a reasonable mathematical definition of what we mean by a *computable* element of each CPO $[\![\tau]\!]$. (The details are rather involved, and we will not give them here.) So the right question to be asking is: Is every *computable* element of each $[\![\tau]\!]$ denotable by some PCF term?

Perhaps surprisingly, the answer is still no. There is a famous counterexample known as the "parallel-or" function. Let $por \in [\![\texttt{bool -> bool -> bool}]\!]$ be the two-argument function defined by:

$$por\ x\ y \ = \ \begin{cases} \mathit{true} & \text{if } x = \mathit{true} \text{ or } y = \mathit{true} \\ \mathit{false} & \text{if } x = y = \mathit{false} \\ \bot & \text{otherwise.} \end{cases}$$

It is clear that this function is "computable" in some intuitive sense: given two arguments x, y, we may try to evaluate x and y in parallel, and if either ever returns *true* then we return *true*, regardless of whether the other one ever terminates. If one argument returns *false* then we need to keep on trying to evaluate the other argument, but if both eventually return *false* then we may return *false*. In agreement with this intuition, it turns out that *por* is indeed a computable element of $[\![\texttt{bool -> bool -> bool}]\!]$ in the mathematical sense mentioned above.

However, it also seems intuitively plausible that *por* cannot be computed by any PCF program. Intuitively, this is because PCF behaves in a "sequential" manner — any program for *por* would have to evaluate either x or y first, and could not do anything else until this evaluation has finished. The problem is that whichever argument we choose to evaluate first, if we are unlucky this argument might diverge, whilst the other argument might yield *true* and we would never discover this. This argument can in fact be made completely rigorous, and it can be proved as a theorem that *por* is not definable in PCF. Thus, even the computable version of the CPO model is not universal.

Failure of full abstraction. One can also use the parallel-or function to show that the CPO model is not fully abstract. It is possible to give two PCF terms

$$e, e' \ : \ \texttt{(bool->bool->bool)->bool}$$

whose respective denotations are functions

$$[\![e]\!], [\![e']\!] : [\![\texttt{bool -> bool -> bool}]\!] \to [\![\texttt{bool}]\!]$$

which agree on all *PCF-definable* elements $x \in [\![\texttt{bool -> bool -> bool}]\!]$, but which have different values when applied to *por*. For example, helping ourselves to some harmless syntactic sugar, we could take e, e' respectively to be

```
fn f : bool->bool->bool => f true diverge AND f diverge true
fn f : bool->bool->bool => f diverge diverge
```

Although these terms denote different functions in our model, they are observationally equivalent since the difference cannot be detected within PCF itself.

Moving the goalposts. All this might lead us to ask whether one can give an alternative denotational semantics of PCF which is better than our CPO model. However, there is another way of responding to the deficiencies of the CPO model: rather than look for another model to match the language, we can try to change the language to match the model! This may seem like moving the goalposts, but it is in any case an interesting general point that denotational models can in principle suggest possible extensions to existing programming languages. One way to interpret the failure of universality is to say that the parallel-or operator is "missing" from PCF, so let us try adding it! We will extend the grammar of PCF by adding a new expression form $\texttt{por}\ e_0\ e_1$, with the associated typing rule

$$\frac{\Gamma \vdash e_0 : \texttt{bool} \qquad \Gamma \vdash e_1 : \texttt{bool}}{\Gamma \vdash \texttt{por}\ e_0\ e_1 : \texttt{bool}}$$

We now add the following reduction rules to the operational semantics:

$$\begin{array}{rcl} \texttt{por}\ \mathit{true}\ e_1 & \to & \mathit{true} \\ \texttt{por}\ e_0\ \mathit{true} & \to & \mathit{true} \\ \texttt{por}\ \mathit{false}\ \mathit{false} & \to & \mathit{false} \end{array}$$

along with the rules: if $e \to e'$ then $\texttt{por}\ e\ e_1 \to \texttt{por}\ e'\ e_1$ and $\texttt{por}\ e_0\ e \to \texttt{por}\ e_0 e'$. Notice that this makes the reduction relation non-deterministic: given a term $\texttt{por}\ e_0\ e_1$ the computation may proceed by reducing either e_0 or e_1. However, the final *value* resulting from any such computation is still deterministic, and does not depend on the order of evaluation chosen — thus, `por` does not fundamentally alter the "functional" character of the language. We will write **PCF**$^+$ for the new language PCF+`por`.

We can also extend our denotational semantics in an obvious way, using the semantic function *por* to interpret the syntactic construct `por`. [Exercise: formulate the precise definition of $[\![\, \texttt{por}\ e_0\ e_1 \,]\!]_\Gamma$ in terms of $[\![\, e_0 \,]\!]_\Gamma$ and $[\![\, e_1 \,]\!]_\Gamma$.] Rather surprisingly, we now have the following theorem: The CPO model *is* fully abstract for **PCF**$^+$. (The proof of this is quite hard.) The point is that by expanding the language we have made it possible to perform more observations on programs, so the relation of observational equivalence has shrunk, and in fact now coincides exactly with denotational equality in the CPO model.

Although `por` solves the problem with full abstraction, it is not by itself enough to make the (computable) CPO model universal. However, we can achieve universality by adding just one further "parallel" operator to the language: a second-order gadget known as the `exists` operator. This operator allows one to perform (potentially) infinitely many tests of a certain kind in parallel, but despite its infinitary nature it is still "computable" (i.e. you really can implement it on a machine!). The typing rule associated with this operator is as follows:

$$\frac{\Gamma \vdash e : \texttt{int -> bool}}{\Gamma \vdash \texttt{exists}\ e : \texttt{bool}}$$

The reduction rules may be given as follows (we write `diverge` for the non-terminating program `fix (x:int = x)` of type `int`).

- If $e\,n \rightarrow^* \mathit{true}$ for some n, then `exists` $n \rightarrow \mathit{true}$.
- If e `diverge` $\rightarrow^* \mathit{false}$, then `exists` $e \rightarrow \mathit{false}$.

[Exercise: this presentation is slightly sloppy in that the definitions of $\rightarrow$ and $\rightarrow^*$ are now mutually recursive. Give a more rigorous formulation of what I intend to say here.]

Let us write **PCF**$^{++}$ for the language PCF extended with both `por` and `exists` . We can interpret **PCF**$^{++}$ in the CPO model, and we now have a very pleasing result: the (computable) CPO model is both fully abstract and universal for **PCF**$^{++}$. This result tells us that, in some sense, the language **PCF**$^{++}$ is now "complete" in that it can express all computable functions at the types we are considering — no further computable functions can be added which are not already expressible in **PCF**$^{++}$. This striking fact can be thought of as a kind of "Church's Thesis" for functions of higher type.

Fully abstract and universal models in general. Let us return to PCF in its pure "sequential" version, with no funny parallel operators. Can one give a fully abstract and/or universal model for PCF in a reasonable way, perhaps by finding some suitable mathematical property which characterizes the "sequentially computable" functions?

Curiously, this problem turns out to be much harder for pure PCF than for many other apparently more complicated languages that have been studied. Around 1993, some fully abstract and universal models of PCF were indeed constructed using the ideas of *game semantics*, but even these models are in practice not very usable for establishing observational equivalences. In 1994, Ralph Loader proved a surprising negative result, which in some sense says that there are no "really good" models of PCF — this gives some kind of theoretical explanation for the difficulty encountered in constructing such models.

It's worth noting, though, that for PCF and many other languages, one can obtain a fully abstract and universal model in a rather "cheating" way. Define a model M as follows: for each type τ, let M_τ be the set of closed terms $e : \tau$ *modulo* the observational equivalence relation, and define the interpretation of any term e in M to be just the equivalence class of e. It's not too hard to check that this indeed gives us an adequate, compositional model (e.g. for PCF), which is fully abstract and universal virtually by definition. Such models are often called *term models* because they are built using terms of the language itself.

The term model construction is fundamentally useless for *proving* observational equivalences, since it is defined using the notion of observational equivalence. The real interest of denotational semantics lies in trying to give "syntax-independent" constructions of fully abstract/universal models which shed genuinely new light on the languages in question.