PFPL Supplement: Inductive and Coinductive Types in FPC*

Robert Harper

December 2021

1 Introduction

FPC is an extension of PCF with (unrestricted) recursive types, $\rho(t\,.\,\tau)$, that represent solutions of type equations "up to isomorphism" mediated by fold and unfold:

$$\begin{aligned}\texttt{fold}(\ -\ ) &: [\rho(t\,.\,\tau)/t]\tau \to \rho(t\,.\,\tau)\\ \texttt{unfold}(\ -\ ) &: \rho(t\,.\,\tau) \to [\rho(t\,.\,\tau)/t]\tau\end{aligned}$$

Recursive types are powerful. They may be used to define self-referential types, including recursive function types, and are sufficient to encode the untyped λ-calculus. Divergent computations, and therefore partial functions, are therefore unavoidable in FPC: as soon as recursive types are admitted, divergence is too.

It is natural to ask whether, when restricted to polynomial type operators,[1] recursive types include inductive and coinductive types. The answer is closely tied to whether the dynamics of FPC is eager or lazy.[2] Consider, for example, the recursive type $\rho(t\,.\,\mathtt{1}+t)$, deliberately not named to avoid putting the syntax ahead of the semantics. Under an eager dynamics, which distinguishes values from computations, the values of this type are in one-to-one correspondence with the natural numbers; it is therefore equivalent to the inductive type $\mu(t\,.\,\mathtt{1}+t)$. Under a lazy dynamics, however, values and computations are conflated, with the result that the recursive type contains as values all divergent computations of its type, and consequently all values that inject a divergent computation into the right summand. Moreover, it contains all values of the coinductive type $\nu(t\,.\,\mathtt{1}+t)$, including the "infinite stack of right injections" and all values of the inductive type. It is thus very far from being that of the natural numbers, or even the co-natural numbers, $\nu(t\,.\,\mathtt{1}+t)$.

In the lazy case there is no way to avoid some amount of "cruft" in polynomial recursive types. For example, even if the injections of sum types were stipulated to be eager, the range of values in the recursive type is cut down to those of $\nu(t\,.\,\mathtt{1}+t)$, *plus* the divergent computation of that type, which is never avoidable in the lazy setting. Thus, the coinductive type can never be properly interpreted; the best one can do is this augmented form of it. Ironically, the eager interpretation of FPC can come just as close as the lazy using suspensions: the type $\rho(t\,.\,\mathtt{susp}(\mathtt{1}+t))$ does the trick. Thus, the eager interpretation can represent all the types of the lazy interpretation, plus considerably more; it is unequivocally superior in its powers of representation.

[1] From here on all type operators are assumed to be polynomial.

[2] Meaning that all constructors are eager or lazy, according to whether computations are distinguished from values.

2 Inductive Types in Eager FPC

As illustrated in the introduction, under the eager dynamics for **FPC** the recursive type $\rho(\,t\,.\,\tau\,)$ is the inductive type $\mu \triangleq \mu(\,t\,.\,\tau\,)$. To see this it suffices to define the introduction and elimination forms for the inductive type in terms of those of the recursive type, and to check that they behave as expected.

The introductory form for the recursive type, $\mathtt{fold}(\,-\,)$, serves as the introductory form for that type viewed as an inductive type. Note that under the eager interpretation $\mathtt{fold}(\,e\,)$ is a value only if e is a value. The eliminatory form $\mathtt{rec}\{t\,.\,\tau\}(\,e\,;x\,.\,e'\,) : \rho$, may be defined as the application, $R(\,e\,)$, of the recursive function

$$R \triangleq \mathtt{fun}\, r(\,x{:}\mu\,){:}\rho\, \mathtt{is}\, [\mathtt{map}\{t\,.\,\tau\}(\,x\,.\,\mathtt{ap}(\,r\,;x\,)\,)(\,\mathtt{unfold}(\,x\,)\,)/x]e'.$$

When e is a value,

$$R(\,\mathtt{fold}(\,e\,)\,) \longmapsto^* [\mathtt{map}\{t\,.\,\tau\}(\,x\,.\,\mathtt{ap}(\,R\,;x\,)\,)(\,e\,)/x]e',$$

which is essentially equivalent to

$$R(\,\mathtt{fold}(\,e\,)\,) \longmapsto^* [\mathtt{rec}\{t\,.\,\tau\}(\,e\,;x\,.\,e'\,)/x]e',$$

as specified for inductive types. For example, in the case of the type operator $t\,.\,\mathtt{1} + t$, the recursor steps to a case analysis distinguishing zero from successor, and performing the recursive call on the predecessor in the latter case.

3 Coinductive Types in Lazy FPC

As suggested in the introduction under the lazy interpretation the recursive type $\rho(\,t\,.\,\tau\,)$ contains any divergent computation, plus all further elements constructed with it. Indeed, the eliminatory form of the recursive type diverges when applied to a divergent element. The generator is defined by taking $\mathtt{gen}\{t\,.\,\tau\}(\,e\,;x\,.\,e'\,)$ to be the application, $G(\,e\,)$, of the recursive function

$$G \triangleq \mathtt{fun}\, g(\,x{:}\sigma\,){:}\nu\, \mathtt{is}\, \mathtt{fold}(\,\mathtt{map}\{t\,.\,\tau\}(\,x\,.\,g(\,x\,)\,)(\,e'\,)\,).$$

Under lazy evaluation the application $G(\,e\,)$ steps immediately to a value, because $\mathtt{fold}(\,e\,)$ is always a value. And indeed one may check that

$$\mathtt{unfold}(\,G(\,e\,)\,) \longmapsto^* \mathtt{map}\{t\,.\,\tau\}(\,x\,.\,G(\,x\,)\,)(\,[e/x]e'\,),$$

which is morally equivalent to the transition

$$\mathtt{unfold}(\,G(\,e\,)\,) \longmapsto^* \mathtt{map}\{t\,.\,\tau\}(\,x\,.\,\mathtt{gen}\{t\,.\,\tau\}(\,x\,;x\,.\,e'\,)\,)(\,[e/x]e'\,).$$

4 Not Conversely

Does the definition of the generator give a coinductive interpretation in the eager case? No. Consider the recursive type $\rho \triangleq \rho(\,t\,.\,\mathtt{1} + t\,)$. Were it coinductive, the function

$$I \triangleq \lambda\,(\,x : \rho\,)\,\mathtt{gen}\{t\,.\,\mathtt{1} + t\}(\,x\,;y\,.\,\mathtt{unfold}(\,y\,)\,)$$

would be the identity function, which it is not under the by-value dynamics. For example, the application

$$I(\,\mathtt{gen}\{t\,.\,\mathtt{1}+t\}(\,\langle\rangle\,;\,\_\,.\,\mathtt{r}\cdot x\,)\,)$$

diverges, rather than computing to a value of the recursive type.

Does the definition of the recursor give an inductive interpretation the lazy case? No, not even if folding and injection are eager. For example, were the recursive type $\rho \triangleq \rho(\,t\,.\,\mathtt{1}+t\,)$ inductive, then it would also have elements $\bot$, $\mathtt{fold}(\,\bot\,)$, $\mathtt{fold}(\,\mathtt{r}\cdot\bot\,)$, $\mathtt{fold}(\,\mathtt{r}\cdot\mathtt{fold}(\,\mathtt{r}\cdot\bot\,)\,)$, and so on. All of these would be sent to $\bot$ by the function

$$I \triangleq \lambda\,(\,x:\rho\,)\,\mathtt{rec}\{t\,.\,\mathtt{1}+t\}(\,x\,;\,y\,.\,\mathtt{fold}(\,y\,)\,),$$

which would be the identity in the inductive case.

References

Robert Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, Cambridge, England, Second edition, 2016.