A Note on Recursive Types and Fixed Points

Aaron Stump

April 20, 2017

1 Recursive types

A recursive type is one whose definition may reference the type itself. An example is

$$rec\ X.\ 1 + X$$

where 1 is the unit type with just one inhabitant *triv*, and + is a sum type. We will use this example in this note.

There are different ways to set this up, but in my opinion the most desirable typing rules for recursive types allow us to fold and unfold them:

$$\frac{\Gamma \vdash t : [rec\ X.\ T/X]T}{\Gamma \vdash t : rec\ X.\ T} \qquad \frac{\Gamma \vdash t : rec\ X.\ T}{\Gamma \vdash t : [rec\ X.\ T/X]T}$$

The above rules are type-assignment rules: they are not changing the subject term t of typing. For algorithmic typing, one approach is to tell the type-checker where to do a fold or an unfold, by putting an annotation on the term t. But for theoretical study, the type-assignment rules are sufficient (at least for many purposes).

2 Fixed points and recursive types

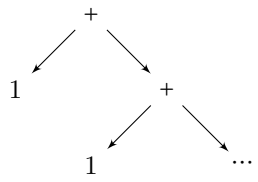
Suppose we are trying to give some kind of semantics for recursive types, with respect to which the above typing rules will be sound. What do we need from such a semantics? Really just one thing: we need to know that the meaning of the recursive type is the same as the meaning of its unfolding. Then (for simplicity assuming Γ is empty) the meaning $t \in [\![rec\ X.\ T]\!]_\rho$ of the conclusion of the rule for folding the definition will be equivalent to the meaning $t \in [\![[rec\ X.\ T/X]T]\!]_\rho$ of the conclusion. (The semantics will need to use a variable assignment ρ to map type variables to semantic types.)

The traditional way to achieve this is to define $[\![[rec\ X.\ T/X]T]\!]$ to be a fixed point of the meta-level function $S \mapsto [\![T]\!]_{\rho[X \mapsto S]}$ (where the variable assignment $\rho[X \mapsto S]$ behaves just like ρ except that it maps X to S). But the question is, which fixed point?

For recursive types are not the same as inductive or coinductive types, which at least in type theory should come with recursors or corecursors. So from the point of view of semantics we are free to take any fixed point we like.

3 Infinite trees

We can think of a recursive type as a finite notation for an infinite type. For our example recursive type, this tree is



4 Least fixed-point

If we define the semantics as the least fixed-point, we can build this up as

$$\bigcup_{n \in \mathbb{N}} (F^n \ \bot)$$

where $\bot$ is the smallest semantic type – it could be $\varnothing$ for example, if semantic types are being interpreted as sets of terms (let us suppose they are); and F is the meta-level function $S \mapsto \mathit{Unit} \uplus S$ arising from interpreting the body of our example recursive type.

With this interpretation, the interpretation of the type will include all the terms of the form (with $n \in \mathbb{N}$)

$$\mathit{inj}_2^n \ (\mathit{inj}_1 \ \mathit{unit})$$

where

$$\begin{array}{lcl} \mathit{inj}_1 & : & \forall X.\forall Y.\ X \to X + Y \\ \mathit{inj}_1 & : & \forall X.\forall Y.\ Y \to X + Y \\ \mathit{unit} & : & \mathit{Unit} \end{array}$$

These values correspond to all the leaves in our infinite unfolding (above) of the recursive type. They can be viewed as an encoding of natural numbers, and the semantics then can be seen as interpreting $\mathit{rec}\ X.\ 1 + X$ as the set of natural numbers.

5 Greatest fixed-point

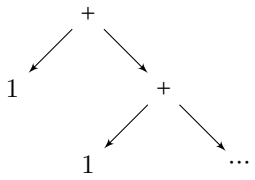
On the other hand, suppose we define the semantics as the greatest fixed-point, which we can construct as

$$\bigcap_{n \in \mathbb{N}} (F^n \ \top)$$

where $\top$ is some greatest semantic type. It might be the set of all closed terms, for example.

What difference will there be with the least fixed-point? By definition, all the elements of the least fixed-point will be in the greatest one (since the least is a subset of all the fixed points). So the question is, are there any extra elements in the meaning of $\mathit{rec}\ X.\ 1 + X$ as greatest fixed points?

Consider again the infinite unfolding of our example type:



We already have values for all the paths to the leaves. Is there a value for the infinite path down the right side of the tree? Such a value would need to be of the form:

$$inj_2\ (inj_2\ (inj_2\ \cdots))$$

We could denote this as inj_2^∞.

Suppose our language has a fixed-point operator *fix* at the term level, satisfying

$$fix\ f = f\ (fix\ f)$$

where let us consider that the notion of equality is equality of infinite unfolding of defining equations. Then we have this equality:

$$inj_2^\infty = fix\ inj_2$$

It is type-correct to apply *fix* to inj_2: *fix* requires a function whose type is of the form $T \to T$ for some type T. inj_2 (implicitly instantiating its quantifiers) has type

$$(rec\ X.\ (1 + X)) \to 1 + (rec\ X.\ (1 + X))$$

and $rec\ X.\ (1 + X)$ unfolds to the codomain type (so inj_2 does have type of the form $T \to T$).

We can identify inj_2^∞ with an infinite number ω. We thus have the coinductive type of co-naturals, using the semantics with the greatest fixed-point.

6 Which fixed point is best?

Thanks to contravariance of function types, if we pick the greatest fixed-point for the interpretation of $rec\ X.\ T$, as being the most liberal, we will end up with smaller interpretations of function types: functions would need to deal not just with the finite terms $inj_2^n\ (inj_1\ unit)$, but also the "infinite" term inj_2^∞. So there is no obvious correct default fixed-point to choose. Indeed, in programming it is not obvious whether inductive types (ML languages) or coinductive types (Haskell) by default is best (though maybe we should give the nod to inductive types if we are thinking of mathematical reasoning, where they are by far more common generally).