Why Not Lambda Encodings?

written by Eashan Hatti on Functor Network
original link: https://functor.network/user/721/entry/473

A short note on lambda encodings for proof assistants. I got this question a few days ago:

> Why don't we use Church encodings or otherwise to represent inductive datatypes instead of these usual method that declares all the formation, introduction, and elimination rules as additional rules added to the theory?

Essentially, the problem is that you can't get the right *induction principles*. Let's take the `Unit` type as an example. In any proof assistant, `Unit` gets the following induction principle:

$$\prod_{P:\text{Unit}\to\text{Type}} \prod_{x:\text{Unit}} P\,\star \to P\,x$$

All is as per usual. Ordinarily with Church encodings the type is defined as the recursion principle, so let's just generalize that to the induction principle. From that the definition seems extremely simple.

$$\text{Unit} := \prod_{P:\text{Unit}\to\text{Type}} \prod_{x:\text{Unit}} P\,\star \to P\,x$$

Now we can clearly see the problem – the type is defined negatively in terms of itself! Remember that with the usual inductive types we see in proof assistants, we have the *positivity restriction*, which means recursive occurances of a type can only appear in positive positions, without which we can easily prove `False`. Thus this "obvious" way of doing lambda encodings in type theory unfortunately isn't sound.

However, there is hope! Aaron Stump has done lots of work on designing a type theory in which lambda encodings can be soundly implemented. I haven't had time to keep up with that line of work, but you can check out this slide deck for an introduction: Slide deck.

Anyway, you might be wondering why strict positivity is necessary, since it's understandably unintuitive for most. I'll write up a note on that sometime soon and link it here.