Reifying reduction

Lucius Gregory Meredith
Michael Stay

Consider the 2-category $\mathrm{Th}(SK)$ with finite products freely generated by the following data:

0. one object T,
1. 1-morphisms
   - $S\colon 1 \to T$
   - $K\colon 1 \to T$
   - $(--)\colon T \times T \to T$, and
2. 2-morphisms
   - $\sigma\colon (((S\,x)\,y)\,z) \Rightarrow ((x\,z)\,(y\,z))$
   - $\kappa\colon ((K\,x)\,y) \Rightarrow x$.

There is a forgetful functor U from the functor 2-category $\mathrm{Model}(SK) = \hom(\mathrm{Th}(SK), \mathrm{Cat})$ to Cat with a left adjoint $F : \mathrm{Cat} \to \mathrm{Model}(SK)$. The free $SK-$calculus on a category adjoins objects S and K and all binary products; it also adjoins morphisms for reducing S and K in head position applied to three and two terms, respectively. The free $SK-$calculus on the empty category $\emptyset$ is a category $F\emptyset$ whose

0. objects are terms of the calculus and whose
1. morphisms are reductions in the calculus

equipped with structural functors and natural transformations.

Let $\mathrm{End}(F\emptyset)$ be the monoidal category of structural endofunctors and structural transformations between them. We get a category C by viewing $\mathrm{End}(F\emptyset)$ as a one-object 2-category and modding out by the 2-morphisms. The category C has

0. one object and
1. a morphism for each equivalence class of terms in the calculus,

where terms t and u are equivalent if there exist some term v and 2-cells $t \Rightarrow v$ and $u \Rightarrow v$. Standard results on abstraction elimination from lambda calculus show that the category C is

cartesian closed, where the one object is the type $X \cong X^X$ of the "untyped" $SK-$calculus, and Scott domains provide models of C in Set.

All that said, lazy languages don't reduce arguments until they are needed, and even eager functional languages like JavaScript, Perl, Python, and Ruby don't reduce under a binder. [[Cite Abramsky and Ong, stuff on deriving the LTS from the term contexts for doing bisimilarity, and that paper on reactive systems.]] If we try to model reductions to weak head normal form as 2-morphisms, however, we can whisker the 2-morphism σ by the 1-morphism $(S\,-)$ and get a reduction of an argument, which violates the reduction strategy. While this is not much of a problem for a convergent calculus like the $SK-$calculus, it completely destroys the semantics of calculi like $\pi-$calculus, where the binders denote *waiting* for a message on a channel.

We propose to model reduction contexts explicitly with a 1-morphism R, adding 2-morphisms to describe the propagation of the context, and modifying the existing reduction 2-morphisms to refer to R. In the example of the $SK-$calculus with lazy reduction, we have

- a 1-morphism $R\colon T \to T$,
- a 2-isomorphism $\rho\colon R(x\,y) \xRightarrow{\sim} (Rx\,y)$,
- a 2-morphism $\sigma'\colon R(((S\,x)\,y)\,z) \Rightarrow R((x\,z)\,(y\,z))$, and
- a 2-morphism $\kappa'\colon R((K\,x)\,y) \Rightarrow Rx$.

The morphism R behaves something like a catalyst, enabling a term to reduce. The 2-morphism ρ acts linearly on R, preserving the amount of catalyst available while moving it from one part of the term to another up and down the left side of the tree of applications.

[[What happens if we mod out by rho, sigma', and kappa'? Do we still get a cartesian closed category? How does it differ from the one above? Can we prove something about full abstraction or something else to show that it really does capture the intended semantics?]]

References