CS 6110 Lecture 3 Equivalence, Reductions and Normal Forms 30 January 2009

Lecturer: Andrew Myers

1 Term Equivalence

When are two terms equal? This is not as simple a question as it may seem. As *intensional* objects, two terms are equal if they are syntactically identical. As *extensional* objects, however, two terms should be equal if they represent the same function. We will say that two terms are *equivalent* if they are equal in an extensional sense.

For example, it seems clear that the terms $\lambda x.\, x$ and $\lambda y.\, y$ are equivalent. The name of the variable is not essential. But we also probably think that $\lambda x.\, (\lambda y.\, y)\ x$ is equivalent to $\lambda x.\, x$ too, in a less trivial sense. And there are even more interesting cases, like $\lambda x.\, \lambda y.\, x\ y$.

But what function does a term like $\lambda x.\, x$ represent? Intuitively, it's the identity function, but over what domain and codomain? We might think of it as representing the set of all identity functions, but this interpretation quickly leads to Russell's paradox. In fact, defining a precise mathematical model for lambda-calculus terms is far from straightforward, requiring some sophisticated domain theory.

One possible meaning of a term is divergence. There are infinitely many divergent terms; one example is Ω. In some sense, all divergent terms are equivalent, since none of them produce a value. The implication is that it is undecidable to determine whether two terms are equivalent, because otherwise, given the relationship between the λ-calculus and Turing machines, we could solve the halting problem on lambda calculus terms by testing equivalence to Ω.

1.1 Observational equivalence

Another way of approaching the problem is to say that two terms are equivalent if they behave indistinguishably in all possible contexts.

More precisely, two terms will be considered equal if in every context, either

- they both converge and produce the same value, or
- they both diverge.

A *context* is just a term $C[[\,\cdot\,]]$ with a single occurrence of a distinguished special variable, called the *hole*. The notation $C[e]$ denotes the context $C[[\,\cdot\,]]$ with the hole replaced by the term e. Then we then define equality in the following way:

$$e_1 = e_2 \quad \Longleftrightarrow \quad \text{for all contexts } C[[\,\cdot\,]],\ C[e_1] \Downarrow v \text{ iff } C[e_2] \Downarrow v.$$

Without loss of generality, we can simplify the definition to

$$e_1 = e_2 \quad \Longleftrightarrow \quad \text{for all contexts } C[[\,\cdot\,]],\ C[e_1] \Downarrow \text{ iff } C[e_2] \Downarrow,$$

because if they converge to different values, it is possible to devise a context that causes one to converge and the other to diverge. Suppose that $C[e_1] \Downarrow v_1$ and $C[e_2] \Downarrow v_2$, where v_1 and v_2 have different behavior. Then we can find some context $C'[[\,\cdot\,]]$, which applied to v_1 converges, and applied to v_2 diverges. Therefore, the context $C'[C[\cdot]]$ is a context that causes the original e_1, e_2 to converge and diverge respectively, satisfying the simpler definition.

A conservative approximation (but unfortunately still undecidable) is the following. Let e_1 and e_2 be terms, and suppose that e_1 and e_2 converge to the same value when reductions are applied according to some strategy. Then e_1 is equivalent to e_2. This *normalization* approach (in which terms are reduced to a a *normal form* on which no more reductions can be done) is useful for compiler optimization and for checking type equality in some advanced type systems.