

CS256: Programming Languages and Semantics

Denotational Semantics and Equational Theory for STLC

<http://www.eecs.harvard.edu/~greg/cs256sp2006/>

Greg Morrisett
greg@eecs.harvard.edu
327 Maxwell Dworkin

March 1, 2006

1 Equational Theory

Last time, we talked about reduction, normal forms, and strong normalization. Today, we're going to directly define ways to prove that two terms are equivalent to one another (in terms of input/output relationships.) We'll begin by writing down a proof system for establishing that two expressions are equivalent. You'll note that the reduction rules are just particular orientations of the last few rules. That is, the reduction rules should be easily justified as sound if we can establish the soundness of this equational theory.

$$\begin{aligned}(\text{refl}) \quad & \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e \equiv e : \tau} \\(\text{symm}) \quad & \frac{\Gamma \vdash e_1 \equiv e_2 : \tau}{\Gamma \vdash e_2 \equiv e_1 : \tau} \\(\text{tran}) \quad & \frac{\Gamma \vdash e_1 \equiv e : \tau \quad \Gamma \vdash e \equiv e_2 : \tau}{\Gamma \vdash e_1 \equiv e_2 : \tau} \\(\beta) \quad & \frac{\Gamma \vdash (\lambda x:\tau_2.e_1) e_2 : \tau_2 \rightarrow \tau_1}{(\lambda x:\tau.e_1) e_2 \equiv e_1[e_2/x] : \tau_2} \\(\eta) \quad & \frac{\Gamma \vdash e : \tau_2 \rightarrow \tau}{\Gamma \vdash \lambda x:\tau.(e x) \equiv e} \\(\text{app}) \quad & \frac{\Gamma \vdash e_1 \equiv e'_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 \equiv e'_2 : \tau_2}{\Gamma \vdash e_1 e_2 \equiv e'_1 e'_2 : \tau} \\(\text{fn}) \quad & \frac{\Gamma, x:\tau_1 \vdash e_1 \equiv e_2 : \tau_2}{\Gamma \vdash \lambda x:\tau_1.e_1 \equiv \lambda x:\tau_1.e_2 : \tau_1 \rightarrow \tau_2}\end{aligned}$$

How can we establish soundness of this equational theory? At first blush, it might seem sufficient to prove that whenever we can derive $\Gamma \vdash e_1 \equiv e_2 : \tau$, then for all substitutions γ such that $\vdash \gamma(x) : \Gamma(x)$, we can prove $\gamma(e_1) \Downarrow v$ iff $\gamma(e_2) \Downarrow v$. But in general, we only get values that are equivalent, not syntactically equal. The trick is to reduce all of our reasoning to base types, since then the only equivalent values are in fact syntactically equal.

So again, we might define some type-indexed sets, this time of pairs of equivalent expressions:

$$\begin{aligned} V[[b]] &= \{(c, c)\} \\ V[[\tau_1 \rightarrow \tau_2]] &= \{(v_1, v_2) \mid \vdash v_i : \tau_i \wedge \forall (e_1, e_2) \in C[[\tau_1]]. (v_1 e_1, v_2 e_2) \in C[[\tau_2]]\} \\ C[[\tau]] &= \{(e_1, e_2) \mid \vdash e_i : \tau \wedge e_i \Downarrow v_i \wedge (v_1, v_2) \in V[[\tau]]\} \end{aligned}$$

Here, I've stratified the definition into sets corresponding to equivalent *values* (V) and sets corresponding to equivalent *computations* (C). Two (closed computations) are equivalent if they reduce to equivalent values. Two values are equivalent at base type if they are the same constant. Two functions are equivalent at $\tau_1 \rightarrow \tau_2$ if whenever we apply them to equivalent computations, they produce equivalent computations.

We can extend the definition to contexts as follows:

$$C[[\Gamma]] = \{(\gamma_1, \gamma_2) \mid \forall x \in \text{Dom}(\Gamma). (\gamma(x), \gamma_2(x)) \in C[[\Gamma(x)]]\}$$

Using these definitions, we can formulate an appropriate induction hypothesis: if we can derive $\Gamma \vdash e_1 \equiv e_2 : \tau$, then for all $(\gamma_1, \gamma_2) \in C[[\Gamma]]$, $(\gamma_1(e_1), \gamma_2(e_2)) \in C[[\Gamma]]$.

2 Denotational Semantics

I'm going to define a denotational semantics for the call-by-value variant of the simply-typed lambda calculus. (Call-by-name is essentially the same since they both are strongly normalizing.) The goal here is to give a natural, set-theoretic semantics which turns out to be pretty easy to do. We'll begin by providing mappings from types to sets. To be concrete, I'll assume that our only base type is "int" and our only constants are integer values.

$$\begin{aligned} \mathcal{T}[\text{int}] &= Z \\ \mathcal{T}[\tau_1 \rightarrow \tau_2] &= \mathcal{T}[\tau_2]^{\mathcal{T}[\tau_1]} \\ &= \mathcal{T}[\tau_1] \rightarrow \mathcal{T}[\tau_2] \end{aligned}$$

In other words, we're interpreting the type "int" as the set of all integers, and the type " $\tau_1 \rightarrow \tau_2$ " as the type of (set-theoretic) functions from the set denoted by τ_1 to the set denoted by τ_2 . We'll define our universe as:

$$\mathcal{U} = \bigcup_{\tau} \mathcal{T}[\tau]$$

and extend the interpretation to contexts, Γ as follows:

$$\begin{aligned} \mathcal{T}[\bullet] &= \{(x, 0)\} \\ \mathcal{T}[\Gamma, x:\tau] &= \{\gamma[x \mapsto v] \mid \gamma \in \mathcal{T}[\Gamma] \wedge v \in \mathcal{T}[\tau]\} \end{aligned}$$

Thus, $\mathcal{T}[\Gamma] : \text{Id} \rightarrow \mathcal{U}$, or more precisely:

$$\mathcal{T} : \Pi \Gamma : \text{Id} \rightarrow \text{Type}. \Pi x : \text{Id}. \mathcal{T}[\Gamma(x)]$$

We now will define a mapping \mathcal{E} from typing judgments $\Gamma \vdash e : \tau$ to set theoretic functions with the property that:

$$\mathcal{E}[\Gamma \vdash e : \tau] : \mathcal{T}[\Gamma] \rightarrow \mathcal{T}[\tau]$$

The definition is given by induction on typing derivations:

$$\begin{aligned} \mathcal{E}[\Gamma \vdash i : \text{int}] \gamma &= i \\ \mathcal{E}[\Gamma \vdash x : \Gamma(x)] \gamma &= \gamma(x) \\ \mathcal{E}[\Gamma \vdash e_1 e_2 : \tau] \gamma &= (\mathcal{E}[\Gamma \vdash e_1 : \tau_2 \rightarrow \tau] \gamma) (\mathcal{E}[\Gamma \vdash e_2 : \tau_2] \gamma) \\ \mathcal{E}[\Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2] \gamma &= f \text{ where } f(v) = \mathcal{E}[\Gamma, x:\tau_1 \vdash e : \tau_2] (\gamma[x \mapsto v]) \end{aligned}$$

It's relatively easy to see that indeed, \mathcal{E} maps contexts and terms to the set-theoretic values of the appropriate (mathematical) type.

We now want to establish the *adequacy* of the translation with respect to our equational theory. That is, we want to show that $\Gamma \vdash e_1 \equiv e_2 : \tau$ iff $\mathcal{E}[\Gamma \vdash e_1 : \tau] = \mathcal{E}[\Gamma \vdash e_2 : \tau]$. Note that since evaluation is a sub-set of reduction, which in turn is a sub-set of the equivalence relation, this implies that the denotational model respects the operational semantics. That is, as a corollary of the above, if $\Gamma \vdash e : \tau$ then $e \Downarrow v$ iff $\mathcal{E}[\Gamma \vdash e : \tau] = \mathcal{E}[\Gamma \vdash v : \tau]$.

Establishing adequacy allows us to use the denotational semantics to directly argue that two expressions are equivalent in the realm of set theory. In some sense, \mathcal{E} compiles out all of the computation allowing us to reason completely syntactically (but at the level of sets).

For instance, consider the two expressions f and $\lambda x.f x$ which are related by the η equivalence rule. We know that intuitively, these are equivalent in the sense that, given the same environments and arguments, they will produce the same results. We can show this using the model by picking an arbitrary input environment γ and argument v , applying the translation of each term to γ and v , and see if we get the same result.

$$\begin{aligned} \mathcal{E}[f:\tau_1 \rightarrow \tau_2 \vdash f : \tau_1 \rightarrow \tau_2] \gamma v &= (\gamma f) v \\ \mathcal{E}[f:\tau_1 \rightarrow \tau_2 \vdash \lambda x:\tau_1.(f x) : \tau_1 \rightarrow \tau_2] \gamma v &= (\mathcal{E}[f:\tau_1 \rightarrow \tau_2, x:\tau_1 \vdash f x : \tau_2] (\gamma[x \mapsto v])) \\ &= (\gamma[x \mapsto v] f) (\gamma[x \mapsto v] v) \\ &= (\gamma f) v \end{aligned}$$

Since the two expressions produce equal (set-theoretic) values, given equal inputs, we can conclude that they denote the same thing. Generalizing this argument so that it handles all of the reduction rules is not too hard.