

CS256: Programming Languages and Semantics

Reduction, Logical Relations and Strong Normalization

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

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

February 24, 2006

1 Reduction and Normal Forms

Last time, I said that every program in the simply-typed lambda calculus terminates. In fact, we can say something a bit stronger: Given two well-typed expressions e_1 and e_2 , there is a decision procedure to determine whether e_1 and e_2 produce “equivalent” values as results. For base types, “equivalent” means the same constant (syntactically). For function types, “equivalent” means that given equivalent inputs, the functions compute equivalent outputs. Note that such an equivalence captures a semantic notion of function equivalence, and thus is better than one which only equates, say, α -equivalent functions or some other relatively syntactic notion of equality.

What is the process for deciding whether or not e_1 and e_2 are equivalent? The trick is to define a notion of *reduction* or *normalization* which transforms the expressions in a meaning-preserving fashion until it is syntactically apparent whether or not they are equal. If each expression has a unique normal form, if we can compute those normal forms, and if syntactically comparing normal forms is complete (in an appropriate sense), then we’ll have a decision procedure for determining the computational equivalence of two expressions.

So let us begin by constructing our notion of reduction. Reduction generalizes evaluation in that we’ll be doing rewriting steps *under* λ -abstractions. That is, we’ll be symbolically evaluating some expressions (not knowing what the actual parameter to a function is.) The reduction relation, $e_1 \longrightarrow e_2$ is defined as follows:

$$\begin{aligned} (\beta) \quad & (\lambda x:\tau.e_1) e_2 \longrightarrow e_1[e_2/x] \\ (\eta) \quad & \lambda x:\tau.(e x) \longrightarrow e \quad (x \notin FV(e)) \\ & \frac{e_1 \longrightarrow e'_1}{e_1 e_2 \longrightarrow e'_1 e_2} \\ & \frac{e_2 \longrightarrow e'_2}{e_1 e_2 \longrightarrow e_1 e'_2} \\ & \frac{e \longrightarrow e'}{\lambda x:\tau.e \longrightarrow \lambda x:\tau.e'} \end{aligned}$$

The first rule corresponds to our β -rule for CBN evaluation. The second rule says that if e is a function, then $\lambda x.(e x)$ is an equivalent function (assuming x is not a free variable of e .) It’s easy to see that if we pass an argument e_2 to the eta-expanded version (i.e., $(\lambda x.(e x)) e_2$), then it evaluates to $e e_2$ so the two forms will compute the same value. The remainder of the rules tell us that in we can substitute a reduced form of an expression anywhere we want.

Note that the rules do not force a particular reduction order on terms: For instance, given $(\lambda f.f)(\lambda x.ex)$, I could choose to use the third rule, coupled with β to reduce to $(\lambda x.ex)$ and then use the η rule to reduce to e . Alternatively, I could start by using the fourth rule, coupled with η to reduce to $(\lambda f.f)e$ and then use β to reduce to e . In either case, we get the same result. Of course, it's not clear that (a) any sequence of reductions will necessarily terminate, (b) any two sequences will produce the same terminal configuration. That is, we need to establish that unique normal forms exist for this notion of reduction. Furthermore, we need to show that two expressions have the same behavior iff they have the same normal forms.

So our work is cut out for us...

2 Confluence

A rewriting system or relation is *confluent* when for any two reduction sequences $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$, there exists a configuration e' such that $e_1 \longrightarrow^* e'$ and $e_2 \longrightarrow^* e'$. That is, confluence tells us that when there's a fork in the road, it doesn't matter because we can always rejoin the other fork later down the path.

Confluence is a great property to have in a rewriting system because it means that we don't have to think globally: we can just react locally to whatever seems good to us. That's not to say that one path might not be more efficient than the other, rather that we can't get stuck choosing the wrong fork in the road. Confluence also guarantees us that if two reduction sequences, starting from the same configuration, reach terminal configurations, then those terminal configurations must be the same. It's easy to see this because if $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$ and both e_1 and e_2 are terminal, then the only way to satisfy confluence is if $e_1 = e_2$. So confluence tells us that if reduction is terminating, then we have unique normal forms.

It turns out that reduction for both the untyped lambda calculus and the simply-typed lambda calculus enjoy confluence. However, proving confluence in the untyped setting is much harder because some reductions might not terminate (e.g., $(\lambda x.xx)(\lambda x.xx)$ reduces via β to itself.)

When every sequence for a reduction relation must terminate (i.e., eventually reach a terminal configuration), we say that it is *terminating*. When a reduction relation is both terminating and confluent, then it is said to be *strongly normalizing*. So a strongly normalizing reduction relation gives us a guarantee of unique normal forms, regardless of the reduction strategy we use.

Actually, it is sufficient to show that a terminating relation is *locally confluent* to establish strong normalization. Local confluence says that when $e \longrightarrow e_1$ and $e \longrightarrow e_2$, then there exists an e' such that $e_1 \longrightarrow^* e'$ and $e_2 \longrightarrow^* e'$.

So how do we prove that local confluence and termination guarantee confluence? Let us define the bound of an expression e as the length of the longest reduction sequence that starts with e . Since the relation is terminating, we know that all expressions have finite bounds. The proof of confluence, assuming local confluence, then proceeds by induction on b , showing the result for each expression with bound b .

For the base case, pick any expression e with bound 0. Suppose $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$. Since e 's bound is 0, it must be that $e = e_1$ and $e = e_2$. Therefore, choosing $e' = e$, we have that $e_1 \longrightarrow^* e'$ and $e_2 \longrightarrow^* e'$. So all expressions with bound 0 are confluent.

Now assume that for all expressions of bound b , local confluence implies confluence. Pick an expression e with bound $b+1$ and suppose that $e \longrightarrow^{b_1} e_1$ and $e \longrightarrow^{b_2} e_2$. Note that if b_1 is 0, then $e = e_1$ and we can pick $e' = e_2$ and show that $e_1 \longrightarrow^* e'$ and $e_2 \longrightarrow^* e'$. Similarly, if b_2 is 0, then confluence holds trivially. So assume that both b_1 and b_2 are greater than 0.

It follows that there exists an e'_1 such that $e \longrightarrow e'_1 \longrightarrow^{b_1-1} e_1$ and an e'_2 such that $e \longrightarrow e'_2 \longrightarrow^{b_2-1} e_2$.

By local confluence, it follows that there exists an e_3 such that $e'_1 \longrightarrow^* e_3$ and $e'_2 \longrightarrow^* e_3$. To summarize, at this point we have:

$$\begin{array}{ccccc}
 & & e'_1 & \longrightarrow^{b_1-1} & e_1 \\
 & \nearrow & & \searrow^* & \\
 e & & & & e_3 \\
 & \searrow & & \nearrow^* & \\
 & & e'_2 & \longrightarrow^{b_2-1} & e_2
 \end{array}$$

Furthermore, e'_1 and e'_2 must have bounds at most b since otherwise, our original expression e would have a reduction sequence longer than $b + 1$. Consequently, our induction hypothesis applies to e'_1 and e'_2 , meaning that we can find e''_1 and e''_2 such that $e_1 \longrightarrow^* e''_1$, $e_3 \longrightarrow^* e''_1$ and $e_2 \longrightarrow^* e''_2$, $e_3 \longrightarrow^* e''_2$:

$$\begin{array}{ccccc}
 & & e'_1 & \xrightarrow{b_1-1} & e_1 & \xrightarrow{*} & e''_1 \\
 & \nearrow & \searrow^* & & \nearrow^* & & \\
 e & & & & e_3 & \searrow^* & \\
 & \searrow & \nearrow^* & & \searrow^* & & \\
 & & e'_2 & \xrightarrow{b_2-1} & e_2 & \xrightarrow{*} & e''_2
 \end{array}$$

Now since e_3 must also have a bound less than b , our induction hypothesis tells us that e_3 is confluent. Therefore, we know there exists an e' such that $e''_1 \longrightarrow^* e'$ and $e''_2 \longrightarrow^* e'$ completing the diagram:

$$\begin{array}{ccccccc}
 & & e'_1 & \xrightarrow{b_1-1} & e_1 & \xrightarrow{*} & e''_1 & & \\
 & \nearrow & \searrow^* & & \nearrow^* & & \searrow^* & & \\
 e & & & & e_3 & & & & e' \\
 & \searrow & \nearrow^* & & \searrow^* & & \nearrow^* & & \\
 & & e'_2 & \xrightarrow{b_2-1} & e_2 & \xrightarrow{*} & e''_2 & &
 \end{array}$$

So we've just showed that any two paths starting from e must eventually rejoin at some e' .

3 Local Confluence

We've reduced our confluence problem to two sub-tasks: First, showing that reduction is *locally* confluent, and second, showing that reduction is terminating (i.e., strongly normalizing.) To establish local confluence, we need to show that if a term has two possible reductions, then we can further reduce the resulting terms to the same term.

This proceeds by a rather tedious case analysis on terms. In essence, you need to look at each possible term and look at each pair of possible reductions on that form of term. As an example, consider the case where the term is $e_1 e_2$. We might be able to reduce e_1 , and we might be able to reduce e_2 . Doing one doesn't interfere with the other so we don't really need to worry about reductions that don't overlap.

On the other hand, consider $(\lambda x.e_1) e_2$. Here, we can maybe do a β reduction to get $e_1[e_2/x]$ or we might be able to reduce e_2 to e'_2 . In the latter case, we can obviously do the β -reduction, but in the former case, we need to show that we can apply a bunch of reductions to all of the copies of e_2 that got substituted for x . That is, we need a lemma that shows that (transitive closure of) reduction commutes with substitution.

I'll leave the rest of the analysis for you...

4 Termination

The hardest and most challenging aspect of this line is showing that reduction terminates. Instead of proving this, I'm going to prove a weaker theorem, that *evaluation* always terminates, and then ask you to generalize it to reduction.

The theorem I'm going to prove is "If $\bullet \vdash e : \tau$, then there exists a v such that $e \Downarrow v$ ".

I'll begin by defining a family of sets of terms, indexed by type:

$$\begin{aligned}
 T(b) &= \{e \mid \bullet \vdash e : b \wedge \exists v.e \Downarrow v\} \\
 T(\tau_1 \rightarrow \tau_2) &= \{e \mid \bullet \vdash e : \tau_1 \rightarrow \tau_2 \wedge \exists v.e \Downarrow v \wedge \forall e' \in T(\tau_1).(e e') \in T(\tau_2)\}
 \end{aligned}$$

Notice that the definition of the sets is well-founded, because the type always decreases. Notice also that if we can show that an expression e is in $T(\tau)$, then evaluation of e must terminate. So the property we're looking for is baked into the definition of the sets. Finally, in the case of function types, the sets require

that we are closed under application. That is, it's not enough to show that the expression e terminates, but also that when it is applied to terminating arguments, it results in a terminating term (at the result type.)

What remains is to show that $\bullet \vdash e : \tau$ implies that $e \in T(\tau)$. This looks like it might be a straightforward induction on the height of the derivation that tells us e is well-typed. But alas, we'll get to a λ -expression sooner or later, in which case, the context will become non-empty.

Therefore, we need to strengthen our theorem to cover terms with free variables. To do so, we lift T to map contexts to substitutions. A substitution is a finite map from variables to closed terms:

$$(\text{substitution}) \quad \gamma : \text{Var} \xrightarrow{\text{fin}} \text{Term}$$

We define $\hat{\gamma}(e)$ to apply the substitution to a term e as follows:

$$\begin{aligned} \hat{\gamma}(c) &= c \\ \hat{\gamma}(x) &= \gamma(x) && (x \in \text{Dom}(\gamma)) \\ \hat{\gamma}(x) &= x && (x \notin \text{Dom}(\gamma)) \\ \hat{\gamma}(\lambda x:\tau.e) &= \lambda x:\tau.(\hat{\gamma}(e)) \\ \hat{\gamma}(e_1 e_2) &= (\hat{\gamma}(e_1) \hat{\gamma}(e_2)) \end{aligned}$$

We define $T(\Gamma)$ as the set of all possible substitutions that map variables in Γ to terms in the appropriate T -set:

$$T(\Gamma) = \{\gamma \mid \forall x \in \text{Dom}(\Gamma). \gamma(x) \in T(\Gamma(x))\}$$

That is, γ is in $T(\Gamma)$ if whenever $x:\tau$ is in Γ , then $\gamma(x)$ yields a term in $T(\tau)$.

Now we can finally state the (strengthened) induction hypothesis:

Lemma: If $\Gamma \vdash e : \tau$ then for all $\gamma \in T(\Gamma)$, $\hat{\gamma}(e) \in T(\tau)$.

Notice that as a corollary, $\bullet \vdash e : \tau$ implies $e \in T(\tau)$ which implies that any well-typed term terminates.

We will establish the lemma by induction on the derivation D that allows us to conclude $\Gamma \vdash e : \tau$. For reference, here are the typing rules again: The typing rules are as follows:

$$\begin{aligned} (\text{const}) \quad & \Gamma \vdash c : b \\ (\text{var}) \quad & \Gamma \vdash x : \Gamma(x) \\ (\rightarrow I) \quad & \frac{\Gamma, x:\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x:\tau_1.e : \tau_1 \rightarrow \tau_2} \quad (x \notin \text{Dom}(\Gamma)) \\ (\rightarrow E) \quad & \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \end{aligned}$$

- The first base case is when \mathcal{D} ends with an instance of the const rule. In this case, we have $\Gamma \vdash c : b$ and must show for all $\gamma \in T(\Gamma)$ that $\hat{\gamma}(c) \in T(b)$. From the definition $\hat{\gamma}(c) = c$, so this reduces to showing $c \in T(b)$. Since $\bullet \vdash c : b$ and $c \downarrow c$, the result follows trivially.
- The second base case is when \mathcal{D} ends with an instance of the var rule. In this case, we have $\Gamma \vdash x : \Gamma(x)$. Pick $\gamma \in T(\Gamma)$. We must show $\hat{\gamma}(x) = \gamma(x) \in T(\Gamma(x))$. But this follows from the fact that $\gamma \in T(\Gamma)$.
- Suppose \mathcal{D} ends with an instance of the $\rightarrow E$ rule with $e = e_1 e_2$. This means that we have derivations \mathcal{D}_1 and \mathcal{D}_2 showing $\Gamma \vdash e_1 : \tau_2 \rightarrow \tau$ and $\Gamma \vdash e_2 : \tau_2$ respectively.

Pick a $\gamma \in T(\Gamma)$. We must show $\hat{\gamma}(e_1 e_2) = (\hat{\gamma}(e_1)) (\hat{\gamma}(e_2)) \in T(\tau)$. Applying our induction hypothesis to \mathcal{D}_1 , we know that $\hat{\gamma}(e_1) \in T(\tau_2 \rightarrow \tau)$. From the definition of $T(\tau_2 \rightarrow \tau)$, it follows that there exists a v_1 such that $e_1 \downarrow v_1$ and for all $e' \in T(\tau_2)$, $\hat{\gamma}(e_1) e' \in T(\tau)$.

Applying our induction hypothesis to \mathcal{D}_2 , we know that $\hat{\gamma}(e_2) \in T(\tau_2)$. Therefore, $\hat{\gamma}(e_1 e_2) = \hat{\gamma}(e_1) \hat{\gamma}(e_2) \in T(\tau)$.

- Suppose \mathcal{D} ends with an instance of the $\rightarrow I$ rule with $e = \lambda x:\tau_1.e'$ and $\tau = \tau_1 \rightarrow \tau_2$. Then we have a derivation \mathcal{D}' deriving $\Gamma, x:\tau \vdash e' : \tau_2$. Pick $\gamma \in T(\Gamma)$. We must show $\hat{\gamma}(\lambda x:\tau_1.e') = \lambda x:\tau_1.\hat{\gamma}(e') \in T(\tau_1 \rightarrow \tau_2)$.

First, note that $\lambda x:\tau_1.\hat{\gamma}(e') \Downarrow \lambda x:\tau_1.\hat{\gamma}(e')$, so the termination of e is ensured. Now we must show that when applied to a terminating argument, the result is terminating. That is, we must show for all $e_1 \in T(\tau_1)$, $e e_1 \in T(\tau_2)$.

So pick an $e_1 \in T(\tau_1)$. Note that $\gamma[x \mapsto e_1] \in T(\Gamma, x:\tau_1)$. Thus, applying our induction hypothesis to \mathcal{D}' , it follows that $\gamma[x \hat{\mapsto} e_1](e') \in T(\tau_2)$. The result then follows from the definition of evaluation for applications.

This establishes that evaluation is terminating. Your job is to figure out how to extend this sort of argument to reduction. You'll need to define type-indexed sets of terms $SN(\tau)$ that have the strong-normalization property with respect to reduction and then show that every well-typed term of type τ lies within $SN(\tau)$. The trick is defining the right closure conditions on $SN(\tau_1 \rightarrow \tau_2)$ to ensure that the argument goes through.

5 Homework

Prove that if $\Gamma \vdash e : \tau$, then every reduction sequence starting from e terminates.