CS611 Lecture 31 Logical relations: strong normalization November 12, 2001

Scribe: Andrew Chung, Bill McCloskey Lecturer: Andrew Myers

1 Strong Normalization

Recall from before that $\lambda^{\rightarrow}$ is a typed version of the lambda calculus with lambda terms of the form

$$\lambda x\!:\!\tau.\, e$$

A language has the property of strong normalization if every expression can be normalized, or evaluated to a value. The language $\lambda^{\rightarrow}$ has this property. Such languages are not as expressive as languages without strong normalization, since the halting problem is solvable. But they are useful for more restricted applications where termination is desirable, such as modelling the properties of other languages.

There are two ways to show that a language exhibits strong normalization. The first is to show that the operational semantics agrees with the denotational semantics. The other is to show directly that every expression can be normalized.

2 Agreement between SOS and Denotational Semantics

Usually, to show the two semantics agree, it is sufficient to show that $e\Uparrow \iff \mathcal{C}[\![\vdash e\!:\!B]\!]\rho = \bot$. Any other difference between the SOS and denotational semantics can be transformed into expressions that diverge in only one of the semantics.

The argument for equivalence can be formalized in two statements.

- Soundness: $e \longmapsto^* v \;\Rightarrow\; \mathcal{C}[\![\vdash e : \tau]\!]\rho = \mathcal{C}[\![\vdash v : \tau]\!]\rho$
- Adequacy: $e \longmapsto^* b \;\Leftarrow\; \mathcal{C}[\![\vdash e : B]\!]\rho = \mathcal{C}[\![\vdash b : B]\!]\rho$

$b\!:\!B$ is a base value. Notice that the meanings are actually equal, and not merely equivalent, as has been the case before. The first condition states that if e evaluates to a value in the operational semantics, then the meaning of e is equal to the meaning of the value. The second condition only needs to be proved for base values. If there is a difference in an expression that has a non-base type, base values can be extracted from the term, the equivalence can be proved, and the based values can be wrapped back into the expression.

3 Proof of Strong Normalization

3.1 Logical Relations

However, a more direct statement will be proved in this lecture. It will be shown that $\vdash e : \tau \;\Rightarrow\; \exists v.\, e \longmapsto^* v$. This statement will be proved using logical relations. A logical relation is a relation defined on terms and indexed by types. $e\ R_\tau\ e'$ is a relation on expressions e and e' of type τ. Logical relations can be used to prove properties of terms using induction over typing derivations.

In this case, a unary relation R_τ will be used. R_τ essentially behaves like a set, so that $R_\tau\ e \iff e \in R_\tau$. R_τ will be defined as the set of terms of type τ which terminate and which preserve termination behavior. Formally,

$$\begin{aligned} e \in R_B &\iff \vdash e : B \;\wedge\; \exists v.\, e \longmapsto^* v \\ e \in R_{\tau\rightarrow\tau'} &\iff \vdash e : \tau \rightarrow \tau' \;\wedge\; \exists v.\, e \longmapsto^* v \;\wedge\; \forall e' \in R_\tau.\, (e\ e') \in R_{\tau'} \end{aligned}$$

B is any of the three base types: $\mathsf{1}$, bool, or int. Note that although e may have type B, it is not necessarily a base value, syntactically. For example, the term $(\lambda x : \mathsf{1}.\, 2)\ \#\mathsf{u} \in R_{\mathsf{int}}$, but it is not a base value. Also, notice the last clause of the second condition. It states that when the function is applied to a term that terminates, the result of the application must terminate. This clause captures the essence of the logical relations approach.

3.2 A Substitution Operator

Now the goal is to prove $\vdash e : \tau \Rightarrow e \in R_\tau$. If this were true, it would imply $\exists v.\, e \longmapsto^* v$ by the definition of R_τ. Unfortunately, it is not possible to directly prove this goal. The hypothesis is too weak, since it is only valid in an empty typing context. But to prove the statement in a typing context Γ, a special substitution operator will be needed to ensure that the term remains in R_τ even after substitution.

The substitution operator $\gamma \in \mathbf{Var} \rightharpoonup \mathbf{Expr}$ will be defined. In some sense, γ must agree with the typing context Γ. Any substitution made for a variable x must have type $\Gamma(x)$ and must be in $R_{\Gamma(x)}$. Define $\gamma \models \Gamma$ to be true when the domains of γ and Γ are equal and when $\forall x \in \mathrm{dom}(\gamma).\, \gamma(x) \in R_{\Gamma(x)}$. Then define an operator $\hat{\gamma} \in \mathbf{Expr} \to \mathbf{Expr}$ that applies the substition γ on an entire expression. Formally this operator $\hat{\cdot}$ is defined on an arbitrary γ as follows, by structural induction:

$$
\begin{array}{rcl}
\hat{\gamma}(x) & = & \begin{cases} \gamma(x) & \text{if } x \in \mathrm{dom}(\gamma) \\ x & \text{otherwise} \end{cases} \\
\hat{\gamma}(b) & = & b \\
\hat{\gamma}(e_0\ e_1) & = & \hat{\gamma}(e_0)\ \hat{\gamma}(e_1) \\
\hat{\gamma}(\lambda x\!:\!\tau.\, e) & = & \lambda x\!:\!\tau.\, \hat{\gamma}_x(e) \\
\gamma_x & = & \begin{cases} \gamma & \text{if } x \notin \mathrm{dom}(\gamma) \\ \gamma - \langle x, \gamma(x) \rangle & \text{otherwise} \end{cases}
\end{array}
$$

Now the new goal, using the substitution operator, is $\Gamma \vdash e : \tau \ \wedge\ \gamma \models \Gamma \ \Rightarrow\ \hat{\gamma}(e) \in R_\tau$. This goal is a generalization of the previous goal, with $\gamma = \Gamma = \emptyset$. The proof of this statement will use a substitution lemma $\Gamma \vdash e : \tau \ \wedge\ \gamma \models \Gamma \ \Rightarrow \vdash \hat{\gamma}(e) : \tau$. The proof of this lemma is similar to the proof of other substitution lemmas, and will be omitted. The substitution lemma is needed to ensure that $\vdash \hat{\gamma}(e) : \tau$ (note the lack of a typing context), which is required by the definition of R_τ.

Another lemma is also used in the proof below. It states that $\Gamma \vdash e : \tau \wedge e \longmapsto^* e' \wedge e' \in R_\tau \Rightarrow e \in R_\tau$. If e steps to e', and e' terminates, then e also terminates, and types are preserved across the step. The proof of this lemma is left as an exercise.

3.3 Proof of Strong Normalization

Recall the syntax of $\lambda^{\to}$. It will be useful during the proof.

$$e ::= b \ \mid\ x \ \mid\ e_0\ e_1 \ \mid\ \lambda x\!:\!\tau.\, e$$

Case 1. Assume $e = b$, a base value. Clearly $\vdash e : B$. It is necessary to show that $\Gamma \vdash b : B \ \wedge\ \gamma \models \Gamma \Rightarrow \hat{\gamma}(b) \in R_B$. Since b is a base value, it is a value, so $b \longmapsto^* v = b$. Therefore $b = \hat{\gamma}(b) \in R_B$.

Case 2. Assume $e = x$, a variable. It is necessary to show that $\Gamma \vdash x : \tau \ \wedge\ \gamma \models \Gamma \ \Rightarrow\ \hat{\gamma}(x) \in R_\tau$. Now since x is a variable, $\tau = \Gamma(x)$, and so $x \in \mathrm{dom}(\Gamma)$. Therefore, because $\gamma \models \Gamma$, $\gamma(x) \in R_\tau$ and $\mathrm{dom}(\gamma) = \mathrm{dom}(\Gamma)$. And finally, since $\hat{\gamma}(x) = \gamma(x)$, $\hat{\gamma}(x) \in R_\tau$.

Case 3. Assume $e = e_0\ e_1$. It is necessary to show that $\Gamma \vdash e_0\ e_1 : \tau \ \wedge\ \gamma \models \Gamma \ \Rightarrow\ \hat{\gamma}(e_0\ e_1) \in R_\tau$. A typing derivation exists for $\Gamma \vdash e_0\ e_1 : \tau$, and it must have the form

$$\frac{\Gamma \vdash e_0 : \tau' \to \tau \qquad \Gamma \vdash e_1 : \tau'}{\Gamma \vdash e_0\ e_1 : \tau}$$

Therefore, there are derivations for the two typing judgments in the antecedent of the inference rule. So, by the induction hypothesis, $\hat{\gamma}(e_0) \in R_{\tau' \to \tau}$ and $\hat{\gamma}(e_1) \in R_{\tau'}$. By the definition of $R_{\tau' \to \tau}$, $\hat{\gamma}(e_0\ e_1) = \hat{\gamma}(e_0)\ \hat{\gamma}(e_1) \in R_\tau$, since $\hat{\gamma}(e_1) \in R_{\tau'}$.

Case 4. Assume $e = \lambda x\!:\!\tau.\, e'$. It is necessary to show that

$$\Gamma \vdash (\lambda x\!:\!\tau.\, e') : \tau \to \tau' \ \wedge\ \gamma \models \Gamma \ \Rightarrow\ \hat{\gamma}(\lambda x\!:\!\tau.\, e') \in R_{\tau \to \tau'}$$

By the definition of $\hat{\gamma}$, $\hat{\gamma}(\lambda x\!:\!\tau.\, e') = \lambda x\!:\!\tau.\, \hat{\gamma}_x(e')$, which is a value. Using the assumptions, the substitution lemma states, $\vdash \hat{\gamma}(\lambda x : \tau.\, e') : \tau \to \tau'$. Now it's necessary to show the extra third clause, which was used in case 3. Let e'' be any element of R_τ. The clause requires that $\hat{\gamma}(e)\ e'' = (\lambda x : \tau.\, \hat{\gamma}_x(e'))\ e'' \in R_{\tau'}$. By the evaluation rules (call-by-name) of $\lambda^{\rightarrow}$, $(\lambda x : \tau.\, \hat{\gamma}_x(e'))\ e'' \longmapsto \hat{\gamma}_x(e')\{e''/x\}$. The second lemma mentioned above can be used here. It states that if $\hat{\gamma}_x(e')\{e''/x\} \in R_{\tau'}$, then $(\lambda x\!:\!\tau.\, \hat{\gamma}_x(e'))\ e'' \in R_{\tau'}$. Define $\gamma'_x = \gamma_x[x \to e''] = \gamma[x \to e'']$ (since γ_x was just γ on all values except x). Now $\hat{\gamma}_x(e')\{e''/x\} = \hat{\gamma}'_x(e')$, so it's only necessary to show that $\hat{\gamma}'_x(e') \in R_{\tau'}$.

By assumption, $\Gamma \vdash (\lambda x\!:\!\tau.\, e') : \tau \to \tau'$. The derivation of this judgment must have the form

$$\frac{\Gamma, x\!:\!\tau \vdash e' : \tau'}{\Gamma \vdash (\lambda x\!:\!\tau.\, e') : \tau \to \tau'}$$

and so there must be a derivation for $\Gamma, x\!:\!\tau \vdash e' : \tau'$. Now observe that $\gamma'_x \models \Gamma, x\!:\!\tau$. This is because $\gamma \models \Gamma$ and $\gamma'_x(x) = e' \in R_\tau$ by assumption. Therefore, by the induction hypothesis, $\hat{\gamma}'_x(e') \in R_{\tau'}$, which completes the proof.

Notes: If the evaluation rules are call-by-value, this proof will still work with minor modifications. Since $e'' \in R_{\tau'}$, it should step to a value v'' by the definition of $R_{\tau'}$. The rest of the proof is similar as before.