CMSC 336: Type Systems for Programming Languages

Lecture 12: Equational Reasoning, Parametricity

Acar & Ahmed **26 February 2008**

Contents

1 Parametricity

Consider the encoding of booleans in System F.

$$\texttt{bool} \equiv \forall\alpha.\alpha \ \rightarrow \ \alpha \ \rightarrow \ \alpha$$

Now we can start to write the terms for members of type `bool` by simply looking at the types. Expression of type `bool` will either be equal to `true` of `false`, which can be written as follows.

$$\texttt{true} \equiv \Lambda\alpha.\lambda x : \alpha.\forall y : \alpha.x$$
$$\texttt{false} \equiv \Lambda\alpha.\lambda x : \alpha.\forall y : \alpha.y$$

Of course there are many functions of type `bool`, but they are all observationally equivalent to either `true` or to `false` (i.e., a function can take the two arguments spend some cycles doing something else, but at the end it will have to return either the first argument x of the function—which means it's equivalent to `true`—or return the second argument y—which means it's equivalent to `false`).

2 Representation Independence

Representation independence says that the behavior of the client does not depend on the particular representation of an existential type.

Example: Set ADT

$$\exists\alpha.\ \{new : \alpha, insert : \alpha \ \rightarrow \ \tau \ \rightarrow \ \alpha, remove : \alpha \ \rightarrow \ \tau \ \rightarrow \ \alpha,$$
$$isEmpty : \alpha \ \rightarrow \ \texttt{bool}, member : \alpha \ \rightarrow \ \tau \ \rightarrow \ \texttt{bool}\}$$

Here you can pick whatever appropriate concrete type you want for α; for example, you could use a list or a balanced tree as the concrete representation of a set.

3 Equivalence

How do we know that two expressions are equal? We can say that two expressions are equal if they are syntactically equal. But, this is somewhat unsatisfactory. For example, are the following equal?

$$e_1 \equiv \lambda x : \texttt{nat}.\lambda y : \texttt{nat}.x + y e_2 \equiv \lambda x : \texttt{nat}.\lambda y : \texttt{nat}.y + x$$

Syntactically these two are not equal but behaviorally they are. So how can we formalize this notion of behavioral equivalence.

We define a *complete program* as a closed expression.

Let's consider Goedel's T. There programs are closed expressions of type `nat`. Let's define the *observable behavior* as whether the result is zero `zero` or a non-zero term `succ` $-$.

Define an *expression context* as an expression with a single hole in it. Examples: `succ` $[\cdot], \lambda x : \texttt{nat}.[\cdot], [\cdot]\ e_1, \lambda x : nat.\lambda y : nat.[\cdot]$. In the final example, the variables x and y are called *exposed*. If we later substitute expressions for holes and if these expressions have the exposed variables as free variables, then these free variables will be captured. In this sense, plugging an expression into a hole is different than conventional substitution which avoids capture.

Formally we can define program contexts as follows. The idea is that we have a single hole for any position that an expression may appear.

$$\begin{array}{lll} \text{Program Contexts} & C & ::= \\ & & \\ & & \\ & & \\ & & \end{array} \begin{array}{l} \texttt{succ}\ C \mid \lambda x.C \mid C_1\ e_2 \mid e_1\ C_2 \mid \\ \texttt{rec}\ C\ \{\ \texttt{zero} \Rightarrow e_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow e_1\} \\ \texttt{rec}\ e\ \{\ \texttt{zero} \Rightarrow C_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow e_1\} \mid \\ \texttt{rec}\ e\ \{\ \texttt{zero} \Rightarrow e_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow C_1\} \end{array}$$

Typing a program context.

$$\vdash C : (\Gamma \rhd \tau) \rightsquigarrow (\Gamma' \rhd \tau')$$

The above judgment indicates that if $\Gamma \vdash e : \tau$ then $\Gamma' \vdash C[e] : \tau'$.

Equivalently, we will write the above judgment as: $\Gamma' \vdash C : (\Gamma \rhd \tau) \rightsquigarrow \tau'$.

The typing rules for contexts are as follows:

$$\frac{}{\Gamma \vdash [\cdot] : (\Gamma \rhd \tau) \rightsquigarrow \tau}$$

$$\frac{\Gamma' \vdash C : (\Gamma \rhd \tau) \rightsquigarrow \texttt{nat}}{\Gamma' \vdash \texttt{succ}\ C : (\Gamma \rhd \tau) \rightsquigarrow \texttt{nat}}$$

$$\frac{\Gamma', x : \tau_1 \vdash C : (\Gamma \rhd \tau) \rightsquigarrow \tau_2}{\Gamma' \vdash \lambda x : \tau_1 . C : (\Gamma \rhd \tau) \rightsquigarrow \tau_1 \ \rightarrow \ \tau_2}$$

$$\frac{\Gamma' \vdash C_1 : (\Gamma \rhd \tau) \rightsquigarrow \tau_1 \ \rightarrow \ \tau_2 \qquad \Gamma' \vdash e_2 : \tau_1}{\Gamma' \vdash C_1\ e_2 : (\Gamma \rhd \tau) \rightsquigarrow \tau_2}$$

$$\frac{\Gamma' \vdash e_1 : \tau_1 \ \rightarrow \ \tau_2 \qquad \Gamma' \vdash C_2 : (\Gamma \rhd \tau) \rightsquigarrow \tau_1}{\Gamma' \vdash e_1\ C_2 : (\Gamma \rhd \tau) \rightsquigarrow \tau_2}$$

$$\frac{\Gamma' \vdash C : (\Gamma \rhd \tau) \rightsquigarrow \texttt{nat} \qquad \Gamma' \vdash e_0 : \tau' \qquad \Gamma', x : \texttt{nat}, y : \tau' \vdash e_1 : \tau'}{\Gamma' \vdash \texttt{rec}\ C\ \{\ \texttt{zero} \Rightarrow e_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow e_1\} : (\Gamma \rhd \tau) \rightsquigarrow \tau'}$$

$$\frac{\Gamma' \vdash e : \texttt{nat} \quad \Gamma' \vdash C_0 : (\Gamma \rhd \tau) \rightsquigarrow \tau' \quad \Gamma', x : \texttt{nat}, y : \tau' \vdash e_1 : \tau'}{\Gamma' \vdash \texttt{rec}\ e\ \{\ \texttt{zero} \Rightarrow C_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow e_1\} : (\Gamma \rhd \tau) \rightsquigarrow \tau'}$$

$$\frac{\Gamma' \vdash e : \texttt{nat} \quad \Gamma' \vdash e_0 : \tau' \quad \Gamma', x : \texttt{nat}, y : \tau' \vdash C_1 : (\Gamma \rhd \tau) \rightsquigarrow \tau'}{\Gamma' \vdash \texttt{rec}\ e\ \{\ \texttt{zero} \Rightarrow e_0 \mid \texttt{succ}\ x\ \texttt{with}\ y\ \Rightarrow C_1\} : (\Gamma \rhd \tau) \rightsquigarrow \tau'}$$

Lemma 1
Suppose $\vdash C : (\Gamma \rhd \tau) \rightsquigarrow (\Gamma' \rhd \tau')$. *We have*

1. $\Gamma \vdash e : \tau$ *then* $\Gamma' \vdash C[e] : \tau'$
2. $\Gamma' \subseteq \Gamma$.

We can now define observational equivalence, a.k.a., contextual equivalence.

Definition 2 (Observational Equivalence)
Suppose $\Gamma \vdash e_1 : \tau$ *and* $\Gamma \vdash e_2 : \tau$. *Then observational equivalence of* e_1 *and* e_2 *is defined as follows:*

$\Gamma \vdash e_1 \sim^{obs} e_2 : \tau \stackrel{def}{=} \forall C. \vdash C : (\Gamma \rhd \tau) \rightsquigarrow \texttt{nat} \Rightarrow C[e_1] \rightarrow^* z$ *if and only if* $C[e_2] \rightarrow^* z$.

Lemma 3 (Congruence)
If $\Gamma \vdash e_1 \sim^{obs} e_2 : \tau$ *and* $C : (\Gamma \rhd \tau) \rightsquigarrow (\Gamma' \rhd \tau')$ *then* $\Gamma' \vdash C[e_1] \sim^{obs} C[e_2] : \tau'$.

4 Logical Equivalence

Direct proofs of observational equivalence of two expressions are nontrivial. The problem is the quantification over all contexts C in the definition of $\sim^{obs}$. Thus, to prove observational equivalence, we develop a proof method based on *logical equivalence*. We show that logical equivalence matches observational equivalence. Thus, to show that two expressions are observationally equivalent, it suffices to show that they are logically equivalent. (Our definition of logical equivalence below is an instance of a *logical relation*.

Logical equivalence, $e \sim e' : \tau$, is a type-indexed family of releations on closed expressions e, e' of type τ.

Definition 4 (Logical equivalence, closed terms $e \sim e' : \tau$)

- $e \sim s' : \texttt{nat}$ *if and only if*
  $e \to^* z$ *and* $e' \to^* z$ *or* $e \to^* \texttt{succ}\, e_1$ *and* $e' \to^* \texttt{succ}\, e'_1$ *and* $e_1 \sim e'_1 : \texttt{nat}$
- $e \sim e' : \tau_1 \to \tau_2$ *if and only if*
  $\forall e_1, e_1.\ e_1 \sim e'_1 : \tau_1 \Rightarrow e\ e_1 \sim e'\ e'_1 : \tau_2$.

Here's an alternative definition of logical equivalence that separates value equivalence from expression equivalene:

Definition 5 (Logical equivalence $v \approx v' : \tau$ and $e \sim e' : \tau$)

- $v \approx v' : \texttt{nat}$ *if and only if*
  $v = v' = \texttt{zero}$ *or* $v = \texttt{succ}\ v_1 \wedge v' = \texttt{succ}\ v'_1 \wedge v_1 \approx v'_1 : \texttt{nat}$.
- $\lambda x : \tau_1.e \approx \lambda x : \tau_1.e' : \tau_1 \to \tau_2$ *if and only if*
  $\forall v_1, v'_1.v_1 \approx v'_1 : \tau_1 \Rightarrow e[v_1/x] \sim e'[v'_1/x] : \tau_2$

Thus far we have defined logical equivalence for closed terms. We generalize these for open terms.

Definition 6 (Logical equivalence, substitutions $\gamma \approx \gamma' : \Gamma$)
Two substitutions γ and γ' (which are mappings from variables to closed values) are logically equivalent at Γ:

$$\gamma \approx \gamma' : \Gamma \stackrel{def}{=} \begin{array}{l} \mathrm{dom}(\gamma) = \mathrm{dom}(\gamma') = \mathrm{dom}(\Gamma) \ \wedge \\ \quad \forall x \in \mathrm{dom}(\Gamma).\ \gamma(x) \approx \gamma'(x) : \Gamma(x) \end{array}$$

Definition 7 (Logical equivalence, open terms)
Suppose $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$. Then

$$\Gamma \vdash e \sim e' : \tau \stackrel{def}{=} \forall \gamma, \gamma'.\gamma \approx \gamma' : \Gamma \Rightarrow \gamma(e) \sim \gamma(e') : \tau$$