CMSC 336: Type Systems for Programming Languages

Lecture 13,14: Parametricity & Representation Independence

Acar & Ahmed **28 Feb and 4 Mar 2008**

Contents

1 Gödel's T: Reasoning about Equivalence **1**
1.1 Fundamental Property 2
1.2 Soundness and Completeness 3

2 Parametricity: Logical Relation for System F **3**
2.1 Reynolds' Abstraction Theorem / Parametricity 6
2.2 Soundness wrt Observational Equivalence 8
2.3 Completeness wrt Contextual Equivalence 8

3 Theorems for Free **9**

4 Representation Independence **10**
4.1 Encoding Existential Types using Universal Types 10

1 Gödel's T: Reasoning about Equivalence

Last time we gave a formal definition of what it means for two programs (or program fragments) to be equivalent. We formalized the notion of program contexts, gave typing rules for contexts, and then defined observational equivalence (aka contextual equivalence) for Gödel's T. Informally, we say that two programs are observationally equivalent if there exists no closed program context C that can distinguish between them. More formally, we defined observational equivalence for Gödel's T as follows:

Definition 1 (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:*

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

Proving that two programs are observationally equivalence is important for a variety of reasons, for instance, for proving the correctness of compiler optimizations and other program transformations, and for establishing that two implementations (packages) of the same interface (with the same existential type) are equivalent.

Reasoning about observational equivalence is also important for reasoning about secure information flow. For instance, we may want to establish that a

program is "secure" in that it does not leak high-security inputs (data) to an observer with a low-security clearance. To formally establish such a property, one would have to show that the program applied to different high-security inputs produces observationally equivalent low-security outputs.

Direct proofs of observational equivalence are generally not feasible due to the quantification over all possible contexts C in the definition of $\sim^{obs}$. Instead to prove observational equivalence we devise a proof method based on *logical relations* and show that the proof method is sound and complete with respect to contextual equivalence.

Last time we defined *logical equivalence* (i.e., a logical relation for equivalence of programs) for Gödel's T as follows:

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

Definition 2 (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* $v = \texttt{succ}\ v_1 \wedge v' = \texttt{succ}\ v_1' \wedge v_1 \approx v_1' : \texttt{nat}$.
- $\lambda x : \tau_1.e \approx \lambda x : \tau_1.e' : \tau_1 \ \rightarrow \ \tau_2$ *if and only if*
  $\forall v_1, v_1'.v_1 \approx v_1' : \tau_1 \ \Rightarrow \ e[v_1/x] \sim e'[v_1'/x] : \tau_2$

We generalize logical equivalence to open terms as follows:

Definition 3 (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:

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

Definition 4 (Logical equivalence, open terms)
Suppose $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$. Then

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

1.1 Fundamental Property

Once we have defined the logical relation, we must show that it is consistent, which we do by proving that if e is a well-typed term, then e is related to itself. That is, we must show that the relation we have defined is reflexive. This is known as the *Fundamental Property* of a logical relation, or alternatively, as the *Basic Lemma*.

Theorem 5 (Fundamental Property)
If $\Gamma \vdash e : \tau$ then $\Gamma \vdash e \sim e : \tau$.

Aside: We have proved above that the logical relation is reflexive. We can now show that the logical relation we have defined is symmetric; the proof of symmetry follows easily from the definition of the logical relation. A *direct proof* of transitivity of the logical relation, however, is much harder. Thus, transitivity is not generally proved directly. It is easier to instead show (see below) that the logical relation is sound and complete with respect to contextual equivalence, and then, since contextual equivalence can easily be shown to be transitive, it follows that the logical relation we have defined is also transitive.

1.2 Soundness and Completeness

Having proved the fundamental property, we can now show that the logical relation is sound with respect to contextual equivalence:

Theorem 6 (Soundness wrt Observational Equivalence)
If $\Gamma \vdash e_1 \sim e_2 : \tau$ *then* $\Gamma \vdash e_1 \sim^{obs} e_2 : \tau$.

Informally, soundness wrt observational equivalence implies that the logical equivalence relation is a subset of observational equivalence. On the other hand, completeness wrt observational equivalence (which we prove next) implies that the observational equivalence relation is a subset of logical equivalence.

Theorem 7 (Completeness wrt Observational Equivalence)
If $\Gamma \vdash e_1 \sim^{obs} e_2 : \tau$ *then* $\Gamma \vdash e_1 \sim e_2 : \tau$.

2 Parametricity: Logical Relation for System F

In System F, suppose you have a polymorphic function f of type $\forall\alpha.\,\alpha \to \alpha$. What function could it be? When instantiated at a type τ, it should evaluate to a function g of type $\tau \to \tau$ that, when further applied to a value v of type τ returns a value v' of type τ. But since f is polymorphic, it cannot depend on τ, which means that g cannot depend on v, so the only value g can return is v. That is, g has to be the identity function at type τ, which means that f must be the polymorphic identity function.

The fact that any expression e of type $\forall\alpha.\,\alpha \to \alpha$ must be equivalent to the polymorphic identity function is a *free theorem* that we can establish as a consequence of *parametricity*. Informally, parametricity can be paraphrased as: if you run a program with related inputs, you will get related outputs. To formally define parametricity, we first have to define what it means for two values (or two expressions) to be *related*. We formalize this by means of a logical relation.

Let us now extend the logical relation for Gödel's T to System F. The syntax of types in System F is as follows:

$$\tau ::= \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\tau$$

As with the logical relation for Gödel's T, we must define when two values v and v' are related at each type τ.

When are two values v and v' related at the type α?

$$v \approx v' : \alpha \text{ iff } ???$$

It is not clear when we should consider two values to be related at the type α.

One possibility is that we don't need to define when two values are related at a type τ that has free type variables—that is we don't need to define when v and v' are related at the type α.

But then let us look at how we would define when two values $\Lambda\alpha.e$ and $\Lambda\alpha.e'$ are related at the type $\forall\alpha.\tau$. Recall that we defined two functions $\lambda x.e$ and $\lambda x.e'$ to be related at the type $\tau_1 \to \tau_2$ if, when given values v_1 and v_1' related at the argument type τ_1, the result of applying the functions to these arguments are two computations $e[v_1/x]$ and $e'[v_1'/x]$ related at the result type τ_2. We follow a similar idea at the type $\forall\alpha.\tau$. We wish to pick two arbitrary types σ and σ' (which we wish to consider related), apply the type abstractions to these two types, and show that we get related computations. For instance, we may try:

$$\Lambda\alpha.e \approx \Lambda\alpha.e' : \forall\alpha.\tau \text{ iff } \forall\sigma,\sigma'.\ e[\sigma/\alpha] \sim e'[\sigma'/\alpha] : \tau$$

Unfortunately the above definition is wrong. The problem is that $e[\sigma/\alpha]$ has type $\tau[\sigma/\alpha]$ while $e[\sigma'/\alpha]$ has type $\tau[\sigma'/\alpha]$, so it is incorrect to require that these computations be related at type τ. But it is just at incorrect to require that they be related at the type $\tau[\sigma/\alpha]$ or at the type $\tau[\sigma'/\alpha]$ since these are different types.

The solution is to define the logical relation so that it is parametrized with a relational type substitution η that maps type variables α to pairs of *closed* types (σ,σ'). Informally, η remembers that we substituted σ for α in the term on the left and σ' for α in the term on the right. Thus, we may write:

$$\Lambda\alpha.e \approx \Lambda\alpha.e' : \forall\alpha.\tau \mid \eta \text{ iff } \forall\sigma,\sigma'.\ e[\sigma/\alpha] \sim e'[\sigma'/\alpha] : \tau \mid \eta[\alpha \mapsto (\sigma,\sigma')]$$

Thus, when we write $v \approx v' : \tau \mid \eta$, it must be the case that η provides a mapping for all free type variables in τ—i.e., $FTV(\tau) \subset \mathrm{dom}(\eta)$.

However, this still isn't enough to define:

$$v \approx v' : \alpha \mid \eta \text{ iff } ???$$

We can now say that we know that if $\eta(\alpha) = (\sigma,\sigma')$, then v must have type σ and v' must have type σ'. However, we still cannot say whether v and v' should be considered related.

We need an extra ingredient: the substitution η should tell us not only what types σ and σ' were substituted for α on each side, but also should give us a relation R on values of type σ and σ'.

More formally, we define a *parametrized logical relation*. Let R be a relation on *closed* values—that is, if $(v, v') \in R$ then $\mathrm{FV}(v) = \mathrm{FV}(v') = \mathrm{FTV}(v) = \mathrm{FTV}(v') = \emptyset$. Let σ and σ' be closed types. Then:

$$R : \sigma \leftrightarrow \sigma' \;\text{ iff }\; \forall (v, v') \in R.\ \vdash v : \sigma \ \wedge\ \vdash v' : \sigma'$$

Now we can define $v \approx v' : \tau \mid \eta$ as follows:

$$\begin{array}{lll}
v \approx v' : \alpha \mid \eta & \text{iff} & \eta(\alpha) = (\tau, \tau', R) \ \wedge\ (v, v') \in R \\
\Lambda\alpha.\, e \approx \Lambda\alpha.\, e' : \forall\alpha.\, \tau \mid \eta & \text{iff} & \forall \sigma, \sigma', R.\ R : \sigma \leftrightarrow \sigma' \ \Rightarrow \\
 & & \quad e[\sigma/\alpha] \sim e'[\sigma'/\alpha] : \tau \mid \eta[\alpha \mapsto (\sigma, \sigma', R)] \\
\lambda x{:}\eta^1(\tau_1).\, e \approx \lambda x{:}\eta^2(\tau_1).\, e' : \tau_1 \to \tau_2 \mid \eta & \text{iff} & \forall v, v'.\ v \approx v' : \tau_1 \mid \eta \ \Rightarrow \\
 & & \quad e[v/x] \sim e'[v'/x] : \tau_2 \mid \eta
\end{array}$$

Next, we define $e \sim e' : \tau \mid \eta$ (which says that two closed expressions e and e' are related at the type τ with respect to relation assignment η) as follows:

$$e \sim e' : \tau \mid \eta \quad \text{iff} \quad \exists v, v'.\ e \to^* v \ \wedge\ e' \to^* v' \ \wedge\ v \approx v' : \tau \mid \eta$$

Thus e is related to e' at type τ if they each evaluate to values v and v' and if these values are related at the type τ. (Recall that all expressions in System F terminate. If we add nontermination to the language, say by adding *fix*, then the above definition would be incorrect.)

Aside: Nontermination In a language with nontermination, we would define $e \sim e' : \tau \mid \eta$ as follows:

$$\begin{array}{lll}
e \sim e' : \tau \mid \eta & \text{iff} & (\forall v.\ e \to^* v \ \Rightarrow\ \exists v'.\ e' \to^* v' \ \wedge\ v \approx v' : \tau \mid \eta) \ \wedge \\
 & & (\forall v'.\ e' \to^* v' \ \Rightarrow\ \exists v.\ e \to^* v \ \wedge\ v \approx v' : \tau \mid \eta)
\end{array}$$

We generalize the above logical relation to open expressions as follows:

A value assignment γ is a finite map from term variables x to closed values v (i.e., values with no free type or term variables).

If $\eta = \{\alpha_1 \mapsto (\sigma_1, \sigma_1', R_1), \ldots, \alpha_n \mapsto (\sigma_n, \sigma_n', R_n)\}$ then, we use the following abbreviations:

$$\begin{array}{rcl}
\eta^1(e) & = & e[\sigma_1/\alpha_1, \ldots, \sigma_n/\alpha_n] \\
\eta^2(e) & = & e[\sigma_1'/\alpha_1, \ldots, \sigma_n'/\alpha_n]
\end{array}$$

We define $\eta \models \Delta$, which says that a relation assignment η satisfies a context Δ, as follows:

$$\begin{array}{lll}
\eta \models \Delta & \text{iff} & \forall \alpha \in \Delta.\ \alpha \in \mathrm{dom}(()\eta) \ \wedge \\
 & & \quad (\eta(\alpha) = (\sigma, \sigma', R) \ \Rightarrow\ R : \sigma \leftrightarrow \sigma')
\end{array}$$

We write $\gamma \approx \gamma' : \Gamma \mid \eta$ to indicate that the value assignments γ and γ' are related at Γ relative to η:

$$\begin{array}{lll}
\gamma \approx \gamma' : \Gamma \mid \eta & \text{iff} & \mathrm{dom}(()\gamma) = \mathrm{dom}(()\gamma') = \mathrm{dom}(()\Gamma) \ \wedge \\
 & & \forall x \in \mathrm{dom}(()\Gamma).\ \gamma(x) \approx \gamma'(x) : \Gamma(x) \mid \eta
\end{array}$$

Definition 8 (Parametrized Logical Relation)
Suppose $\Delta;\Gamma \vdash e : \tau$ *and* $\Delta;\Gamma \vdash e' : \tau$. *Then*

$$\Delta;\Gamma \vdash e \sim e :' \tau \quad \text{iff} \quad \forall \eta, \gamma, \gamma'.\ \eta \models \Delta \ \wedge\ \gamma \approx \gamma' : \Gamma \mid \eta \ \Rightarrow \\ \gamma(\eta^1(e)) \sim \gamma'(\eta^2(e')) : \tau \mid \eta$$

2.1 Reynolds' Abstraction Theorem / Parametricity

Next we prove the Fundamental Property of the logical relation. This theorem is also known as Reynolds' Abstraction theorem, or as the Parametricity theorem.

Theorem 9
If $\Delta;\Gamma \vdash e : \tau$ *then* $\Delta;\Gamma \vdash e \sim e : \tau$.

The proof is by induction on the derivation $\Delta;\Gamma \vdash e : \tau$

We present here, the proof cases for lambda abstraction and application.

- **Case:** $\dfrac{\Delta;\Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta;\Gamma \vdash \lambda x{:}\tau_1.\, e : \tau_1 \to \tau_2}$ (Abs)

  Proof

  We are required to show that $\Delta;\Gamma \vdash \lambda x{:}\tau_1.\, e \sim \lambda x{:}\tau_1.\, e : \tau_1 \to \tau_2$.
  Consider arbitrary η, γ, and γ' such that
  - $\eta \models \Delta$, and
  - $\gamma \approx \gamma' : \Gamma \mid \eta$.

  We are required to show that $\gamma(\eta^1(\lambda x{:}\tau_1.\, e)) \sim \gamma'(\eta^2(\lambda x{:}\tau_1.\, e)) : \tau_1 \to \tau_2 \mid \eta$
  $\equiv \lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e)) \sim \lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e)) : \tau_1 \to \tau_2 \mid \eta$.
  Take $v = \lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e))$.
  Take $v' = \lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e))$.
  We are required to show that
  - $\lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e)) \to^* \lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e))$,
    which is immediate since $\lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e))$ is a value,
  - $\lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e)) \to^* \lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e))$,
    which is immediate since $\lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e))$ is a value, and
  - $\lambda x{:}\eta^1(\tau_1).\, \gamma(\eta^1(e)) \approx \lambda x{:}\eta^2(\tau_1).\, \gamma'(\eta^2(e)) : \tau_1 \to \tau_2 \mid \eta$,
    which we conclude as follows:

    Consider arbitrary v_1 and v_1' such that
    - $v_1 \approx v_1' : \tau_1 \mid \eta$.

    We are required to show $\gamma(\eta^1(e))[v_1/x] \sim \gamma'(\eta^2(e))[v_1'/x] : \tau_2 \mid \eta$.
    Applying the induction hypothesis to $\Delta;\Gamma, x : \tau_1 \vdash e : \tau_2$, we have that $\Delta;\Gamma, x : \tau_1 \vdash e \sim e : \tau_2$.
    Instantiate the latter with η, $\gamma[x \mapsto v_1]$, and $\gamma'[x \mapsto v_1']$. Note that

* $\eta \models \Delta$,
  which follows from above, and
* $\gamma[x \mapsto v_1] \approx \gamma'[x \mapsto v_1'] : \Gamma, x : \tau_1 \mid \eta$,
  which follows from
  · $\gamma \approx \gamma' : \Gamma \mid \eta$ (follows from above), and
  · $v_1 \approx v_1' : \tau_1 \mid \eta$ (follows from above).

Hence, $\gamma[x \mapsto v_1](\eta^1(e)) \sim \gamma'[x \mapsto v_1'](\eta^2(e)) : \tau_2 \mid \eta$.
Thus, $\gamma(\eta^1(e))[v_1/x] \sim \gamma'(\eta^2(e))[v_1'/x] : \tau_2 \mid \eta$, as we were required to show.

■

- **Case:** $\dfrac{\Delta; \Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Delta; \Gamma \vdash e_2 : \tau_1}{\Delta; \Gamma \vdash e_1\ e_2 : \tau_2}$ (App)

Proof

We are required to show that $\Delta; \Gamma \vdash e_1\ e_2 \sim e_1\ e_2 : \tau_2$.
Consider arbitrary η, γ, and γ' such that

– $\eta \models \Delta$, and
– $\gamma \approx \gamma' : \Gamma \mid \eta$.

We are required to show that $\gamma(\eta^1(e_1\ e_2)) \sim \gamma'(\eta^2(e_1\ e_2)) : \tau_2 \mid \eta$
$\equiv \gamma(\eta^1(e_1))\ \gamma(\eta^1(e_2)) \sim \gamma'(\eta^2(e_1))\ \gamma'(\eta^2(e_2)) : \tau_2 \mid \eta$.
Applying the induction hypothesis to $\Delta; \Gamma \vdash e_1 : \tau_1 \to \tau_2$, we have that $\Delta; \Gamma \vdash e_1 \sim e_1 : \tau_1 \to \tau_2$.
Instantiate the latter with η, γ, and γ'. Note that

– $\eta \models \Delta$, and
– $\gamma \approx \gamma' : \Gamma \mid \eta$.

Hence, $\gamma(\eta^1(e_1)) \sim \gamma'(\eta^2(e_1)) : \tau_1 \to \tau_2 \mid \eta$.
From the latter, there exist v_1 and v_1' such that

– $\gamma(\eta^1(e_1)) \to^* v_1$,
– $\gamma'(\eta^2(e_1)) \to^* v_1'$, and
– $v_1 \approx v_1' : \tau_1 \to \tau_2 \mid \eta$.

From the latter it follows that $v_1 = \lambda x{:}\eta^1(\tau_1).\, e_{11}$ and $v_1' = \lambda x{:}\eta^2(\tau_1).\, e_{11}'$.

Applying the induction hypothesis to $\Delta; \Gamma \vdash e_2 : \tau_1$, we have that $\Delta; \Gamma \vdash e_2 \sim e_2 : \tau_1$.
Instantiate the latter with η, γ, and γ'. Note that

– $\eta \models \Delta$, and
– $\gamma \approx \gamma' : \Gamma \mid \eta$.

Hence, $\gamma(\eta^1(e_2)) \sim \gamma'(\eta^2(e_2)) : \tau_1 \mid \eta$.
From the latter, there exist v_2 and v_2' such that

– $\gamma(\eta^1(e_2)) \to^* v_2$,
– $\gamma'(\eta^2(e_2)) \to^* v_2'$, and
– $v_2 \approx v_2' : \tau_1 \mid \eta$.

Instantiate $\lambda x{:}\eta^1(\tau_1).\, e_{11} \approx \lambda x{:}\eta^2(\tau_1).\, e_{11}' : \tau_1 \to \tau_2 \mid \eta$ (from above) with v_2 and v_2'. Note that

– $v_2 \approx v_2' : \tau_1 \mid \eta$.

Hence, $e_{11}[v_2/x] \sim e_{11}'[v_1'/x] : \tau_2 \mid \eta$.
From the latter, there exist v and v' such that

– $e_{11}[v_2/x] \to^* v$,
– $e_{11}'[v_2'/x] \to^* v'$, and
– $v \approx v' : \tau_2 \mid \eta$.

Thus, there exist v and v' such that

– $\gamma(\eta^1(e_1))\ \gamma(\eta^1(e_2)) \to^* v$ (from above by operational semantics),
– $\gamma'(\eta^2(e_1))\ \gamma'(\eta^2(e_2)) \to^* v'$ (from above by operational semantics), and
– $v \approx v' : \tau_2 \mid \eta$, which follows from above.

■

2.2 Soundness wrt Observational Equivalence

The logical relation we have defined above is sound with respect to observational equivalence. As before, once we have proved the Fundamental Property we can show that the logical equivalence implies observational equivalence.

Theorem 10 (Soundness wrt Observational Equivalence)
If $\Delta; \Gamma \vdash e \sim e' :: \tau$ *then* $\Delta; \Gamma \vdash e \sim^{obs} e' : \tau$.

2.3 Completeness wrt Contextual Equivalence

The logical relation we have defined above is *not* complete with respect to contextual equivalence. Let $\Delta \vdash \tau$. If we try to prove that $\Delta; \Gamma \vdash e \sim^{obs} e' : \tau$ implies $\Delta; \Gamma \vdash e \sim e' : \tau$, we will have to proceed by induction on the type τ—actually, to be precise, by induction on the derivation $\Delta \vdash \tau$. The proof fails to go through for the case when $\tau = \alpha$.

To make our logical relation complete with respect to observational equivalence, we need to impose an additional requirement on relations $R : \sigma \leftrightarrow \sigma'$. We require that such relations be *equivalence-respecting*:

$$\begin{aligned} R : \sigma \leftrightarrow \sigma' \quad \text{iff} \quad \forall (v, v') \in R.\ (\ \vdash v : \sigma\ \wedge\ \vdash v' : \sigma'\ )\ \wedge \\ (\ \forall v_1, v_1'.\ \vdash v \sim^{obs} v_1 : \sigma\ \wedge \\ \vdash v' \sim^{obs} v_1' : \sigma\ \Rightarrow \\ (v_1, v_1') \in R\ ) \end{aligned}$$

3 Theorems for Free

A number of "free theorems" (Wadler 1989) follow as a consequence of parametricity. Here we look at one such theorem, which says that any expression e of type $\forall\alpha.\,\alpha \;\to\; \alpha$ must be equivalent to the polymorphic identity function.

Theorem 11
If $e : \forall\alpha.\alpha \;\to\; \alpha$ *and* $v : \tau$, *then* $e\,[\tau]\; v \to^* v$.

Proof

From the fundamental property, since $e : \forall\alpha.\alpha \;\to\; \alpha$, it follows that $\vdash e \sim e : \forall\alpha.\alpha \;\to\; \alpha$.

Instantiate the latter with $\emptyset$, $\emptyset$, and $\emptyset$.

Hence, $e \sim e : \forall\alpha.\alpha \;\to\; \alpha \mid \emptyset$.

From the latter, it follows that

- $e \to^* f$ (where f is a value), and
- $f \approx f : \forall alpha.\alpha \;\to\; \alpha \mid \emptyset$.

Hence, $f = \Lambda\alpha.\,.e_1$.

Instantiate $f \approx f : \forall alpha.\alpha \;\to\; \alpha \mid \emptyset$ with τ, τ, and $R = \{(v, v)\}$. Note that

- $R : \tau \leftrightarrow \tau$ since $\vdash v : \tau$.

Hence, $e_1[\tau/\alpha] \sim e_1[\tau/\alpha] : \tau \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$.

From the latter, it follows that

- $e_1[\tau/\alpha] \to^* g$, and
- $g \approx g : \alpha \;\to\; \alpha \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$.

From the latter, it follows that $g = \lambda x{:}\tau.\, e_2$.

Instantiate $g \approx g : \alpha \;\to\; \alpha \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$ with v and v. Note that

- $v \approx v : \alpha \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$,
  which follows from $(v, v) \in R$.

Hence, $e_2[v/x] \sim e_2[v/x] : \alpha \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$.

From the latter, it follows that

- $e_2[v/x] \to^* v'$, and
- $v' \approx v' : \alpha \mid \emptyset[\alpha \mapsto (\tau, \tau, R)]$.

From the latter, it follows that $(v', v') \in R$.

But then, since $R = \{(v, v)\}$, it must be that $v' = v$.

Thus, we have that $e_2[v/x] \rightarrow^* v$.

Hence, it follows from above that:

$$\begin{array}{lll} e\,[\tau]\ v & \rightarrow^* & f\,[\tau]\ v \\ & \equiv & (\Lambda\alpha.\, e_1)\,[\tau]\ v \\ & \rightarrow & e_1[\tau/\alpha]\ v \\ & \rightarrow^* & g\ v \\ & \equiv & \lambda x{:}\tau.\, e_2\ v \\ & \rightarrow & e_2[v/x] \\ & \rightarrow^* & v' \\ & \equiv & v \end{array}$$

■

4 Representation Independence

The typing rule for unpack ensures that the client of an abstract type is polymorphic in the representation type. A consequence of parametricity is that this means that the behavior of the client is independent of the choice of representation.

To say that no client can distinguish between two implementations of an existential type means that the two implementations are observationally equivalent. Thus representation independence for abstract types boils down to observational equivalence. But as we explained above, it is difficult to reason about observational equivalence directly. Instead we use the logical relation defined above to establish the equivalence of two implementations of an abstrat type.

A relation $R : \tau_1 \leftrightarrow \tau_2$ is a *bisimulation* between two packages of type $\exists\alpha.\,\tau$: $\mathsf{pack}\,\tau_1, v_1\,\mathsf{as}\,\exists\alpha.\,\tau$ and $\mathsf{pack}\,\tau_2, v_2\,\mathsf{as}\,\exists\alpha.\,\tau$ if and only if:

$$v_1 \approx v_2 : \tau \mid \emptyset[\alpha \mapsto (\tau_1, \tau_2, R)]$$

When two packages e_1 and e_2 are *bisimilar*, they are indistinguishable by any client — that is, the execution of the client cannot depend on whether e_1 or e_2 is the provided package.

In other words, if $\alpha; x : \tau \vdash e_c : \tau_c$, then

$$\mathsf{unpack}\,e_1\,\mathsf{as}\,\alpha, x\,\mathsf{in}\,e_c \sim \mathsf{unpack}\,e_2\,\mathsf{as}\,\alpha, x\,\mathsf{in}\,e_c : \tau_c \mid \emptyset$$

4.1 Encoding Existential Types using Universal Types

That parametricity is the essence of representation independence is related to the fact that existential types can be encoded using universal types. The typing rule for unpack is central to the following encoding. Specifically, from the

premise of the unpack rule (for a closed unpack expression) that typechecks the client expression $\alpha; x : \tau \vdash e_c : \tau_c$, it follows that we can think of the client of an existential package of type $\exists\alpha.\tau$ as the function $\Lambda\alpha.\, e_c$ of type $\forall\alpha.\tau \;\rightarrow\; \tau_c$.

$$
\begin{array}{lcl}
\exists\alpha.\tau & \equiv & \forall\beta.\ (\forall\alpha.\ \tau \rightarrow \beta) \rightarrow \beta \\
\mathsf{pack}\,\sigma, e\,\mathsf{as}\,\exists\alpha.\,\tau & \equiv & \Lambda\beta.\,\lambda c{:}(\forall\alpha.\tau \rightarrow \beta).\, c\,[\sigma]\ e \\
\mathsf{unpack}\,e\,\mathsf{as}\,\alpha, x\,\mathsf{in}\,e_c & \equiv & e\,[\tau_c]\ (\Lambda\alpha.\,\lambda x{:}\tau.\, e_c)
\end{array}
$$