Lecture Notes 12
Parametricity

Carlo Angiuli

B522: PL Foundations
April 21, 2025

these notes are even shorter on words...

In this lecture, we will prove, or at least sketch the proof of, the *parametricity theorem* for System F (Theorem 12.21), which states that a certain binary logical relation is reflexive. The key idea behind parametricity is that we can "instantiate" type variables in System F not only with types but with a much larger class of type-like relations. We can take advantage of this fact to derive various free theorems and representation independence results.

This lecture will draw heavily upon ideas from the past three lectures on termination, observational equivalence, and System F. The parametricity theorem and its consequences are covered in Chapter 48 of Harper [Har16], although our presentation is somewhat different and closer to Sections 4 and 5 of Lau Skorstengaard's notes from Amal Ahmed's OPLSS lectures on logical relations.

1 Termination and *candidats de réducibilité*

The goal of the parametricity theorem is to characterize program equivalence in System F, so we will once again extend our language with a type `ans` with two values `yes` and `no`. As with the STLC, this lets us easily define a notion of program outcome with two distinct possibilities.

The statement of the parametricity theorem is quite involved, so we will start by discussing the key idea in isolation. Let's imagine trying to extend our termination proof for STLC—which you may recall involved a unary logical relation called *hereditary termination*—to System F. (Our goal is not in fact to prove termination, although we will end up proving termination for `ans` *en passant*.)

Just as in the STLC, we cannot prove termination by a direct inductive argument because the conjunction of $f \Downarrow$ and $e \Downarrow$ does not imply $f\ e \Downarrow$. In order to apply

the technique of hereditary termination, we need to decide what it means for a term to be hereditarily terminating at type $\forall\alpha.\tau$. A natural but flawed choice is the following:

Invalid Definition (Hereditary termination)**.** A closed term e is *hereditarily terminating at type* τ, or $\mathbf{HT}_\tau(e)$, when:

- If $\tau = \mathsf{ans}$, then $e \Downarrow \mathtt{yes}$ or $e \Downarrow \mathtt{no}$.
- If $\tau = \tau_1 \to \tau_2$, then for all $\cdot \vdash e_1 : \tau_1$ such that $\mathbf{HT}_{\tau_1}(e_1)$, we have $\mathbf{HT}_{\tau_2}(e\ e_1)$.
- If $\tau = \forall\alpha.\tau_2$, then for all $\cdot \vdash \tau_1\ \mathsf{ty}$, $\mathbf{HT}_{\tau_2[\tau_1/\alpha]}(e@\tau_1)$.

The issue with this definition is subtle but fatal: it is actually circular! We are defining $\mathbf{HT}_\tau$ by structural recursion on τ, but the definition of $\mathbf{HT}_{\forall\alpha.\tau_2}$ makes reference to every $\mathbf{HT}_{\tau_2[\tau_1/\alpha]}$ which includes type expressions that are just as large or even larger than $\forall\alpha.\tau_2$ itself! Compare this to the definition of $\mathbf{HT}_{\tau_1\to\tau_2}$, which refers only to $\mathbf{HT}_{\tau_1}$ and $\mathbf{HT}_{\tau_2}$.

To see concretely where this goes wrong, let us rephrase the definition as an infinite conjunction: we define $\mathbf{HT}_{\forall\alpha.\alpha}(e)$ to hold if and only if $\mathbf{HT}_{\mathsf{ans}}(e@\mathsf{ans})$ holds, and $\mathbf{HT}_{\mathsf{ans}\to\mathsf{ans}}(e@(\mathsf{ans}\to\mathsf{ans}))$ holds, and $\mathbf{HT}_{\forall\alpha.\alpha}(e@(\forall\alpha.\alpha))$ holds, and $\mathbf{HT}_{\forall\alpha.\alpha\to\alpha}(e@(\forall\alpha.\alpha\to\alpha))$ holds, and…

Remark 12.1. The fact that $\forall\alpha.\tau$ quantifies over all types—including $\forall\alpha.\tau$—is known as *impredicative* quantification/polymorphism. The distinction between impredicative and predicative quantification is a big topic, but for now we simply note:

- Impredicativity is an essential ingredient of our Church encodings. Defining the function $\mathtt{not} : \mathtt{bool} \to \mathtt{bool}$, for example, requires instantiating the type quantifier of a term of type $\mathtt{bool} = \forall\alpha.\alpha \to \alpha \to \alpha$ at $\mathtt{bool}$ itself.
- On the other hand, the parametric nature of System F's polymorphism is quite separate from its impredicative nature; it is possible to study generics and free theorems in the setting of what is called *predicative* System F.

2 Parametricity

Definition 12.2. A *candidate relation* or *candidate* is a triple (τ, τ', R) such that τ, τ' are closed types, R is a binary relation between closed terms of type τ and closed terms of type τ', and R is closed under head expansion in both arguments.

Definition 12.3. A *relational environment* ρ for type context Δ, written $\rho : \Delta$, assigns a candidate to each type variable in Δ. That is,

- $\cdot : \cdot$ and
- $(\rho, (\tau, \tau', R)/\alpha) : (\Delta, \alpha\ \mathsf{ty})$ if $\rho : \Delta$ and (τ, τ', R) is a candidate.

Given $\rho : \Delta$ we can extract closing type substitutions $\mathsf{lhs}(\rho), \mathsf{rhs}(\rho)$

$$\mathsf{lhs}(\cdot) = \cdot \qquad \mathsf{rhs}(\cdot) = \cdot$$
$$\mathsf{lhs}(\rho, (\tau, \tau', R)/\alpha) = \mathsf{lhs}(\rho), \tau/\alpha \qquad \mathsf{rhs}(\rho, (\tau, \tau', R)/\alpha) = \mathsf{rhs}(\rho), \tau'/\alpha$$

as well as look up a binary relation $\rho(\alpha)$ between $\alpha[\mathsf{lhs}(\rho)]$ and $\alpha[\mathsf{rhs}(\rho)]$ for any type variable α in Δ.

For $\rho : \Delta$ and $\Delta \vdash \tau\ \mathsf{ty}$ we define $e \sim e' : \tau\ [\rho]$ as a binary relation between terms $\cdot; \cdot \vdash e : \tau[\mathsf{lhs}(\rho)]$ and $\cdot; \cdot \vdash e' : \tau[\mathsf{lhs}(\rho)]$ by induction on τ.

Definition 12.4. Given $\cdot; \cdot \vdash e : \mathsf{ans}$ and $\cdot; \cdot \vdash e' : \mathsf{ans}$ we say that e and e' are *Kleene equivalent*, written $e \simeq e'$, if either $e \Downarrow \mathsf{yes}$ and $e' \Downarrow \mathsf{yes}$, or $e \Downarrow \mathsf{no}$ and $e' \Downarrow \mathsf{no}$.

Invariant of following definition: $\mathsf{dom}(\rho) \vdash \tau\ \mathsf{ty}$.

Definition 12.5 (Closed logical equivalence)**.** We define $e \sim e' : \tau\ [\rho]$ by induction on τ as follows:

- $e \sim e' : \mathsf{ans}\ [\rho]$ when $e \simeq e'$.
- $e \sim e' : \tau_1 \to \tau_2\ [\rho]$ for all $e_1 \sim e_1' : \tau_1\ [\rho]$ we have $e\ e_1 \sim e'\ e_1' : \tau_2\ [\rho]$.
- $e \sim e' : \forall \alpha.\tau_2\ [\rho]$ when for all candidates (τ, τ', R) we have $e@\tau \sim e'@\tau' : \tau_2\ [\rho, (\tau, \tau', R)/\alpha]$.
- $e \sim e' : \exists \alpha.\tau_2\ [\rho]$ when

  $$e \Downarrow \mathsf{pack}\ \langle \tau, e_2 \rangle\ \mathsf{as}\ \exists \alpha.\tau_2 \text{ and } e' \Downarrow \mathsf{pack}\ \langle \tau', e_2' \rangle\ \mathsf{as}\ \exists \alpha.\tau_2$$

  and there exists a candidate (τ, τ', R) such that $e_2 \sim e_2' : \tau_2\ [\rho, (\tau, \tau', R)/\alpha]$.
- $e \sim e' : \alpha\ [\rho]$ when $e\ \rho(\alpha)\ e'$.

Lemma 12.6 (Head expansion)**.** *If $d \longmapsto^* e$ and $d' \longmapsto^* e'$ and $e \sim e' : \tau\ [\rho]$ then $d \sim d' : \tau\ [\rho]$.*

Corollary 12.7. *For $\rho : \Delta$ and $\Delta \vdash \tau\ \mathsf{ty}$, logical equivalence is a candidate relation $(\tau[\mathsf{lhs}(\rho)], \tau[\mathsf{rhs}(\rho)], - \sim - : \tau\ [\rho])$.*

Lemma 12.8 (Compositionality)**.** *Suppose* $\Delta, \alpha\ \mathsf{ty} \vdash \tau\ \mathsf{ty}$*,* $\Delta \vdash \tau_1\ \mathsf{ty}$*, and* $\rho : \Delta$*. Then for all* $\cdot;\cdot \vdash e : \tau[\tau_1/\alpha][\mathsf{lhs}(\rho)]$ *and* $\cdot;\cdot \vdash e' : \tau[\tau_1/\alpha][\mathsf{rhs}(\rho)]$*,*

$$e \sim e' : \tau[\tau_1/\alpha]\ [\rho] \iff e \sim e' : \tau\ [\rho, (\tau_1[\mathsf{lhs}(\rho)], \tau_1[\mathsf{rhs}(\rho)], - \sim - : \tau_1\ [\rho])/\alpha]$$

Proof. By structural induction on τ; note that Corollary 12.7 permits us to extend ρ with logical equivalence at τ_1. □

For a typing context Γ well-formed in Δ and two closing substitutions γ, γ' we define logical equivalence as pointwise (heterogeneous) logical equivalence.

Definition 12.9. We define $\gamma \sim \gamma' : \Gamma\ [\rho]$ by induction on Γ as follows:

- $\cdot \sim \cdot : \cdot\ [\rho]$, and
- $(\gamma, e/x) \sim (\gamma', e'/x) : (\Gamma, x : \tau)\ [\rho]$ if $\gamma \sim \gamma' : \Gamma\ [\rho]$ and $e \sim e' : \tau\ [\rho]$.

Lemma 12.10 (Weakening)**.** *Suppose* $\rho : \Delta$ *and* (τ, τ', R) *is a candidate. Then:*

1. *If* $\Delta \vdash \tau_1\ \mathsf{ty}$ *then for all* $\cdot;\cdot \vdash e : \tau_1[\mathsf{lhs}(\rho)]$ *and* $\cdot;\cdot \vdash e' : \tau_1[\mathsf{rhs}(\rho)]$*,*

$$e \sim e' : \tau_1\ [\rho] \iff e \sim e' : \tau_1\ [\rho, (\tau, \tau', R)/\alpha]$$

2. *If* Γ *is well-formed in* Δ*, then for all* $\gamma : \Gamma[\mathsf{lhs}(\rho)]$ *and* $\gamma' : \Gamma[\mathsf{rhs}(\rho)]$*,*

$$\gamma \sim \gamma' : \Gamma\ [\rho] \iff \gamma \sim \gamma' : \Gamma\ [\rho, (\tau, \tau', R)/\alpha]$$

Definition 12.11 (Open logical equivalence)**.** We say that two terms $\Delta; \Gamma \vdash e : \tau$ and $\Delta; \Gamma \vdash e' : \tau'$ are *(open) logically equivalent*, written $\Delta; \Gamma \vdash e \sim e' : \tau$, if for all relational environments $\rho : \Delta$ and all $\gamma \sim \gamma' : \Gamma\ [\rho]$ we have $e[\mathsf{lhs}(\rho)][\gamma] \sim e'[\mathsf{rhs}(\rho)][\gamma'] : \tau\ [\rho]$.

We can now prove what are called *compatibility lemmas* stating that each typing rule respects open logical equivalence.

Lemma 12.12 (Compatibility for ans-$\textsc{intro}_1$)**.** $\Delta; \Gamma \vdash \mathsf{yes} \sim \mathsf{yes} : \mathsf{ans}$.

Lemma 12.13 (Compatibility for ans-$\textsc{intro}_2$)**.** $\Delta; \Gamma \vdash \mathsf{no} \sim \mathsf{no} : \mathsf{ans}$.

Lemma 12.14 (Compatibility for $\textsc{var}$)**.** $\Delta; \Gamma, x : \tau \vdash x \sim x : \tau$.

Lemma 12.15 (Compatibility for $\to$-$\textsc{intro}$)**.** *If* $\Delta; \Gamma, x : \tau_1 \vdash e_2 \sim e_2' : \tau_2$ *then* $\Delta; \Gamma \vdash \lambda x : \tau_1.e_2 \sim \lambda x : \tau_1.e_2' : \tau_1 \to \tau_2$.

Lemma 12.16 (Compatibility for $\to$-ELIM)**.** *If* $\Delta; \Gamma \vdash f \sim f' : \tau_1 \to \tau_2$ *and* $\Delta; \Gamma \vdash e_1 \sim e_1' : \tau_1$ *then* $\Delta; \Gamma \vdash f\ e_1 \sim f'\ e_1' : \tau_2$.

Lemma 12.17 (Compatibility for $\forall$-INTRO)**.** *If* $\Delta, \alpha\ \mathsf{ty}; \Gamma \vdash e \sim e' : \tau_2$ *then* $\Delta; \Gamma \vdash \Lambda\alpha.e \sim \Lambda\alpha.e' : \forall\alpha.\tau_2$.

Proof. Uses Lemma 12.10. □

Lemma 12.18 (Compatibility for $\forall$-ELIM)**.** *If* $\Delta; \Gamma \vdash e \sim e' : \forall\alpha.\tau$ *and* $\Delta \vdash \tau_1\ \mathsf{ty}$ *then* $\Delta; \Gamma \vdash e@\tau_1 \sim e'@\tau_1 : \tau[\tau_1/\alpha]$.

Proof. Uses Lemma 12.8. □

Lemma 12.19 (Compatibility for $\exists$-INTRO)**.** *If* $\Delta \vdash \tau_r\ \mathsf{ty}$, $\Delta, \alpha\ \mathsf{ty} \vdash \tau_i\ \mathsf{ty}$, *and* $\Delta; \Gamma \vdash e \sim e' : \tau_i[\tau_r/\alpha]$, *then* $\Delta; \Gamma \vdash \mathsf{pack}\ \langle \tau_r, e \rangle\ \mathsf{as}\ \exists\alpha.\tau_i \sim \mathsf{pack}\ \langle \tau_r, e' \rangle\ \mathsf{as}\ \exists\alpha.\tau_i : \exists\alpha.\tau_i$.

Proof. Uses Lemma 12.8. □

Lemma 12.20 (Compatibility for $\exists$-ELIM)**.** *If* $\Delta; \Gamma \vdash e \sim e' : \exists\alpha.\tau_i$, $\Delta \vdash \tau_2\ \mathsf{ty}$, *and* $\Delta, \alpha\ \mathsf{ty}; \Gamma, x : \tau_i \vdash e_2 \sim e_2' : \tau_2$, *then* $\Delta; \Gamma \vdash \mathsf{unpack}\ \langle \alpha, x \rangle = e\ \mathtt{in}\ e_2 \sim \mathsf{unpack}\ \langle \alpha, x \rangle = e'\ \mathtt{in}\ e_2' : \tau_2$.

Proof. Uses Lemma 12.10. □

Theorem 12.21 (Parametricity)**.** *If* $\Delta; \Gamma \vdash e : \tau$ *then* $\Delta; \Gamma \vdash e \sim e : \tau$.

Proof. By structural induction on the typing judgment, straightforwardly applying compatibility lemmas. □

Corollary 12.22 (Termination for ans)**.** *If* $\cdot; \cdot \vdash e : \mathsf{ans}$ *then* $e \Downarrow \mathsf{yes}$ *or* $e \Downarrow \mathsf{no}$.

It is possible to prove that open logically equivalent terms are observationally equivalent, although we skip this proof for now. Proof sketch: open logical equivalence is a consistent congruence; by termination, observational equivalence is a consistent congruence; observational equivalence is the coarsest such.

3 Deriving free theorems

The primary consequence of Theorem 12.21 is that System F's polymorphism is so strong that we can instantiate terms of type $\forall\alpha.\tau$ at arbitrary candidate relations. Many free theorems follow directly from parametricity if we can cook up the right candidate.

Theorem 12.23 (Polymorphic identity)**.** *If* $\cdot;\cdot \vdash f : \forall\alpha.\alpha \to \alpha$*, then for all* $\cdot;\cdot \vdash e : \tau$ *we have* $f@\tau\ e \longmapsto^* e$*.*

Proof. By Theorem 12.21 we have $\cdot;\cdot \vdash f \sim f : \forall\alpha.\alpha \to \alpha$. Unfolding definitions, this means that for all relational environments $\cdot : \cdot$ and all closing substitutions $\cdot \sim \cdot : \cdot\ [\cdot]$ we have $f[\mathsf{lhs}(\cdot)][\cdot] \sim f[\mathsf{rhs}(\cdot)][\cdot] : \forall\alpha.\alpha \to \alpha\ [\cdot]$, i.e., we have the closed logical equivalence $f \sim f : \forall\alpha.\alpha \to \alpha\ [\cdot]$. Expanding this definition, we know that for all candidates (τ, τ', R) we have $f@\tau \sim f@\tau' : \alpha \to \alpha\ [(\tau, \tau', R)/\alpha]$.

We will choose the candidate (τ, τ, R) defined by

$$d\ R\ d' \iff (d \longmapsto^* e \text{ and } d' \longmapsto^* e)$$

It is easy to see that R is closed under head expansion. Expanding the definition of $f@\tau \sim f@\tau : \alpha \to \alpha\ [(\tau, \tau, R)/\alpha]$, we know that for any $e_1 \sim e_1' : \alpha\ [(\tau, \tau, R)/\alpha]$ we have $f@\tau\ e_1 \sim f@\tau\ e_1' : \alpha\ [(\tau, \tau, R)/\alpha]$, or rewriting once more, that for any $e_1\ R\ e_1'$ we have $(f@\tau\ e_1)\ R\ (f@\tau\ e_1')$.

We choose $e_1 = e_1' = e$ (which satisfies $e\ R\ e$), concluding $(f@\tau\ e)\ R\ (f@\tau\ e)$ and hence $f@\tau\ e \longmapsto^* e$ as required. □

Theorem 12.24 (Polymorphic equality)**.** *If* $\cdot;\cdot \vdash f : \forall\alpha.\alpha \to \alpha \to \mathsf{ans}$*, then for all* $\cdot;\cdot \vdash e_1 : \tau$*,* $\cdot;\cdot \vdash e_2 : \tau$*,* $\cdot;\cdot \vdash e_1' : \tau'$*, and* $\cdot;\cdot \vdash e_2' : \tau'$ *we have* $f@\tau\ e_1\ e_2 \simeq f@\tau'\ e_1'\ e_2'$*.*

Proof. This proof proceeds similarly to the previous one, but we will choose to instantiate α at the candidate (τ, τ', R) where $e\ R\ e'$ always holds. (Again, this relation is closed under head expansion.) Unfolding definitions more rapidly, by Theorem 12.21 we have $f \sim f : \forall\alpha.\alpha \to \alpha \to \mathsf{ans}\ [\cdot]$. Instantiating at (τ, τ', R),

$$f@\tau \sim f@\tau' : \alpha \to \alpha \to \mathsf{ans}\ [(\tau, \tau', R)/\alpha]$$

and thus

$$e\ R\ e' \implies f@\tau\ e \sim f@\tau'\ e' : \alpha \to \mathsf{ans}\ [(\tau, \tau', R)/\alpha]$$

Plugging in $e_1\ R\ e_1'$ (which holds by the definition of R) we have

$$f@\tau\ e_1 \sim f@\tau'\ e_1' : \alpha \to \mathsf{ans}\ [(\tau, \tau', R)/\alpha]$$

and thus

$$e\ R\ e' \implies f@\tau\ e_1\ e \sim f@\tau'\ e_1'\ e' : \mathsf{ans}\ [(\tau, \tau', R)/\alpha]$$

Finally, plugging in $e_2\ R\ e_2'$ we have

$$f@\tau\ e_1\ e_2 \sim f@\tau'\ e_1'\ e_2' : \mathsf{ans}\ [(\tau, \tau', R)/\alpha]$$

and thus $f@\tau\ e_1\ e_2 \simeq f@\tau'\ e_1'\ e_2'$ as required. □

4 Deriving representation independence

Proving representation independence theorems is slightly more involved. Given two pack terms of existential type and a bisimulation between them:

1. We use that bisimulation as the candidate relation to establish logical equivalence of the two pack terms;
2. We use the parametricity theorem (Theorem 12.21) to see that all clients are logically equivalent to themselves; and
3. We use the compatibility lemma for unpack (Lemma 12.20) to conclude that unpacking two logically equivalent terms in the same client produces logically equivalent results (and hence Kleene equivalent results, if the client has type ans).

These ideas scale directly to arbitrarily complex interfaces, but for the sake of this lecture we will pick a rather simple "toggling" interface with an abstract type α, a starting state α, a toggling function $\alpha \to \alpha$, and a "toBool" function $\alpha \to \mathsf{bool}$ that converts the internal state to a boolean. We will implement two togglers, one using booleans ($\mathtt{impl}_1$) and the other using natural numbers ($\mathtt{impl}_2$).[9]

Theorem 12.25 (Representation independence)**.** *Let us write*

$$\mathtt{Toggler} \coloneqq \exists\alpha.\alpha \times ((\alpha \to \alpha) \times (\alpha \to \mathsf{bool}))$$

$$\mathtt{impl}_1, \mathtt{impl}_2 : \mathtt{Toggler}$$

$$\mathtt{impl}_1 \coloneqq \mathsf{pack}\ \langle \mathsf{bool}, (\mathtt{true}, (\mathtt{not}, \lambda x : \mathsf{bool}.x))\rangle\ \mathsf{as}\ \mathtt{Toggler}$$

$$\mathtt{impl}_2 \coloneqq \mathsf{pack}\ \langle \mathsf{nat}, (\mathtt{zero}, (\lambda n : \mathsf{nat}.\mathsf{suc}(n), \mathtt{even?}))\rangle\ \mathsf{as}\ \mathtt{Toggler}$$

For any $\alpha\ \mathsf{ty}; x : \alpha \times ((\alpha \to \alpha) \times (\alpha \to \mathsf{bool})) \vdash e : \mathsf{ans}$*, we have*

$$(\mathsf{unpack}\ \langle \alpha, x\rangle = \mathtt{impl}_1\ \mathsf{in}\ e) \simeq (\mathsf{unpack}\ \langle \alpha, x\rangle = \mathtt{impl}_2\ \mathsf{in}\ e).$$

instantiate at the evident bisimulation

References

[Har16] Robert Harper. *Practical Foundations for Programming Languages*. Second Edition. Cambridge University Press, 2016. ISBN: 9781107150300. DOI: 10.1017/CBO9781316576892.

[9] We can either extend System F with bool and nat or Church encode them.