Lecture Notes on Parametricity

15-814: Types and Programming Languages
Frank Pfenning

Lecture 16
Tuesday, October 27, 2020

1 Introduction

> **Disclaimer:** The material in this lecture is a redux of presentations by Reynolds [Rey83], Wadler [Wad89], and Harper [Har16, Chapter 48]. The quoted theorems have not been checked against the details of our presentation of the inference rules and operational semantics.

As discussed in the previous lecture, parametric polymorphism is the idea that a function of type $\forall\alpha.\,\tau$ will "behave the same" on all types σ that might be used for α. This has far-reaching consequences, in particular for modularity and data abstraction. As we will see in the next lecture, if a client to a library that hides an implementation type is *parametric* in this type, then the library implementer or maintainer has the opportunity to replace the implementation with a different one without risk of breaking the client code.

The informal idea that a function behaves parametrically in a type variable α is surprisingly difficult to capture technically. Reynolds [Rey83] realized that it must be done *relationally*. For example, a function $f : \forall\alpha.\,\alpha \to \alpha$ is parametric if for any two types τ and σ, and any relation between values of type τ and σ, if we pass f related arguments it will return related results. As an example, let's consider some (unknown) function

$$\cdot \vdash f : \forall\alpha.\,\alpha \to \alpha \to \alpha$$

and assume it *parametric* in its type argument. We have

$$\begin{array}{lcl} f\,[\mathit{bool}] & : & \mathit{bool} \to \mathit{bool} \to \mathit{bool} \\ f\,[\mathit{nat}] & : & \mathit{nat} \to \mathit{nat} \to \mathit{nat} \end{array}$$

Now consider a relation R such that *false* R $\overline{0}$ and *true* R $\overline{n}$ for $n > 0$. If

$$f\ [\mathit{bool}]\ \mathit{false}\ \mathit{true} \mapsto^* \mathit{false}$$

then it must also be the case that, for example,

$$f\ [\mathit{nat}]\ \overline{0}\ \overline{17} \mapsto^* \overline{0}$$

On the other hand, from the indicated behavior and relation we cannot immediately make a statement about

$$f\ [\mathit{nat}]\ \overline{42}\ \overline{0}$$

But we can pick a different relation! Let *false* S $\overline{42}$ and *true* S $\overline{0}$ (and no other values are related). From the relation S and parametricity we conclude

$$f\ [\mathit{nat}]\ \overline{42}\ \overline{0} \mapsto^* \overline{42}$$

We can see that parametricity is quite powerful, since we can tell a lot about the behavior of f without knowing its definition

What Reynolds showed is that in a polymorphic λ-calculus with products and Booleans, all expressions are parametric in this sense.

We begin by recalling extensional equality and then a new form of equality based on the idea of parametricity called *logical equality*.

2 Extensional Equality

In Lecture 8 we defined an *extensional equality* between expressions. We repeat it here, with a few additional cases. First, expressions are simply evaluated to values that are then compared with a more specialized relation.

Expressions: $e \approx e' : \tau$ iff $e \mapsto^* v$, $e' \mapsto^* v'$ with v, v' values, and $v \sim v' : \tau$ or both e and e' diverge.

For positive types (eager pairs, sums, unit) we compare the structure of the values (which are observable), while for negative types (functions, lazy pairs) we apply the destructor and then compare the results.

Functions: $v \sim v' : \tau_1 \to \tau_2$ iff for all $v_1 : \tau_1$ we have $v\ v_1 \approx v'\ v_1 : \tau_2$.

Pairs: $v \sim v' : \tau_1 \times \tau_2$ iff $v = \langle v_1, v_2 \rangle$, $v' = \langle v_1', v_2' \rangle$ and $v_1 \sim v_1' : \tau_1$ and $v_2 \sim v_2' : \tau_2$.

Units: $v \sim v' : 1$ iff $v = \langle\rangle$ and $v' = \langle\rangle$ (which is always the case, by the canonical forms theorem).

Sums: $v \sim v' : \sum_{i \in I}(i : \tau_i)$ iff $v = k \cdot v_k$ and $v' = k \cdot v'_k$ and $v_k \sim v'_k : \tau_k$ for some $k \in I$.

Lazy Pair: $v \sim v' : \tau_1 \,\&\, \tau_2$ iff $\mathsf{fst}\ v \approx \mathsf{fst}\ v' : \tau_1$ and $\mathsf{snd}\ v \approx \mathsf{snd}\ v' : \tau_2$

We didn't state this explicitly, but these can be extended to polymorphic and recursive types, since of recursive types as positive and universal quantification as negative.

Universal Quantification: $v \sim v' : \forall\alpha.\,\tau$ iff for all closed σ we have $v\,[\sigma] \approx v'[\sigma] : [\sigma/\alpha]\tau$.

Recursion: $v \sim v' : \rho\alpha.\,\tau$ iff $v = \mathsf{fold}\ v_1$ and $v' = \mathsf{fold}\ v'_1$ and $v_1 \sim v'_1 : [\rho\alpha.\,\tau/\alpha]\tau$.

These last two cases are different from the earlier ones in that the types do not get smaller, something that will occupy us shortly. Also, it seems at least possible we may get into a chain of reasoning

$$v \sim v' : \forall\alpha.\,\tau \to \tau \quad \text{iff} \quad \ldots \quad \text{iff} \quad v \sim v' : \forall\alpha.\,\tau \to \tau$$

so the equality may somehow not be well-defined.

3 Logical Equality

The notion of extensional equality (and the underlying Kleene equality) are almost sufficient, but it is insufficient when we come to *parametricity*. The problem is that we want to compare expressions not at the same, but at related types. This means, for example, that in comparing e and e' and type $\forall\alpha.\,\tau$ we cannot apply e and e' to the exact *same* type σ. Instead, we must apply them to *related* types. This in turn means that the two expressions we are comparing may not have the same type but *related* types. The notion of equality we derive from this is called *logical equality* because it is based on *logical relations* [Sta85], one of the many connections between logic and computation. We write

$$e \approx e' \in [\![\tau]\!]$$

if the expressions e and e' stand in the relation designated by τ. This is a slight abuse of notation because, as we will see, τ can be more than just a

type. Also, we no longer require that e and e' should have type τ. For the reason explained above, they may not have the same type. Furthermore, they may not even be well-typed anymore which allows a richer set of applications for logical equality. We also have a second relation, designated by $[\tau]$ that applies only to values. We write $v \sim v' \in [\tau]$ if the values v and v' are related by $[\tau]$. We define

Expressions: $e \approx e' \in [\![\tau]\!]$ iff $e \mapsto^* v$ and $e' \mapsto^* v'$ and $v \sim v' \in [\tau]$.

We assume here, to keep the development simple, that all expressions terminate. In fact, logical relations can be used to prove exactly that. The clauses for the positive types remain essentially the same as for extensional equality, where we restrict recursive types to be purely positive.

Pairs: $v \sim v' \in [\tau_1 \times \tau_2]$ iff $v = \langle v_1, v_2 \rangle$ and $v' = \langle v_1', v_2' \rangle$ for some v_1, v_2, v_1', v_2' and $v_1 = v_1' \in [\tau_1]$ and $v_2 = v_2' \in [\tau_2]$.

Unit: $v \sim v' \in [1]$ iff $v = \langle\,\rangle = v'$.

Sums: $v \sim v' \in [\sum_{i \in I}(i : \tau_i)]$ iff $v = k \cdot v_k$ and $v' = k \cdot v_k'$ for some k, v_k and v_k' with $v_k = v_k' \in [\tau_k]$.

Recursion: $v \sim v' \in [\rho\alpha^+.\, \tau^+]$ iff $v = \mathsf{fold}\ v_1$ and $v' = \mathsf{fold}\ v_1'$ and $v_1 = v_1' \in [[\rho\alpha^+.\, \tau^+/\alpha^+]\tau^+]$.

To be explicit, we define the purely positive types as

$$\tau^+ ::= \tau_1^+ \times \tau_2^+ \mid 1 \mid \sum_{i \in I}(i : \tau_i^+) \mid \rho\alpha^+.\, \tau^+ \mid \alpha^+$$

Even though the type becomes larger in the last clause, the definition is not circular because the values we are comparing get smaller. In fact, we can prove that $v \sim v' \in [\tau^+]$ iff $v = v'$. So the clauses for positive types are mostly useful if negative types are embedded in them.

The case for lazy pairs mirrors what we had before, using the destructors.

Lazy Pairs: $v \sim v' \in [\tau_1 \& \tau_2]$ iff $\mathsf{fst}\ v \approx \mathsf{fst}\ v' \in [\![\tau_1]\!]$ and $\mathsf{snd}\ v \approx \mathsf{snd}\ v' \in [\![\tau_2]\!]$

In some circumstances we can use an equivalent formulation where we require v and v' to be a lazy pairs of two related expressions.

The definition becomes different when we come to universal quantification, where we need to be careful to (a) avoid circularity in the definition, and (b) capture the idea behind parametricity. We write $R : \sigma \leftrightarrow \sigma'$ for a relation between values $v : \sigma$ and $v' : \sigma'$, and $v\ R\ v'$ if R relates v and v'. In

some situations when we would like to reason about parametricity using logical relations, we may need to put some conditions on R, but here we think of it as an arbitrary relation on values. We then define

Universal Quantification: $v \sim v' \in [\forall\alpha.\,\tau]$ iff for all closed types σ and σ' and relations $R : \sigma \leftrightarrow \sigma'$ we have $v[\sigma] \approx v'[\sigma'] \in [\![[R/\alpha]\tau]\!]$

(R) $v \sim v' \in [R]$ iff $v\ R\ v'$.

The second clause here is a new base case in the definition of $[\tau]$, in addition to the type 1. It is needed because we substitute an arbitrary relation R for the type variable α in the clause for universal quantification. So when we encounter R we just use it to compare v and v'.

We have taken a big conceptual step, because what we write as type τ actually now contains relations instead of type variables, as well as ordinary type constructors.

For functions, we apply them to *related* arguments and check that their results are again *related*.

Functions: $v \sim v' \in [\tau_1 \to \tau_2]$ iff for all $v_1 \sim v_1' \in [\tau_1]$ we have $v\,v_1 \approx v'\,v_1' \in [\![\tau_2]\!]$

The quantification structure should make it clear that logical equality in general is difficult to establish. It requires a lot: for two arbitrary types and an arbitrary relation between values, we have to establish properties of e and e'. It is an instructive exercise to check that

$$\Lambda\alpha.\,\lambda x.\,x \sim \Lambda\alpha.\,\lambda x.\,x \in [\forall\alpha.\,\alpha \to \alpha]$$

To check: $\Lambda\alpha.\,\lambda x.\,x \sim \Lambda\alpha.\,\lambda x.\,x \in [\forall\alpha.\,\alpha \to \alpha]$
This holds if $(\Lambda\alpha.\,\lambda x.\,x)\,[\sigma] \approx (\Lambda\alpha.\,\lambda x.\,x)\,\sigma' \in [\![R \to R]\!]$
for arbitrary σ, σ' and $R : \sigma \leftrightarrow \sigma'$
This holds if $\lambda x.\,x \sim \lambda x.\,x \in [R \to R]$
This holds if $(\lambda x.\,x)\,v_1 \approx (\lambda x.\,x)\,v_1' \in [\![R]\!]$ for arbitrary $v_1 \sim v_1' \in [R]$
This holds if $v_1 \sim v_1' \in [R]$, which is true by assumption

There is nothing wrong with this proof, but let's turn this reasoning around and present it in the "forward" direction, just to see it in a different form.

Let σ, σ', $R : \sigma \leftrightarrow \sigma'$ be arbitrary	Assumption
$v_1\ R\ v_1'$ for some arbitrary v_1 and v_1'	Assumption
$v_1 \sim v_1' \in [R]$	By defn. of $\sim$ at $[R]$
$(\lambda x.\,x)\,v_1 \approx (\lambda x.\,x)\,v_1' \in [\![R]\!]$	By defn. of $\approx$ at $[\![R]\!]$

$\lambda x.\, x \sim \lambda x.\, x \in [R \to R]$ By defn. of $\sim$ at $[R \to R]$
since v_1 and v_1' were arbitrary

$(\Lambda\alpha.\, \lambda x.\, x)\,[\sigma] \approx (\Lambda\alpha.\, \lambda x.\, x)\,[\sigma'] \in [\![R \to R]\!]$ By defn. of $\approx$ at $[\![R \to R]\!]$

$\Lambda\alpha.\, \lambda x.\, x \sim \Lambda\alpha.\, \lambda x.\, x$ By defn. of $\sim$ at $[\forall\alpha.\, \alpha \to \alpha]$
since σ, σ', and R were arbitrary

Conversely, we can imagine that *knowing* that two expressions are parametrically equal is very powerful, because we can instantiate this with arbitrary types σ and σ' and relations between them. The *parametricity theorem* now states that all well-typed expressions are related to themselves. This property holds in a language without general recursive types and general fixed point expressions.

Theorem 1 (Parametricity [Rey83]) *If* $\cdot\,;\,\cdot \vdash e : \tau$ *then* $e \approx e \in [\![\tau]\!]$

We will not go into the proof of this theorem, but just explore its consequences. Besides the original paper, there are a number of proofs in the literature including in the textbook [Har16, Chapter 48] in a language and formalization that's quite similar to ours. We do not go into detail under which conditions it might be restored in the presence of recursive types and fixed point expressions (see, for example, Ahmed [Ahm06]).

4 Some Useful Properties

In a couple of places we may use the following properties, which follow directly from small-step determinism (sequentiality) and the definition of $[\![\tau]\!]$.

(Closure under Expansion) If $e \approx e' \in [\![\tau]\!]$ and $e_0 \mapsto^* e$ and $e_0' \mapsto^* e'$ then $e_0 \approx e_0' \in [\![\tau]\!]$.

(Closure under Reduction) If $e \approx e' \in [\![\tau]\!]$ and $e \mapsto^* e_0$ and $e' \mapsto^* e_0'$ then $e_0 \approx e_0' \in [\![\tau]\!]$.

Also, the call-by-value strategy entails the following properties for reasoning about logical equality.

(Closure under Application) If $e_1 \approx e_1' \in [\![\tau_2 \to \tau_1]\!]$ and $e_2 \approx e_2' \in [\![\tau_2]\!]$ then $e_1\, e_2 \approx e_1'\, e_2' \in [\![\tau_1]\!]$.

(Closure under Type Application) If $e \approx e' \in [\![\forall\alpha.\, \tau]\!]$ and $R : \sigma \leftrightarrow \sigma'$ then $e[\sigma] \approx e'[\sigma'] \in [\![[R/\alpha]\tau]\!]$.

5 Exploiting Parametricity

Parametricity allows us to deduce information about functions knowing only their (polymorphic) types. For example, with only terminating functions, the type

$$f : \forall\alpha.\, \alpha \to \alpha$$

implies that f behaves like the identity function! We express this first by proving

$$f\ [\sigma_0]\ v_0 \mapsto^*\ v_0 \quad \text{for all types } \sigma_0 \text{ and values } v_0 : \sigma_0$$

Later, we prove this property in a second form, namely that f is logically equivalent to the polymorphic identity.

For simplicity, assume f is a value. By the parametricity theorem, we have

$$f \approx f \in [\![\forall\alpha.\, \alpha \to \alpha]\!]$$

By definition of $[\![-]\!]$ and the fact that f is a value, we obtain

$$f \sim f \in [\forall\alpha.\, \alpha \to \alpha]$$

By definition of $[\forall\alpha.-]$, this entails that

$$f\ [\tau] \approx f\ [\tau'] \in [\![R \to R]\!]$$

for any τ, τ', and $R : \tau \leftrightarrow \tau'$. In view of the property we want to show, we pick $\tau = \tau' = \sigma_0$ and R_0 such that $v_0\ R_0\ v_0$. That is, $R_0 : \sigma_0 \leftrightarrow \sigma_0$ relates only v_0 to itself and not any other values. This mean we have

$$f\ [\sigma_0] \approx f\ [\sigma_0] \in [\![R_0 \to R_0]\!]$$

Next, by definition of $[\![-]\!]$ we find $f\ [\sigma_0] \mapsto^*\ f_{\sigma_0}$ for some value f_{σ_0} and

$$f_{\sigma_0} \sim f_{\sigma_0} \in [R_0 \to R_0]$$

By definition of $[\_ \to \_]$ this means that for any value v such that $v \sim v \in [R_0]$ we have $f_{\sigma_0}\ v \approx f_{\sigma_0}\ v \in [R_0]$. We pick $v = v_0$ because $v_0\ R_0\ v_0$ and consequently also

$$v_0 \sim v_0 \in [R_0]$$

Therefore we conclude

$$f_{\sigma_0}\ v_0 \approx f_{\sigma_0}\ v_0 \in [\![R_0]\!]$$

Again, by definition of $[\![-]\!]$ we know that $f_{\sigma_0}\, v_0 \mapsto^* w$ for some w with

$$w \sim w \in [R_0]$$

which in turn is the case if and only if

$$w\ R_0\ w$$

by the definition of $[R_0]$. But R_0 was chosen so it relates only v_0 to v_0, so we conclude that

$$w = v_0$$

Unwinding the chain of evaluations under our call-by-value strategy we find

$$f\ [\sigma_0]\ v_0 \mapsto^* f_{\sigma_0}\ v_0 \mapsto^* w = v_0$$

and our theorem is proved.

Out next goal is to show that any value $f : \forall\alpha.\, \alpha \to \alpha$ is (logically) equivalent to the identity function

$$f \sim \Lambda\alpha.\, \lambda x.\, x \in [\forall\alpha.\, \alpha \to \alpha]$$

Let's prove this. Unfortunately, the first few steps are the "difficult" direction of the parametricity.

By definition, this means to show that

> *For every pair of types σ and σ' and relation $R : \sigma \leftrightarrow \sigma'$, we have*
> $f\,[\sigma] \approx (\Lambda\alpha.\, \lambda x.\, x)\,[\sigma'] \in [\![R \to R]\!]$

Now fix arbitrary σ, σ' and R. Because $(\Lambda\alpha.\, \lambda x.\, x)\,[\sigma'] \mapsto \lambda x.\, x$, our desired conclusion holds if $f\,[\sigma] \mapsto^* f_\sigma$ for some value f_σ and

$$f_\sigma \sim \lambda x.\, x \in [R \to R]$$

Applying the definition of $[\_ \to \_]$, this is true if

> *For all $v \sim v' \in [R]$ we have $f_\sigma\, v \approx (\lambda x.\, x)\, v' \in [\![R]\!]$*

So assume $v \sim v' \in [R]$. It remains to show that

> *$f_\sigma\, v \mapsto^* w$ for some w with $w \sim v' \in [R]$.*

By the previous argument (starting from the parametricity of f) we know that $f_\sigma\, v \mapsto^* v$, so determinism gives us $w = v$. Then $w\ R\ v'$ follows from $v\ R\ v'$ and $w = v$.

Let's summarize the reasoning.

To show:	$f \sim \Lambda\alpha.\, \lambda x.\, x \in [\forall\alpha.\, \alpha \to \alpha]$
True, if	$f\,[\sigma] \approx (\Lambda\alpha.\, \lambda x.\, x)\,[\sigma'] \in [\![R \to R]\!]$ for arbitrary σ, σ', $R : \sigma \leftrightarrow \sigma'$
True, if	$f_\sigma \sim \lambda x.\, x \in [R \to R]$ for $f\,[\sigma] \mapsto^* f_\sigma$
True, if	$f_\sigma\, v \approx (\lambda x.\, x)\, v' \in [\![R \to R]\!]$ for arbitrary $v \sim v' \in [R]$
True, if	$w \sim v' \in [R]$ for $f_\sigma\, v \mapsto^* w$
Holds, since	$f_\sigma\, v \mapsto^* v$ (by prior theorem) and determinism imply $w = v$

Similar proofs show, for example, that $f : \forall\alpha.\, \alpha \to \alpha \to \alpha$ must be equal to the first or second projection function. It is instructive to reason through the details of such arguments. At the beginning of the next lecture we explore additional consequences of parametricity, so-called "*theorems for free*" [Wad89].

Exercises

Exercise 1 Prove that $\forall\alpha.\, \alpha \to \alpha \cong 1$. You may use the results of Section 3 and Section 5.

Exercise 2 Prove, using parametricity, that there cannot be a closed value $f : \forall\alpha.\, \alpha$.

Exercise 3 Prove, using parametricity, that if we have $f : \forall\alpha.\, \alpha \to \alpha \to \alpha$ for a value f then either $f \sim \Lambda\alpha.\, \lambda x.\, \lambda y.\, x \in [\forall\alpha.\, \alpha \to \alpha \to \alpha]$ or $f \sim \Lambda\alpha.\, \lambda x.\, \lambda y.\, y \in [\forall\alpha.\, \alpha \to \alpha \to \alpha]$.

Exercise 4 Prove, using parametricity, that $\forall\alpha.\, \alpha \to \alpha \to \alpha \cong 2$.

References

[Ahm06] Amal J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In P. Sestoft, editor, *15th European Symposium on Programming (ESOP 2006)*, pages 69–83, Vienna, Austria, March 2006. Springer LNCS 3924.

[Har16] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, second edition, April 2016.

[Rey83] John C. Reynolds. Types, abstraction, and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing 83*, pages 513–523. Elsevier, September 1983.

[Sta85] Richard Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[Wad89] Philip Wadler. Theorems for free! In J. Stoy, editor, *Proceedings of the 4th International Conference on Functional Programming Languages and Computer Architecture (FPCA'89)*, pages 347–359, London, UK, September 1989. ACM.