Lecture Notes on Parametricity

15-814: Types and Programming Languages
Frank Pfenning

Lecture 14
Thu Oct 23, 2025

1 Introduction

In the last lecture we proved termination for the quantifier-free fragment of our language using a *logical predicate* (also known as *unary logical relation*. In this lecture we explore what happens when we allow *universal quantification*, that is, abstraction over types. As introduced in prior lecture, we obtain the *polymorphic λ-calculus* (**??**) which is closely related to Girard's System F (**??**). Somewhat miraculously, this system still enjoys termination. Some prior proofs of termination focus on *normalization* and *strong normalization* which complicate the definition and proofs but follows a similar blueprint. Here, as in the previous lecture, we focus on termination in a call-by-value setting. The principal difference is that we don't reduce under λ-abstractions, essentially because we deem the structure of functions not to be observable.

2 Taming the Quantifier

As remarked at the end of the last lecture, we cannot just define

$$v \in [\forall\alpha.\,\tau] \quad \text{iff} \quad v[\sigma] \in [\![[\sigma/\alpha]\tau]\!] \text{ for all } \sigma \quad \text{not a well-founded definition!}$$

The problem is that the instantiating type σ could be or contain $\forall\alpha.\,\tau$ and the definition would deteriorate from an induction over the structure of the type to being circular.

Now we remember the intuitive property discussed earlier: a function $f : \forall\alpha.\,\tau$ behaves "the same" on all types σ that may be substituted for α. So, for example, $f : \forall\alpha.\,\alpha \to \alpha$ must in fact be (equivalent to) the identity function. We say that the function must be *parametric in α*. A non-parametric function would be allowed to behave differently on different types. For example, it might be the negation function on booleans and the successor function on natural numbers.

The semantic brackets $[-]$ *interpret a type as a set of values* (while $[\![-]\!]$ interprets a type as a set of expressions). So we interpret the type variable α as an *arbitrary set of values* rather than a syntactic type. This is possible because the function can **not** depend on the actual values, so *any* values are okay. This breaks the circularity in the definition, because when we encounter one of the value sets we just test membership.

$$\begin{array}{lll} f \in [\forall\alpha.\,\tau] & \text{iff} & v[S] \in [\![[S/\alpha]\tau]\!] \text{ for all sets of values } S \\ v \in [S] & \text{iff} & v \in S \end{array}$$

A slightly unpleasant aspect of this definition is that the language of types now has to account for arbitrary value sets. To avoid this unpleasantness, we can record the mapping from α to S separately. Writing η for such a substitution, the definition would then be parameterized by a substitution. For example, we might write

$$\begin{array}{lll} \eta \models f \in [\forall \alpha.\, \tau] & \text{iff} & \eta, S/\alpha \models v[\alpha] \in [\![\tau]\!] \text{ for all sets of values } S \\ \eta \models v \in [\alpha] & \text{iff} & \eta(v) \in \eta(\alpha) \end{array}$$

This becomes a little heavy in the long run, so we'll carry out the substitution and remember that it could be rewritten in the style that separates syntax and semantics more clearly and explicitly.

For many proofs we'd need conditions on the formation of the sets which Girard's calls *reducibility candidates*. I don't think we'll need them here. We also obtain for free from the definition of $[\![-]\!]$ (which remains unchanged):

$$e \in [\![S]\!] \quad \text{iff} \quad e \mapsto^* v \text{ for some } v \in [S]$$

Other conditions might enforce that the values are well-typed, but there are actually applications where expressions and values are not restricted to be well-typed. These are characterized by proving the semantic soundness of a language extended by additional expressions that cannot be typed within the discipline, but are nevertheless semantically sound because they satisfy to the logical predicate.

3 Exploring the Definition

Intuitively, the function $\Lambda\alpha.\, \lambda x.\, \lambda y.\, x$ is parametric in α and should therefore satisfy the logical predicate. That is:

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

Let's walk through the reasoning to verify this.

$\Lambda\alpha.\, \lambda x.\, \lambda y.\, x \in [\forall\alpha.\, \alpha \to \alpha \to \alpha]$
iff $\lambda x.\, \lambda y.\, x \in [\![S \to S \to S]\!]$ for all S
iff $\lambda x.\, \lambda y.\, x \in [S \to S \to S]$ for all S
Now fix an arbitary S. We have to verify that
for all $v \in [S]$ we have $(\lambda x.\, \lambda y.\, x)\, v \in [\![S \to S]\!]$
So assume we have an arbitrary $v \in [S]$ (which, by definition, is true iff $v \in S$).
Since $(\lambda x.\, \lambda y.\, x)\, v \mapsto^* \lambda y.\, v$, we have to check $\lambda y.\, v \in [S \to S]$.
Again, pick an arbitrary $w \in S$.
We have to check that $(\lambda y.\, v)\, w \in [\![S]\!]$.
Since $(\lambda y.\, v)\, w \mapsto^* v$ we have to check that $v \in [S]$. That's exactly one of our assumptions.

Perhaps more interesting is the other direction: what can we deduce if we know $f \in [\forall\alpha.\alpha \to \alpha \to \alpha]$ but nothing else? This is not academic: we will show in **??** that every $\cdot \vdash v : \tau$ (for a closed τ) will satisfy $v \in [\tau]$. So typing implies membership in the unary logical relation, and typing can be established mechanically because it is decidable. Anyway, let's see what happens.

$f \in [\forall\alpha.\, \alpha \to \alpha \to \alpha]$	Assumption
$f[S_2] \in [\![S_2 \to S_2 \to S_2]\!]$ for $S_2 = \{v, w\}$ for arbitrary values v and w	By defn. of $[-]$
$f[S_2] \mapsto^* f_0$ and $f_0 \in [S_2 \to S_2 \to S_2]$	By definition of $[\![-]\!]$
$f_0\, v \in [\![S_2 \to S_2]\!]$ since $v \in S_2$	By defn. of $[-]$

$f_0\, v \mapsto^* f_1$ and $f_1 \in [S_2 \to S_2]$ By defn. of $[\![-]\!]$
$f_1\, w \mapsto^* u$ and $u \in [S_2]$ since $w \in S_2$ By defn. of $[-]$
$u \in \{v, w\}$ By defn. of S_2.

Putting this together we conclude

$$f\,[S_2]\, v\, w \mapsto^* f_0\, v\, w \mapsto^* f_1\, w \mapsto^* u \quad \text{for } u = v \text{ or } u = w$$

We see that f behaves like a first or second projection. As was remarked by a student in lecture, this does not quite imply that f must be extensionally equivalent to either $\Lambda\alpha.\, \lambda x.\, \lambda y.\, x$ or $\Lambda\alpha.\, \lambda x.\, \lambda y.\, y$ as I wrongly claimed. For this slightly stronger property we seem to need a binary logical relation as introduced in the next lecture.

4 The Fundamental Theorem

Theorem 1 (Fundamental Theorem of Unary Logical Relations, v1)
If $\cdot \vdash e : \tau$ *for* $\cdot \vdash \tau$ *type then* $e \in [\![\tau]\!]$.

This immediately implies termination by the definition of $e \in [\![\tau]\!]$. Just like in the previous lecture, we'd like to prove this by rule induction on the typing derivation, but will fail since the context will not remain empty. As before, we'd like to substitute for the context Γ in $\Gamma \vdash e : \tau$, but now this substitution also has to account for type variables, and their substitution by sets of values. This is trickier than before, because a type variable may occur in later type declarations.

Contexts	Γ	$::=$	$\cdot \mid \Gamma, x : \tau \mid \Gamma, \alpha$ *type*
Simultaneous substitution	η	$::=$	$\cdot \mid \eta, v/x \mid \eta, S/\alpha$

The definition of $\eta \in [\Gamma]$ is now slightly more complicated. As before, we assume S is a set of closed values.

$\eta \in [\cdot]$	iff	$\eta = (\cdot)$
$\eta \in [\Gamma', \alpha \textit{ type}]$	iff	$\eta = (\eta', S/\alpha)$ with $\eta' \in [\Gamma']$
$\eta \in [\Gamma', x : \tau]$	iff	$\eta = (\eta', v/x)$ with $\eta' \in [\Gamma']$ and $v \in [\eta'(\tau)]$

The reason we need to apply $\eta'(\tau)$ in the last clause is because τ may contain type variables declared in Γ'. We need to map them to their interpretation under η'. This gives us the more general form of the theorem. As a side remark, if we are interested in *dependent types* we also have the situation that types of later declarations may depend on earlier variables.

Theorem 2 (Fundamental Theorem of Unary Logical Relations, v2)
If $\Gamma \vdash e : \tau$ *and* $\eta \in [\Gamma]$ *then* $\eta(e) \in [\![\eta(\tau)]\!]$. *We presuppose here that* $\Gamma \vdash \tau$ *type.*

The main difference in comparison to the statement of the theorem from last lecture is that η must be applied to the type τ. Or, as indicated earlier, the free type variables in τ must be interpreted by η if they are written separately.

Proof: By rule induction on $\Gamma \vdash e : \tau$. We revisit one prior case and then show some new cases.

Case:

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau}\ \textsf{tp/app}$$

$\eta \in [\Gamma]$	Assumption
$\eta(e_1) \in \llbracket \eta(\tau_2 \to \tau) \rrbracket$	By ind. hyp.
$\eta(e_1) \mapsto^* v_1$ and $v_1 \in [\eta(\tau_2 \to \tau)]$	By defn. of $\llbracket - \rrbracket$
$\eta(e_2) \in \llbracket \eta(\tau_2) \rrbracket$	By ind. hyp.
$\eta(e_2) \mapsto^* v_2$ and $v_2 \in [\eta(\tau_2)]$	By defn. of $\llbracket - \rrbracket$
$\eta(\tau_2 \to \tau) = \eta(\tau_2) \to \eta(\tau)$	By defn. of $\eta(-)$
$v_1\, v_2 \in \llbracket \eta(\tau) \rrbracket$	By defn. of $[-]$
$\eta(e_1\, e_2) = \eta(e_1)\, \eta(e_2) \mapsto^* v_1\, v_2 \in \llbracket \eta(\tau) \rrbracket$	By properties of reduction
$\eta(e_1\, e_2) \in \llbracket \eta(\tau) \rrbracket$	By closure under reverse evaluation (extending Lemma 13.4)

Case:

$$\frac{\Gamma, \alpha\ \mathit{type} \vdash e_2 : \tau_2}{\Gamma \vdash \Lambda\alpha.\, e_2 : \forall\alpha.\, \tau_2}\ \mathsf{tp/tplam}$$

$\eta \in [\Gamma]$	Assumption
Let S be an arbitrary set of values	
$(\eta, S/\alpha) \in [\Gamma, \alpha\ \mathit{type}]$	By defn. of $[-]$ on contexts
$(\eta, S/\alpha)(e_2) \in \llbracket (\eta, S/\alpha)(\tau_2) \rrbracket$	By ind. hyp.
$(\eta(\Lambda\alpha.\, e_2))\, [S] \mapsto^* [\eta, S/\alpha](e_2)$	By rule and properties of substitution
$(\eta(\Lambda\alpha.\, e_2))\, [S] \in \llbracket (\eta, S/\alpha)(\tau_2) \rrbracket$	By closure under reverse evaluation
$(\eta(\Lambda\alpha.\, e_2))\, [S] \in \llbracket [S/\alpha](\eta, \alpha/\alpha)(\tau_2) \rrbracket$	By properties of substitution
$\eta(\Lambda\alpha.\, e_2) \in \llbracket \eta(\forall\alpha.\, \tau) \rrbracket$	By defn. of $[-]$ and defn. of substitution

Case:

$$\frac{\Gamma \vdash e_1 : \forall\alpha.\, \tau \quad \Gamma \vdash \sigma\ \mathit{type}}{\Gamma \vdash e_1\, [\sigma] : [\sigma/\alpha]\tau}\ \mathsf{tp/tpapp}$$

$\eta \in [\Gamma]$	Assumption
$\eta(e_1) \in \llbracket \eta(\forall\alpha.\, \tau) \rrbracket$	By ind. hyp.
$\eta(e_1) \mapsto^* v_1$ with $v_1 \in [\eta(\forall\alpha.\, \tau)]$	By defn. of $\llbracket - \rrbracket$
$v_1 \in [\forall\alpha.\, (\eta, \alpha/\alpha)(\tau)]$	By defn. of substitution
Choose $S = [\eta(\sigma)] = \{v \mid v \in [\eta(\sigma)]\}$	
$v_1[\eta(\sigma)] \in \llbracket [\eta, \eta(\sigma)/\alpha](\tau) \rrbracket$	By defn. of $[-]$
$v_1[\eta(\sigma)] \in \llbracket \eta([\sigma/\alpha]\tau) \rrbracket$	By substitution lemma (see below)
$\eta(e_1[\sigma]) = \eta(e_1)[\eta(\sigma)] \mapsto^* v_1[\eta(\sigma)] \in \llbracket \eta([\sigma/\alpha]\tau) \rrbracket$	By properties of evaluation
$\eta(e_1[\sigma]) \in \llbracket \eta([\sigma/\alpha]\tau) \rrbracket$	By closure under reverse evaluation

The key step in this case is that

$$\llbracket [\eta, \eta(\sigma)/\alpha](\tau) \rrbracket = \llbracket \eta([\sigma/\alpha]\tau) \rrbracket$$

which follows by induction on the structure of the type τ. When we reach α, we test membership in $\eta(\sigma)$ in both expressions. This is a consequence of the way that $[\tau]$ and $\llbracket \tau \rrbracket$ are defined by induction over the structure of the type τ.

□

From what we have done so far, we easily conclude termination.

Corollary 3 (Termination) *If $\cdot \vdash e : \tau$ then $e \mapsto^* v$ for some value v.*

Proof: By the fundamental theorem for $\Gamma = (\cdot)$, $\eta = (\cdot)$ we have $e \in \llbracket \tau \rrbracket$. By definition, this means that $e \mapsto^* v$ for some value v. (Also $v \in [\tau]$, but this property is irrelevant to this corollary.) □

Exercises

Exercise 1 Consider the relation $\tau \leq \sigma$ as defined for the fragment of the language without polymorphism or recursive types. Recall that on this fragment its definition is *inductive* rather than coinductive (as is natural in the presence of recursive types).

(i) Extend the *syntactic definition of subtyping* to parametric polymorphism. You should recall its defining properties to make sure they continue to hold.

(ii) Give a corresponding *semantic definition of subtyping*, either referencing $[\![-]\!]$ and $[-]$ or defining it in a way that reflects the spirit of the logical relation from this lecture.

(iii) Prove the *soundness* of the syntactic rules with respect to their semantic interpretation. This should be in some way analogous to proving the fundamental theorem which states that syntactic typing ($e : \tau$) implies semantic typing ($e \in [\![\tau]\!]$).

Exercise 2 In this problem we explore the definition of a unary logical relation for existential types.

(i) Give a definition of the unary logical relation for $v \in [\exists\alpha.\, \tau]$.

(ii) Show the case for the constructor for $\exists\alpha.\, \tau$ in the proof of the fundamental theorem.

(iii) Show the case for the destructor for $\exists\alpha.\, \tau$ in the proof of the fundamental theorem.

In the proofs, you may assume simple properties of substitution and evaluation, as well as lemmas such as closure under reverse evaluation (essentially: the properties we used in our proof for parametric polymorphism). If you need additional lemmas or properties, please state them.

References

Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971.

Jean-Yves Girard. *Proofs and Types*. Cambridge University Press, 1989. Translated and with appendices by Paul Taylor and Yves Lafont.

John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.

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