Practical Semantics

Jonathan Sterling

Updated October 21, 2022

Contents

Chapter 1

Logical relations *are* compositional

Abstract. Logical relations arguments are usually phrased non-compositionally, so as to appear artificially distant from denotational semantics. We describe how a small amount of refactoring and notational adjustment exhibits standard logical relations arguments as an instance of denotational semantics. In this chapter, we work in a provisional and unstructured way, avoiding category theory at the cost of some precision.

1.1 Our running example: hereditary termination

We begin with a completely conventional example that shows *termination* of operational semantics on closed terms in typed λ-calculus, roughly following the explanation of Harper [9]. By typed λ-calculus, we mean a language whose types are closed under a base type bool, function types $\sigma \Rightarrow \tau$, and product types $\sigma * \tau$. Untyped terms are defined by the following binding grammar:

$$M ::\equiv x \mid \lambda x.M \mid M \cdot N \mid \langle M, N \rangle \mid M.1 \mid M.2 \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{if}\, L\, M\, N$$

Typed terms in contexts Γ are defined by an indexed inductive definition:

$$\frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau} \qquad \frac{\Gamma \vdash M : \sigma \Rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M \cdot N : \tau}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma * \tau} \qquad \frac{\Gamma \vdash M : \sigma * \tau}{\Gamma \vdash M.1 : \sigma} \qquad \frac{\Gamma \vdash M : \sigma * \tau}{\Gamma \vdash M.2 : \tau} \qquad \frac{}{\Gamma \vdash \mathsf{tt} : \mathsf{bool}}$$

$$\frac{}{\Gamma \vdash \mathsf{ff} : \mathsf{bool}} \qquad \frac{\Gamma \vdash L : \mathsf{bool} \quad \Gamma \vdash M : \tau \quad \Gamma \vdash N : \tau}{\Gamma \vdash \mathsf{if}\, L\, M\, N : \tau}$$

Remark 1. We will not belabor things like variable freshness; we simply assume that the reader is comfortable with one of the dozens of methods to ensure that operations involving syntax and variables is well-defined.

Lemma 2 (Substitution). *If* $\Gamma, x : \sigma \vdash M : \tau$ *and* $\Gamma \vdash N : \sigma$, *then* $\Gamma \vdash M[N/x] : \tau$.

In order to keep close to what is familiar, we do not yet impose any equational theory on our terms. Instead, we will define *structural operational semantics* on untyped closed terms through the following inductive definition:

$$\frac{}{\lambda x.M\ \mathit{val}} \qquad \frac{}{\mathsf{tt}\ \mathit{val}} \qquad \frac{}{\langle M,N\rangle\ \mathit{val}} \qquad \frac{}{\mathsf{ff}\ \mathit{val}} \qquad \frac{}{(\lambda x.M)N \mapsto M[N/x]}$$

$$\frac{}{\langle M,N\rangle.1 \mapsto M} \qquad \frac{}{\langle M,N\rangle.2 \mapsto N} \qquad \frac{}{\mathsf{if}\,\mathsf{tt}\,M\,N \mapsto M} \qquad \frac{}{\mathsf{if}\,\mathsf{ff}\,M\,N \mapsto N}$$

$$\frac{M \mapsto M'}{M\cdot N \mapsto M'\cdot N} \qquad \frac{M \mapsto M'}{M.1 \mapsto M'.1} \qquad \frac{M \mapsto M'}{M.2 \mapsto M'.2} \qquad \frac{L \mapsto L'}{\mathsf{if}\,L\,M\,N \mapsto \mathsf{if}\,L'\,M\,N}$$

Remark 3. We have given a call-by-name operational semantics above for the sake of simplicity, but it doesn't matter. The rest of our discussion would also work with a call-by-value operational semantics.

Definition 4. We will write $\mapsto^*$ for the reflexive–transitive closure of the $\mapsto$ relation on untyped closed terms.

The goal of our running example is to prove the following *termination* theorem for *typed* closed terms.

Proposition 5 (Termination)**.** *For any well-typed closed term* $\cdot \vdash M : \mathsf{bool}$*, there exists* $\cdot \vdash N : \mathsf{bool}$ *such that* N *val and* $M \mapsto^* N$*.*

It is commonly understood that you cannot simply prove this theorem by induction on M because the induction hypothesis in the case of the λ-abstraction is too weak (see Harper [9] for more discussion). The search for a strong induction hypothesis with which to prove Proposition 5 culminated in Tait's invention of the *method of computability* [19], which is referred to more commonly as *logical relations*. In this tutorial we assume basic familiarity with conventional accounts of logical relations, and our goal is to explain a more structured (and therefore superior) way to think about them.

We quickly outline the way that a logical relations argument is conventionally formulated to establish Proposition 5. The main input of logical relations is to introduce a generalized notion of termination that coincides with ordinary termination at base types, but is customized at higher types. This notion is called *hereditary termination*, and is defined by induction on the types.

Definition 6. We will write $(\Gamma \vdash \sigma)$ for the set $\{M \mid \Gamma \vdash M : \sigma\}$ of well-typed terms of type σ in context Γ.

We define by induction on types τ a family of predicates $\mathbf{isHT}_\tau M$ on closed terms $\cdot \vdash M : \tau$ to mean that "M is a hereditarily terminating element of type τ":

$$\begin{aligned}
\mathbf{isHT}_{\mathsf{bool}} M &\iff M \mapsto^* \mathsf{tt} \vee M \mapsto^* \mathsf{ff} \\
\mathbf{isHT}_{\sigma\Rightarrow\tau} M &\iff \forall N \in (\cdot \vdash \sigma).\ \mathbf{isHT}_\sigma N \Rightarrow \mathbf{isHT}_\tau (M\cdot N) \\
\mathbf{isHT}_{\sigma*\tau} M &\iff \mathbf{isHT}_\sigma (M.1) \wedge \mathbf{isHT}_\tau (M.2)
\end{aligned}$$

We must extend our notion of hereditary termination to contexts and *environments*.

Definition 7. An *environment* for a context Γ is a closing substitution, *i.e.* an element of the indexed product $\prod_{(x:\sigma)\in\Gamma}(\cdot \vdash \sigma)$. We will write $(\cdot \vdash \Gamma)$ for the set of environments for Γ, and for any term $\Gamma \vdash M : \tau$, we will write $\cdot \vdash M[\gamma] : \tau$ for the closed term obtained by substituting the components of γ for all the free variables of M.

Hereditary termination of environments $\gamma \in (\cdot \vdash \Gamma)$ is defined as follows:

$$\mathbf{isHT}_\Gamma \gamma :\equiv \textstyle\prod_{(x:\sigma)\in\Gamma} \mathbf{isHT}_\sigma(\gamma x)$$

We will need the following technical lemma, usually called either *closure under converse evaluation* or *closure under head expansion*.

Lemma 8 (Converse evaluation)**.** *If $M \mapsto M'$ and $\mathbf{isHT}_\tau M'$ then $\mathbf{isHT}_\tau M$.*

Then Proposition 5 is proved as a corollary to the *fundamental theorem of logical relations* which is stated below.

Theorem 9 (Fundamental Theorem)**.** *For any $\Gamma \vdash M : \tau$ and $\gamma \in (\cdot \vdash \Gamma)$, if $\mathbf{isHT}_\Gamma \gamma$ then we have $\mathbf{isHT}_\tau M[\gamma]$.*

Proof. This follows by induction on the derivation of $\Gamma \vdash M : \tau$; we display a representative case below:

> In the case of $\lambda x.M$, we need to show that $\lambda x.M[\gamma]$ is hereditarily terminating when γ is. Fixing a hereditarily terminating element $\cdot \vdash N : \sigma$ of the domain, we must verify that $(\lambda x.M[\gamma]) \cdot N$ is a hereditarily terminating element of the codomain. Because $(\lambda x.M[\gamma]) \cdot N \mapsto M[\gamma, x \hookrightarrow n]$, by converse evaluation (Lemma 8) it suffices to check that $M[\gamma, x \hookrightarrow n]$ is hereditarily terminating. But this is our induction hypothesis. □

The termination property follows now from the fundamental theorem.

Proposition 5 (Termination)**.** *For any well-typed closed term $\cdot \vdash M : \mathsf{bool}$, there exists $\cdot \vdash N : \mathsf{bool}$ such that N val and $M \mapsto^* N$.*

Proof. By the fundamental theorem (Theorem 9), we have $\mathbf{isHT}_{\mathsf{bool}} M$ and thus $M \mapsto^* \mathsf{tt}$ or $M \mapsto^* \mathsf{ff}$; we have both tt *val* and $\cdot \vdash \mathsf{tt} : \mathsf{bool}$, and likewise for ff. □

1.2 A fully compositional logical relation for termination

When they see the definition of hereditary termination (reproduced below), some practitioners think of it as defining an **interpretation** of each term M as a proposition denoting the property that M is hereditarily terminating:

$$\begin{aligned}
\mathbf{isHT}_{\mathsf{bool}} M &\Longleftrightarrow M \mapsto^* \mathsf{tt} \vee M \mapsto^* \mathsf{ff} \\
\mathbf{isHT}_{\sigma \Rightarrow \tau} M &\Longleftrightarrow \forall N \in (\cdot \vdash \sigma).\ \mathbf{isHT}_\sigma N \Rightarrow \mathbf{isHT}_\tau (M \cdot N) \\
\mathbf{isHT}_{\sigma * \tau} M &\Longleftrightarrow \mathbf{isHT}_\sigma (M.1) \wedge \mathbf{isHT}_\tau (M.2)
\end{aligned}$$

If $\mathbf{isHT}_\tau M$ is an "interpretation" of M, then it is certainly not a compositional one. A compositional interpretation defines the meaning of a composite object in terms of the meanings of its constitutents, but here the "meaning" of a term M is defined not in terms of the meanings of its immediate subterms but rather in terms of evaluation and the meanings of certain *non-subterms* of M.

The way out of this quandary is, first, to recognize that the definition of hereditary termination is **not** an interpretation of terms, but rather an interpretation of types. The interpretation of *terms* in a logical relation must, then, lie within the proof of the fundamental theorem.

1.2.1 Hereditary termination is compositional on types

It requires only superficial notational changes to see the potential of the logical relation to be compositional. First we "curry" the definition of hereditary termination to define for each type τ a subset $\mathbf{HT}_\tau \subseteq (\cdot \vdash \tau)$ which is meant to be spanned by hereditarily terminating elements.

$$\begin{aligned}
\mathbf{HT}_{\mathsf{bool}} &:\equiv \{M \in (\cdot \vdash \mathsf{bool}) \mid M \mapsto^* \mathsf{tt} \lor M \mapsto^* \mathsf{ff}\} \\
\mathbf{HT}_{\sigma \Rightarrow \tau} &:\equiv \{M \in (\cdot \vdash \sigma \Rightarrow \tau) \mid \forall N \in \mathbf{HT}_\sigma.\ M \cdot N \in \mathbf{HT}_\tau\} \\
\mathbf{HT}_{\sigma * \tau} &:\equiv \{M \in (\cdot \vdash \sigma * \tau) \mid M.1 \in \mathbf{HT}_\sigma \land M.2 \in \mathbf{HT}_\tau\}
\end{aligned}$$

It is now clear enough that the definition of hereditary termination *is* compositional when viewed as an interpretation of types. So what of the interpretation of terms? We must admit that it has not been made precise what it even means to interpret terms in a logical relation, so the first step to answering this question must be to *rephrase* the statement of the fundamental theorem (Theorem 9) in terms of an interpretation.

1.2.2 Logical relations as a compositional interpretation

We will now specify, in an admittedly still ad hoc way, the logical relation and its fundamental theorem in terms of an *interpretation*. The first step is to give a *specification* of what kinds of objects the interpretation will be valued in, for each syntactic sort of the typed λ-calculus.

Specification 10. We begin by specifying what kinds of things contexts and types shall be interpreted by, rephrasing our existing specification.

1. We shall interpret a context Γ as a subset $[\![\Gamma]\!] \subseteq (\cdot \vdash \Gamma)$ of the collection of environments for Γ.

2. We shall likewise interpet a type τ as a subset $[\![\tau]\!] \subseteq (\cdot \vdash \tau)$ of the collection of closed terms of type τ.

In the prior sections, $[\![\Gamma]\!]$ and $[\![\tau]\!]$ were written $\mathbf{HT}_\Gamma$ and $\mathbf{HT}_\tau$ respectively. What is new is that we shall interpret terms as *functions* satisfying a particular property, exactly as in conventional denotational semantics:

3. We shall interpret a term $\Gamma \vdash M : \tau$ as a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ such that for each $\gamma \in [\![\Gamma]\!]$ we have $[\![M]\!]\gamma = M[\gamma]$.

The key observation here is that the interpretation function $[\![-]\!]$ for terms exists *if and only if* the fundamental theorem of logical relations holds. This is the sense in which the fundamental theorem is the interpretation of terms.

1.2.2.1 An almost-compositional interpretation

Unfortunately, we will see that the interpretation of terms suggested by our proof of the fundamental lemma is not entirely compositional, but it is very close. Let's see what goes wrong by writing out part of the definition:

$$\begin{aligned} [\![\Gamma \vdash x : \sigma]\!]\gamma &= \gamma x \\ [\![\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau]\!]\gamma &= \lambda z.[\![\Gamma, x : \sigma \vdash M : \tau]\!](\gamma, x \mapsto z) \end{aligned}$$

Pausing here, we note that it is part of our proof obligation to show that $\lambda z.[\![M]\!](\gamma, x \hookrightarrow z)$ is an element of $[\![\sigma \Rightarrow \tau]\!]$. Recalling that we have provisionally defined $[\![\rho]\!] :\equiv \mathbf{HT}_\rho$, we unfold the definition of hereditary termination at the function space. It follows that we must prove for any $N \in [\![\sigma]\!]$ that $(\lambda z.[\![M]\!](\gamma, x \hookrightarrow z)) \cdot N$ lies in $[\![\tau]\!]$. It so happens that the former term reduces to $[\![M]\!](\gamma, x \hookrightarrow N)$ which lies in $[\![\tau]\!]$ by the induction hypothesis for $[\![M]\!]$.

Therefore, **if we knew** that $[\![\tau]\!]$ satisfied the converse evaluation lemma, then we would be able to deduce that $\lambda z.[\![M]\!](\gamma, x \hookrightarrow z)$ lies in $[\![\tau]\!]$. Unfortunately, this fact is not one of the induction hypotheses induced by the subterms L, M, and N. Thus our definition lies on the borderline of compositionality — we could apply Lemma 8, but in a **compositional** interpretation we prefer not to rely on the fact that some semantic object is definable, *i.e.* lies in the image of the interpretation map. We can, however, make our interpretation more explicitly compositional by changing the specification of the interpretation of *types* to incorporate converse evaluation.

Remark 11. The change we suggest here is only cosmetic in the case of the simply typed λ-calculus, but it becomes non-optional when extending to polymorphic λ-calculus.

1.2.2.2 Specifying a compositional interpretation

Definition 12. A subset $S \subseteq (\cdot \vdash \tau)$ is called *admissible* when it is closed under converse evaluation in the sense that for all $\cdot \vdash M : \tau$ and $M' \in S$ such that $M \mapsto M'$, we have $M \in S$.

Specification 13 (Corrected)**.** Now we amend our specification as follows:

1. We shall interpret a context Γ as a subset $[\![\Gamma]\!] \subseteq (\cdot \vdash \Gamma)$ of the collection of environments for Γ.

2. We shall interpet a type τ as an **admissible** subset $[\![\tau]\!] \subseteq (\cdot \vdash \tau)$ of the collection of closed terms of type τ.

3. We shall interpret a term $\Gamma \vdash M : \tau$ as a function $[\![M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ such that for each $\gamma \in [\![\Gamma]\!]$ we have $[\![M]\!]\gamma = M[\gamma]$.

With the revised specification, the interpretation of terms is now fully compositional as promised.

Remark 14. Our situation is somewhat similar to the non-compositionality of the naïve domain semantics of call-by-name λ-calculus, in which types are interpreted as cpos and terms are interpreted as continuous maps. The interpretation can in fact be carried out, using an auxiliary induction to equip every cpo of the form $[\![\tau]\!]$ with a bottom element, but such an interpretation is not compositional [11]. Compositionality is restored by instead interpreting types as *pointed cpos*; the auxiliary induction is then folded into the very definition of the interpretation of types. Category theorists would say that the original interpretation took place "in the wrong category", or that it was the shadow of the good interpretation viewed from inside the wrong category.

1.2.2.3 The full logical relation

We now summarize with the full logical relation specified by Specification 13 as a fully compositional interpretation. We recall that Specification 13 fixes the types of the interpretation as follows:

$$\begin{aligned}
[\![\Gamma]\!] &\subseteq (\cdot \vdash \Gamma)\\
[\![\tau]\!] &\in \{S \subseteq (\cdot \vdash \tau) \mid S \textit{ is admissible}\}\\
[\![\Gamma \vdash M : \tau]\!] &\in \{f : [\![\Gamma]\!] \to [\![\tau]\!] \mid \forall \gamma \in [\![\Gamma]\!].\, f\gamma = M[\gamma]\}
\end{aligned}$$

We now implement the specification above compositionally, first by interpreting contexts and types:

$$\begin{aligned}
[\![\Gamma]\!] &= \textstyle\prod_{(x:\sigma)\in\Gamma}[\![\sigma]\!]\\
[\![\mathsf{bool}]\!] &:\equiv \{M \in (\cdot \vdash \mathsf{bool}) \mid M \mapsto^{*} \mathsf{tt} \vee M \mapsto^{*} \mathsf{ff}\}\\
[\![\sigma \Rightarrow \tau]\!] &:\equiv \{M \in (\cdot \vdash \sigma \Rightarrow \tau) \mid \forall N \in [\![\sigma]\!].\, M \cdot N \in [\![\tau]\!]\}\\
[\![\sigma * \tau]\!] &:\equiv \{M \in (\cdot \vdash \sigma \Rightarrow \tau) \mid M.1 \in [\![\sigma]\!] \wedge M.2 \in [\![\tau]\!]\}
\end{aligned}$$

The interpretation of types above implicitly carries the inductive proof that each of the chosen subsets is admissible; in this sense, we have replaced Lemma 8 by a more structured definition. The compositional interpretation of terms is given below:

$$\begin{aligned}
[\![\Gamma \vdash M : \tau]\!] &: [\![\Gamma]\!] \to [\![\tau]\!]\\
[\![\Gamma \vdash x : \sigma]\!]\gamma &= \gamma x\\
[\![\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau]\!]\gamma &= \lambda z.[\![\Gamma, x : \sigma \vdash M : \tau]\!](\gamma, x \mapsto z)\\
[\![\Gamma \vdash M \cdot N : \tau]\!]\gamma &= [\![\Gamma \vdash M : \sigma \Rightarrow \tau]\!]\gamma \cdot [\![\Gamma \vdash N : \sigma]\!]\gamma\\
[\![\Gamma \vdash \langle M, N\rangle : \sigma * \tau]\!]\gamma &= \langle [\![\Gamma \vdash M : \sigma]\!]\gamma, [\![\Gamma \vdash N : \tau]\!]\gamma\rangle\\
[\![\Gamma \vdash M.1 : \sigma]\!]\gamma &= ([\![\Gamma \vdash M : \sigma * \tau]\!]\gamma).1\\
[\![\Gamma \vdash M.2 : \tau]\!]\gamma &= ([\![\Gamma \vdash M : \sigma * \tau]\!]\gamma).2\\
[\![\Gamma \vdash \mathsf{tt} : \mathsf{bool}]\!]\gamma &= \mathsf{tt}\\
[\![\Gamma \vdash \mathsf{ff} : \mathsf{bool}]\!]\gamma &= \mathsf{ff}\\
[\![\Gamma \vdash \mathsf{if}\, L\, M\, N : \tau]\!]\gamma &= \mathsf{if}\; ([\![\Gamma \vdash L : \mathsf{bool}]\!]\gamma)\; ([\![\Gamma \vdash M : \tau]\!]\gamma)\; ([\![\Gamma \vdash N : \tau]\!]\gamma)
\end{aligned}$$

The well-definedness of these clauses appeals to the fact each $[\![\tau]\!]$ is defined to be an *admissible* subset of $(\cdot \vdash \tau)$.

Remark 15 (Where did the fundamental theorem go?)**.** The recursive definition of $[\![M]\!]$ is an alternative, compositional, presentation of the fundamental theorem. The inductive structure of the fundamental theorem is "packed" into the recursive structure of the interpretation; the reasoning involved in Theorem 9 is now part of proof that each clause if $[\![M]\!]$ is well-typed.

We may now deduce the termination theorem.

Proposition 5 (Termination)**.** *For any well-typed closed term* $\cdot \vdash M : \mathsf{bool}$*, there exists* $\cdot \vdash N : \mathsf{bool}$ *such that* N *val and* $M \mapsto^* N$*.*

Proof. Applying the interpretation, we have $[\![\cdot \vdash M : \mathsf{bool}]\!]$ which is (by Specification 13) a function $[\![M]\!] : [\![\cdot]\!] \to [\![\mathsf{bool}]\!]$ such that $[\![M]\!]\gamma = M[\gamma]$ for any $\gamma \in [\![\cdot]\!]$. Choosing γ to be the empty environment ϵ, we have $[\![M]\!]\epsilon = M$. Thus we have $M \in [\![\mathsf{bool}]\!]$, from which we deduce that either $M \mapsto^* \mathsf{tt}$ or $M \mapsto^* \mathsf{ff}$. $\square$

1.2.2.4 Discussion: toward proof relevant logical relations

Inspecting the interpretation of terms in Section 1.2.2.3, it appears that $[\![\Gamma \vdash M : \tau]\!]\gamma$ is literally the substitution action. This is almost true, but note that these two operations have different types: the first is a function from $[\![\Gamma]\!]$ to $[\![\tau]\!]$ and the second is a function from $(\cdot \vdash \Gamma)$ to $(\cdot \vdash \tau)$. It is therefore better to say that $[\![\Gamma \vdash M : \tau]\!]$ is a *refinement* of the substitution action, where $[\![\Gamma]\!]$ and $[\![\tau]\!]$ are thought of as refinements of $(\cdot \vdash \Gamma)$ and $(\cdot \vdash \tau)$ respectively.

That $[\![\Gamma \vdash M : \tau]\!]$ is "tracked" by the substitution action in this sense is exactly the correctness condition on the denotations of terms imposed in Specification 13. It is reasonable, therefore, to ask: why do we bother with such an artificial "interpretation" that cannot help but be given by a *property* of substitution in relation to hereditary termination?

One answer is that the interpretation-style formulation is compositional on terms in contrast to the usual formulation in terms of the fundamental theorem of logical relations, but is this compositionality really that useful? We will see later on that there are several reasons to favor the compositional approach that hinge on the generalization of ordinary logical relations to ***proof relevant* logical relations**, *i.e.* logical relations where $[\![\tau]\!]$ expresses not a subset of $(\cdot \vdash \tau)$ but rather a family of sets of "computability data" indexed in $(\cdot \vdash \tau)$. In these scenarios, there is no analogue to the "fundamental theorem" and the only move possible is to define a compositional interpretation function that equips syntactic terms with their computability data.

The impact of proof relevance Many real-world logical relations are proof-relevant [4, 17, 2, 16]. One motivation of proof-relevant logical relations is to avoid certain onerous admissibility conditions, as in the work of Benton, Hofmann, and Nigam [4], thus leading to a simpler alternative to biorthogonality/$\top\top$-closure.

Another motivation for proof relevance, exploited by Fiore [7], is to enable the formulation of normalization arguments with respect to *equivalence classes of terms* rather than raw terms, which can in some cases decrease by a factor of ten the number of pages needed to deduce the main result. It turns out that this insight of *op. cit.* concerning proof relevance can be used to entirely *avoid* the technical need for operational semantics in many applications of logical relations in both programming language theory and type theory. Thus although operational semantics are no doubt important, many of the problems that we had previously *only* known how to solve using operational methods can now be approached more directly through proof-relevant logical relations.

Along these lines, an important application of proof relevance is to extend the applicability of Girard's method of candidates to languages that have not only polymorphic types like $\forall_{\alpha:\mathbf{type}} A$, but also **type universes** as in the work of Atkey [3] and Sterling and Harper [18]. Prior to the advent of proof relevant logical relations, type universes had been accountable in logical relations *only* by using the very brittle and highly subtle to verify methods of Allen [1] and Harper [8] — methods that are not compatible with the parametric reasoning offered by Girard's type candidates, and are thus inapplicable to most goals within programming language theory.

One major application in this line has been the extension of the method of candidates (and thus parametric reasoning) to ML module systems by Sterling and Harper [18], where the core language contains a type universe; the method of *op. cit.* is a more direct alternative to the reduction of ML modules to System F-style polymorphism [15], which also enables parametric reasoning for modules in a highly *indirect* fashion. This application of proof relevance to type candidates with universes has been exploited by many other authors in recent times, and most of the non-trivial syntactic results in dependent type theory of the last decade have relied heavily on it.

Chapter 2

Categorical structure of λ-calculus

Abstract. We introduce the (β, $\beta\eta$) equational theories of the typed λ-calculus, and relate them to the categorical notion of (weakly) cartesian closed categories. Having established this correspondence, we show how to use categorical methods to deduce the termination theorem of Chapter 1 from much more generally applicable results.

2.1 Equational presentation of the λ-calculus

So far we have presented the λ-calculus according to the American school consensus, divided into statics (typing rules) and dynamics (structural operational semantics on closed terms). The first step in moving toward category theory is to impose a *congruence* on the typed open terms that we shall think of as an extension of the statics. We will see not only that we can use this extended statics to prove non-trivial results about the dynamics, but also that the statics construed in this style poses an *viable alternative* to the introduction of dynamic semantics in the first place.

The purpose of imposing a congruence is to be able to work *semantically* in a variety of models at a high level of abstraction; our goal is to many study different models of λ-calculus, and use the rich variety of potential models to prove theorems about the actual models we care about (*e.g.* the term model, or the Scott model, or a logical relations model, *etc.*).

2.1.1 Introducing the β-congruence

One potential congruence that we could choose is the contextual/observational congruence, but this is not a viable choice as it rules out all models except the fully abstract ones. We will instead choose a congruence that is inspired more directly by operational semantics, namely the β-congruence $\equiv_\beta$ which we define to be the *smallest* congruence closed under the following rules:

$$\frac{\Gamma, x : \sigma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (\lambda x.M) \cdot N \equiv_\beta M[N/x] : \tau} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle.1 \equiv_\beta M : \sigma}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle.2 \equiv_\beta N : \tau} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \mathsf{if}\,\mathsf{tt}\,M\,N \equiv_\beta M : \tau} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \mathsf{if}\,\mathsf{ff}\,M\,N \equiv_\beta N : \tau}$$

That $\equiv_\beta$ is a congruence means that is an equivalence relation *and* that it is preserved by all operators of the λ-calculus in the sense of the following representative congruence rules:

$$\frac{\Gamma, x : \sigma \vdash M, M' : \tau \qquad \Gamma, x : \sigma \vdash M \equiv_\beta M' : \tau}{\Gamma \vdash \lambda x.M \equiv \lambda x.M' : \sigma \Rightarrow \tau}$$

$$\frac{\Gamma \vdash M, M' : \sigma \Rightarrow \tau \qquad \Gamma \vdash N, N' : \sigma \qquad \Gamma \vdash M \equiv_\beta M' : \sigma \Rightarrow \tau \qquad \Gamma \vdash N \equiv_\beta N' : \sigma}{\Gamma \vdash M \cdot N \equiv_\beta M' \cdot N' : \tau}$$

2.1.2 Introducing the $\beta\eta$-congruence

Semanticists tend to prefer a coarser equational theory that adds certain *extensionality* or *uniqueness* principles called η-laws. The practical advantage of these laws is twofold:

1. they provide a powerful tool for equational reasoning about programs;
2. they endow the type connectives with categorical universal properties, which in some cases significantly simplifies metatheoretic proofs.

Remark 16 (Why not η?)**.** The main disadvantage of η-laws is that they do not hold in the presence of side effects; for instance, it is not the case in OCaml that a function M necessarily behaves the same as `fun` $x \rightarrow M\ x$. This apparent disadvantage of η-laws is, however, based on a misconception: the full η-laws always hold in the presence of effects *if* effects are decomposed using either a monad [13] or the structure of call-by-push-value [11].

Indeed, the function of both monads and call-by-push-value is to eliminate the concept of evaluation order (and operational semantics) and replace it with type structure (and equational theories). Returning to the OCaml example, it of course remains the case that `fun` $x \rightarrow M\ x$ may be observably different from M, but that is because the semantic interpretation of the function space in OCaml-like languages involves an ambient monad *in addition to* the semantic function space (which satisfies the η-law).

Although we generally advocate for η-equivalence along the lines of Remark 16, in this chapter we will still favor the β-congruence because one of our goals is to make contact with conventional structural operational semantics. The rules of the η-congruence for function and product types extends the prior rules by the following ones:

$$\frac{\Gamma \vdash M, M' : \sigma \Rightarrow \tau \qquad \Gamma, x : \sigma \vdash M \cdot x \equiv_{\beta\eta} M' \cdot x : \tau}{\Gamma \vdash M \equiv_{\beta\eta} M' : \sigma \Rightarrow \tau}$$

$$\frac{\Gamma \vdash M, M' : \sigma * \tau \qquad \Gamma \vdash M.1 \equiv_{\beta\eta} M'.1 : \sigma \qquad \Gamma \vdash M.2 \equiv_{\beta\eta} M'.2 : \tau}{\Gamma \vdash M \equiv_{\beta\eta} M' : \sigma * \tau}$$

Exercise 17. Check for yourself that the rules we have given above are interderivable with the following alternative rules:

$$\frac{\Gamma \vdash M : \sigma \Rightarrow \tau}{\Gamma \vdash M \equiv_{\beta\eta} \lambda x.M \cdot x : \sigma \Rightarrow \tau} \qquad \frac{\Gamma \vdash M : \sigma * \tau}{\Gamma \vdash M \equiv_{\beta\eta} \langle M.1, M.2 \rangle : \sigma * \tau}$$

Thus it is only a matter of taste whether you choose these rules or the others.

2.1.3 The extended $\beta\eta$-congruence: extensionality for booleans

We did not above impose any extensionality rule for the booleans; such rules tend to be true in semantics, but they are often impractical to for machines to decide, and they also complicate the proof of normalization. Nonetheless, we introduce a further-extended congruence $\equiv^{+}_{\beta\eta}$ to include the following rule that characterizes functions out of the booleans by their input-output behavior:

$$\frac{\Gamma, x : \mathsf{bool} \vdash M, M' : \tau \qquad \Gamma \vdash M[\mathsf{tt}/x] \equiv^{+}_{\beta\eta} M'[\mathsf{tt}/x] : \tau \qquad \Gamma \vdash M[\mathsf{ff}/x] \equiv^{+}_{\beta\eta} N'[\mathsf{ff}/x] : \tau}{\Gamma, x : \mathsf{bool} \vdash M \equiv^{+}_{\beta\eta} M' : \tau}$$

Exercise 18. Check that the rule above is interderivable with the following alternative rule:

$$\frac{\Gamma, x : \mathsf{bool} \vdash M : \tau}{\Gamma, x : \mathsf{bool} \vdash M \equiv^{+}_{\beta\eta} \mathsf{if}\, x\, M[\mathsf{tt}/x]\, M[\mathsf{ff}/x] : \tau}$$

Remark 19 (Common mistake)**.** Note that the following rule is strictly weaker than extensionality law above:

$$\frac{\Gamma \vdash N : \mathsf{bool}}{\Gamma \vdash N \equiv^{+}_{\beta\eta} \mathsf{if}\, N\, \mathsf{tt}\, \mathsf{ff} : \mathsf{bool}}$$

2.1.4 Comparing the β-congruence with operational semantics

Because $\equiv_\beta$ allows reductions to occur under binders and in any order (because it is a congruence), it is natural to wonder, for instance, whether $\cdot \vdash u \equiv_\beta \mathsf{tt} : \mathsf{bool}$ implies that $u \mapsto^* \mathsf{tt}$ in operational semantics. This as well as many stronger results can be proved using operationally-based logical relations, but one of our goals in this chapter is to show how to *avoid* operationally-based logical relations in favor of the considerably more modular categorical logical relations.

The following *standardization lemma* of Plotkin [14] expresses the *computational adequacy* of the equational theory relative to operational semantics.[1]

[1]Thanks to Sam Tobin-Hochstadt for alerting me to this proof.

Lemma 20 (Standardization). *If V is a value and $\cdot \vdash M \equiv_\beta V : \mathsf{bool}$, then $M \mapsto^* V$.*

Lemma 20 is proved by straightforward syntactic (inductive) methods, employing a notion of *standard reduction sequence*. The proof of standardization is presented in more modern form by Felleisen, Findler, and Flatt [6] in terms of evaluation contexts.

Remark 21. The importance of the standardization lemma for us is that it allows us to deduce theorems about operational semantics by proving much simpler theorems about the equational theory.

2.2 Categorical preliminaries

In this section, we outline some of the basic machinery of category theory that we will need. The purpose of this section is not to teach these results and their proofs in depth, but instead to outline the basic definitions and results that we employ and provide a guide for further exploration. We have arranged things such that the reader can *black-box* the results mentioned in this section.

Remark 22 (Advice to the reader)**.** We advise the reader to roll up their sleeves and push through the definitions in this section; they may not seem motivated yet, but patience tends to be rewarded.

Remark 23. We will avoid belaboring matters of size here; we will make various size assumptions to ensure that our statements are *correct*, but the reader who is not familiar with size should feel free to simply ignore every mention of "small" or "locally small" or "large".

2.2.1 Cartesian products

In this section, we will describe *cartesian products* in a category. These are a categorical version of the structure of *product types* with both β and η-laws. Although we are trying to study a result about λ-calculus with β-reduction only, our proofs will make use of the theory of cartesian products; indeed, we will soon show that the "weak" product types without the η-law can be described most succinctly in terms of the cartesian product.

Definition 24 (Spans)**.** Let A, B be two objects of $\mathcal{C}$; a *span* between A and B is defined to be a diagram of the form $A \xleftarrow{p_A} P \xrightarrow{p_B} B$. Given another span $A \xleftarrow{q_A} Q \xrightarrow{q_B} B$, a *morphism* of spans between A and B from the former to the latter is defined to be a morphism $Q \xrightarrow{h} P$ that preserves the structure of the span in the sense that the following diagram commutes:

$$\begin{tikzcd}
 & Q \arrow[dl, "q_A"'] \arrow[dr, "q_B"] \arrow[dd, "h"] & \\
A & & B \\
 & P \arrow[ul, "p_A"] \arrow[ur, "p_B"'] &
\end{tikzcd}$$

Definition 25 (Cartesian product spans)**.** A span $\mathbf{P}$ between A and B is called a *cartesian product span* when for any other span $\mathbf{Q}$ between A and B, there exists a *unique* morphism $\mathbf{Q} \to \mathbf{P}$ of spans between A and B. (This condition is the universal property of the product span.)

In the scenario of Definition 25, the apex of a product span between A and B is often written $A \times B$; the universal property of the product span could be said to exhibit $A \times B$ as the "product" of A and B. The legs a product span are the categorical version of the *projections* of a product type; the unique map determined by the universal property of the product span is then the categorical version of the *pairing* operation. **Before proceeding further, make sure you have understood this analogy, getting help if you need it.**

Remark 26. In category theory, it is common to employ a harmless but imprecise metonymy by which a name (like "product") that refers to an entire diagram or configuration of objects and morphisms is attributed to a single object. There is no getting around this, but it can also cause confusion to learners of the subject.

Lemma 27 ("Product spans are unique up to unique isomorphism")**.** *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be two product spans between A and B. Then the unique morphism* $\mathbf{Q} \to \mathbf{P}$ *of spans between A and B is an isomorphism.*

Exercise 28 (Horizontal composition)**.** Suppose that we have products $A \times B$ and $U \times V$ in $\mathcal{C}$. Show that any two maps $U \xrightarrow{f} A$ and $V \xrightarrow{g} B$ can be combined into a single map $U \times V \xrightarrow{f \times g} A \times B$, using the universal property of the product $A \times B$ and the span structure of the product $U \times V$.

2.2.2 Finite products

Generalizing from the treatment of *binary* cartesian products above, we can account for n-ary cartesian products for any n. The cartesian product of zero objects is called a *terminal object*:

Definition 29. An object A of $\mathcal{C}$ is called *terminal* when for any other object $Y \in \mathcal{C}$, there exists a unique morphism $Y \to X$.

Then the cartesian product of $n + 1$ objects can be defined inductively using binary cartesian products and terminal objects. Of course, it makes most sense to define all of these simultaneously, but we have broken it up for the sake of familiarity.

2.2.3 Exponentials

Exponentials are the categorical version of *function types* with both the β- and the η-law. As we mentioned our discussion of products, even though we are studying a weak kind of function space with only β-laws, we will still need to understand exponentials.

Definition 30. Let A, B be a pair of objects in $\mathcal{C}$ such that for every $X \in \mathcal{C}$, the cartesian product $X \times A$ exists. An *exponential* of B by A is given by an object E together with a morphism $E \times A \xrightarrow{\epsilon} B$ (called the "evaluation map") such that for any $X \in \mathcal{C}$ and $X \times A \xrightarrow{f} B$ there exists a unique morphism $X \xrightarrow{\lambda f} E$ (called the "transpose") such that the following diagram commutes:

$$\begin{tikzcd}
X \times A \arrow[r, "f"] \arrow[d, "\lambda f \times \mathbf{id}_A"'] & B \\
E \times A \arrow[ur, "\epsilon"'] &
\end{tikzcd}$$

When $E \times A \xrightarrow{\epsilon} B$ exhibits E as the *exponential* of B by A, we will often write B^A for E. Then $B^A \times A \xrightarrow{B}$ is the categorical analogue of the *evaluation* function that takes a pair of a function and its argument and applies the former to the latter. As our notation suggests, the transpose $X \xrightarrow{\lambda f} B^A$ of a map $X \times A \xrightarrow{f} B$ is the categorical version of the λ-abstraction.

2.2.4 Cartesian closed categories

Definition 31 (Cartesian closed categories)**.** A category $\mathcal{C}$ is called *cartesian closed* when for any two objects $A, B \in \mathcal{C}$, both the cartesian product $A \times B$ and the exponential B^A exist.

There are two paradigmatic examples of cartesian closed categories:

1. The category **Set** of sets and functions between them; here the cartesian product is the ordinary cartesian product of two sets, and the exponential is given by the function space.
2. The category of contexts and substitutions of typed λ-calculus with product and function types, taken up to the $\equiv_{\beta\eta}$-congruence.

The method of categorical approaches to programming language metatheory is to use categories of the first style (categories of things that behave like sets) to study categories of the second paradigm (syntactic categories).

2.3 Categorical structure of the λ-calculus

Our initial goal is to describe how the λ-calculus and its various equational theories are rephrased in terms of categories; the purpose of this rephrasing is to bring to bear the more modular tools of category theory to state and prove results that are ultimately about the λ-calculus.[2]

[2]In future installments, we will see that in many cases, λ-calculus can be used conversely to simplify statements and proofs about category theory! It is in this way that category theory and type theory are locked into a *productive* dialectic spiral.

2.3.1 The structure of contexts

We will incrementally build up categorical notions that mirror the structures of the syntax of typed λ-calculus. To a first approximation, we shall think of the *objects* of a category as representing the *contexts* of a type theory, and the *morphisms* in the category shall represent the equivalence classes of *simultaneous substitutions* in the type theory.

In syntax, the structure of contexts is given by the empty context and the concatenation of contexts. In category theory, we would represent the empty context by a *terminal object* and then the context concatenation by a *cartesian product*; in other words, **a category with sufficient structure to model contexts is exactly a category with all finite products.**

Remark 32. In syntax, context concatenation is associative and unital. We will *not* have any need whatsoever to impose this condition on the categorical side, where it is quite unnatural and creates difficulties; in category theory, we automatically have associativity and unitality of cartesian products *up to isomorphism* but not up to equality. Classic coherence theorems suffice to bridge the gap between the two viewpoints.

2.3.2 Weak products

Let C be a category with finite products, *i.e.* a category modeling the structure of contexts. We will now describe for any two objects $A, B \in C$ the structure of a *weak product* of A and B, which should correspond to a product type in syntax with a β-law but no η-law.

Definition 33. A *weak product structure* for two objects A, B in a category C with finite products is defined to be section–retraction diagram of the following form:

$$\begin{tikzcd}
A \times B \arrow[r, "\mathsf{pair}"] \arrow[dr, "\mathbf{id}_{A\times B}"'] & P \arrow[d, "\mathsf{split}"] \\
& A \times B
\end{tikzcd}$$

In this scenario, we could write $A * B$ for P; the pair map is the categorical representation of the generic pair term $x : A, y : B \vdash \langle x, y \rangle : A * B$, and the split map bundles the representations of the generic first projection term $x : A * B \vdash x.1 : A$ and the generic second projection term $x : A * B \vdash x.2 : B$. The fact that the triangle commutes corresponds to the two β-laws for $A * B$.

Remark 34. Unlike the cartesian product, a weak product is not unique in any sense. There can be many non-isomorphic implementations of the interface of weak products; this is why they are more complex to study than cartesian products.

Exercise 35. Show that a weak product is a cartesian product if and only if the following additional triangle commutes:

$$\begin{tikzcd} P \arrow[r, "\mathsf{split}"] \arrow[dr, "\mathbf{id}_P"'] & A \times B \arrow[d, "\mathsf{pair}"] \\ & P \end{tikzcd}$$

2.3.3 Weak function objects

We now describe the structure of *weak function objects* in a category with finite products, corresponding to function types that have a β-law but no η-law.

Definition 36. A *weak function object structure* for two objects A, B in a category C with finite products to be an object E equipped with a morphism $E \times A \xrightarrow{\mathsf{ap}} B$ together with a function that assigns to each $X \times A \xrightarrow{f} B$ a morphism $X \xrightarrow{\mathsf{fn}\, f} E$ such that the following diagram commutes:

$$\begin{tikzcd} X \times A \arrow[r, "f"] \arrow[d, "\mathsf{fn}\, f \times \mathbf{id}_A"'] & B \\ E \times A \arrow[ur, "\mathsf{ap}"'] & \end{tikzcd}$$

and moreover for each $Y \xrightarrow{u} X$, the composite $Y \xrightarrow{u} X \xrightarrow{\mathsf{fn}\, f} E$ is equal to $\mathsf{fn}\,(Y \times A \xrightarrow{u \times \mathbf{id}_A} X \times A \xrightarrow{f} B)$.

We might suggestively write $A \Rightarrow B$ for a given weak function object of A and B when it exists, but note that (like weak products) these are not unique.

Remark 37. The condition in Definition 36 concerning precomposition with $Y \xrightarrow{u} X$ corresponds to the *substitution law* that commutes a substitution past a λ-abstraction. For actual exponentials, this additional condition is implied by the uniqueness of transposes (λ-abstraction); since weak function objects have no such uniqueness condition (which corresponds to an η-law), we must impose the substitution law explicitly.

Remark 38. We have deliberately chosen to write ap and fn instead of ϵ and λ for the constituent operations of the weak function object to avoid confusion with true exponentials ("strong" function spaces).

2.3.4 The structure of types

Categorical type theorists often treat contexts and types as if they are the same thing; in many cases this is fine as any type τ gives rise to a unary context $x : \tau$, but in order to have our results apply directly to the λ-calculus we have studied so far, we need to be a bit more cautious. In particular, we will consider a form of category that *distinguishes* certain objects as corresponding to types (*i.e.* being unary contexts). The purpose of this distinction is then that we will only require the distinguished contexts to be closed under type theoretic connectives like $\Rightarrow$ and $*$, exactly as in the syntactic presentation.

Definition 39. A *category with types* is defined to be a category C equipped with a chosen terminal object, chosen binary products, and a full subcategory $\mathcal{T}_C \subseteq C$, *i.e.* a subcategory of C such that for any $A, B \in \mathcal{T}_C$ the hom *sub*set $\mathbf{Hom}_{\mathcal{T}_C}(A, B)$ is all of $\mathbf{Hom}_C(A, B)$. We shall refer to an object of $\mathcal{T}_C$ as a *type*.

The word "chosen" in Definition 39 means that the presence of finite products is part of the structure, and is not merely an existence condition.

2.3.5 The boolean type

Definition 40. A *weak boolean type structure* for a category with types C is given by the following data:

1. a type $\mathsf{bool} \in \mathcal{T}_C$;
2. a pair of morphisms $\mathbf{1}_C \xrightarrow{\mathsf{tt},\mathsf{ff}} \mathsf{bool}$;
3. for each type $C \in \mathcal{T}_C$, a morphism $\mathsf{bool} \times C \times C \xrightarrow{\mathsf{if}_C} C$ such that the following diagram commutes:

$$\begin{tikzcd}
C \times C \arrow[r, "{(\mathsf{tt}, p_1, p_2)}"] \arrow[dr, "p_1"'] & \mathsf{bool} \times C \times C \arrow[d, "\mathsf{if}_C"] & C \times C \arrow[l, "{(\mathsf{ff}, p_1, p_2)}"'] \arrow[dl, "p_2"] \\
& C &
\end{tikzcd}$$

2.3.6 A categorical definition of λ-calculus

We now reach the culmination of the definitions of the present section.

Definition 41. A *model* of the λ-calculus with function, product, and boolean types up to the $\equiv_\beta$-congruence is exactly the same thing as a *category with types* equipped with a *weak boolean type structure* together with an assignment of a *weak product* $A * B$ and a *weak function object* $A \Rightarrow B$ for every pair of types $A, B \in \mathcal{T}_C$.

Remark 42 (Cartesian closed categories)**.** Some readers may be familiar with the concept of a *cartesian closed category*, which is exactly a model of the λ-calculus with function and product types under the $\equiv_{\beta\eta}$-congruence. When η-laws hold, every context can be represented as a type, so $\mathcal{T}_C = C$.

2.3.6.1 Denotational semantics

The categorical notion of model that we have described is *sound and complete* for the syntax of typed λ-calculus up to $\equiv_\beta$. For a given model C, the interpretation is specified as follows:

1. a context Γ is interpreted as an object $[\![\Gamma]\!] \in C$;
2. a type τ is interpreted as an object $[\![\tau]\!] \in \mathcal{T}_C$;
3. a term $\Gamma \vdash M : \tau$ is interpreted as a morphism $[\![\Gamma]\!] \xrightarrow{[\![M]\!]} [\![\tau]\!]$ in C;
4. an equation $\Gamma \vdash M \equiv_\beta N : \tau$ is interpreted as a mathematical equation $[\![M]\!] = [\![N]\!]$.

Thus the theory of typed λ-calculus under $\equiv_\beta$ is sound for all models C. To see that it is *complete*, we may construct from the *syntactic model*, which we outline below:

1. an object of C is a syntactic context;
2. the full subcategory $\mathcal{T}_\subseteq C$ is spanned by unary contexts of the form $x : \tau$;
3. a morphism from Γ to Δ is given by a $\equiv_\beta$-equivalence class of simultaneous substitutions.

Together, soundness and completeness allow us to leave the world of syntax behind and work purely in semantics.

2.3.6.2 Remarks

It is worth being very clear about the fact that we have done nothing more than *rephrase* the familiar definitions of λ-calculus in a more modular way. Nothing new has been added — which is important, because if we use category theory to prove something about the categorical version of the λ-calculus, it had better be the case that this actually implies something about the actual λ-calculus. The role of category theory, as ever, is to cut across a given interface and rearrange it so that it can be manipulated more modularly.

Every category theoretic proof or construction can (tautologically) be unfolded to a non-category theoretic proof, so the purpose of introducing categories is not to make new things provable in principle, but rather to make new things provable in *practice*. Once this has been done, the requirements of dissemination and publication within a deeply conservative field often entail unraveling the categorical machinery to the point where someone who does not know the history of a given work might assume that it was carried out in innocence of category theory.

2.4 Functorial semantics

So far we have discussed a categorical notion of model for the λ-calculus. The strength of the categorical machinery lies, however, in the fact that these models themselves can be arranged into a category in which morphisms denote structure-preserving translations between different models of λ-calculus.

Definition 43. Let $\mathcal{C}, \mathcal{D}$ be two models of the λ-calculus in the sense of Definition 41.A functor $\mathcal{C} \xrightarrow{F} \mathcal{D}$ is called a *homomorphism of models* when it satisfies the following conditions:

1. it sends $\mathbf{1}_{\mathcal{C}}$ to the chosen terminal object $\mathbf{1}_{\mathcal{D}}$;
2. it sends $A \times B$ to the chosen cartesian product $FA \times FB$;
3. it sends types to types, *i.e.* if $A \in \mathcal{T}_{\mathcal{C}}$ then $FA \in \mathcal{T}_{\mathcal{D}}$;
4. it strictly preserves the boolean type structure, the weak function type structures, and the weak product type structures.

Remark 44. For clarity, we unfold what it means to preserve the weak product type structure. For two types $A, B \in \mathcal{T}_{\mathcal{C}}$, let the following diagram denote their weak product type structure:

$$\begin{tikzcd}
A \times B \arrow[r, "\mathsf{pair}"] \arrow[dr, "\mathbf{id}_{A\times B}"'] & A * B \arrow[d, "\mathsf{split}"] \\
& A \times B
\end{tikzcd}$$

Applying F to the diagram above, we have a new diagram in $\mathcal{D}$:

$$\begin{tikzcd}
FA \times FB \arrow[r, "F\,\mathsf{pair}"] \arrow[dr, "\mathbf{id}_{F(A\times B)}"'] & F(A * B) \arrow[d, "F\,\mathsf{split}"] \\
& FA \times FB
\end{tikzcd}$$

The requirement that F strictly preserve weak product type structures is then that $F(A * B) = FA * FB$ and moreover that the diagram above shall be the chosen weak product type structure in $\mathcal{D}$.

The other preservation conditions expand analogously. We are now ready to state a categorical reformulation of the soundness and completeness of the denotational semantics:

Theorem 45 (Initiality)**.** *The syntactic model of the λ-calculus is the* initial object *in the category of λ-models and homomorphisms of λ-models.*

Remark 46. Professionals usually prefer to work with a structure in which the cartesian products are not chosen, but instead only assumed to exist; then these products are preserved up to isomorphism rather than up to equality. This results in a *bicategorical* version of the initiality theorem which is considerably harder to state, but equally more powerful. We have stayed in the world of 1-categories in order to avoid introducing too many things at once.

Chapter 3

A general logical relation

We will describe a *general* logical relation that applies to all models of the λ-calculus. The immediate benefit of using categorical language to formulate this logical relation is that it can be done modularly in small and reusable components, and then only at the end assembled into a single logical relation. This is particularly useful as it allows logical relations for different languages to be cobbled together rather than created out of whole cloth each time.

3.1 What is a logical relation?

We have seen in Chapter 1 that a logical relation on the λ-calculus is an interpretation of λ-calculus in which each type is a taken to *semantic predicate* on the terms of a given type in the syntax. Category theory takes this picture and generalizes it in a couple directions.

1. Any model of the λ-calculus can play the role of the syntax in a logical relation, *i.e.* instead of considering predicates on terms, we could consider predicates on the elements within a model.
2. The predicates can be replaced with *indexed families*, leading to a proof-relevant version of logical relations. Here intead of $[\![\tau]\!]$ being a subset of the closed terms of type τ, it could be a family of sets indexed in the closed terms of type τ.
3. Instead of considering closed terms, we could consider terms in certain kinds of contexts or even Kripke worlds (as in Kripke logical relations).

The different dimensions of generality mentioned above have interesting effects. When we generalize conventional presentations of logical relations in these directions, they get harder to construct; from the categorical viewpoint, these generalizations tend to make things *easier* — and we are rewarded with a machine for producing very complex proof-relevant Kripke logical relations without much additional effort.

3.2 A logical relation for contexts

Let $\mathcal{C}$ and $\mathcal{D}$ be two categories with finite products, and let $\mathcal{C} \xrightarrow{F} \mathcal{D}$ be a functor that takes cartesian product spans to cartesian product spans. These data express the basic configuration of a logical relation; before making this precise, we foreshadow a few representative examples.

Example 47. Let $\mathcal{C}$ be the category of contexts of the λ-calculus and let $\mathcal{D}$ be the category of sets, and let F is the functor that sends each Γ to the set of equivalence classes of closed environments $F\Gamma = \mathbf{Hom}_{\mathcal{C}}(\mathbf{1}_{\mathcal{C}}, \Gamma)$, also known as the *global sections functor*. This is the configuration of a proof-relevant logical relation that interprets types as families of sets indexed in closed terms of a given syntactic type, and it is indeed the correct configuration of the categorical version of the closed termination proof.

Example 48. Now let $\mathcal{R}$ be the category of contexts, with morphisms given only by *renamings*, *i.e.* substitutions that send variables to variables rather than variables to terms; we have an obvious functor $\mathcal{R} \xrightarrow{I} \mathcal{C}$ that sends each renaming to the substitution it corresponds to. Then let $\mathcal{D}$ be the category of functors from $\mathcal{R}^{op}$ to **Set**; we have a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ that sends each Γ to the functor $\mathcal{R}^{op} \xrightarrow{F\Gamma} \mathbf{Set}$ sending $\Delta \in \mathcal{R}^{op}$ to the set of substitutions $\mathbf{Hom}_{\mathcal{C}}(I\Delta, \Gamma)$. This is the configuration of a logical relation that can be used to prove **normalization of open terms** for the λ-calculus.

Example 47 is easy to do by hand, but Example 48 is highly non-trivial and corresponds to a Kripke logical relation (see the classic paper of Jung and Tiuryn [10] for this perspective on Kripke logical relations). The benefit of the categorical viewpoint is that both logical relations will arise in exactly the same way; in particular, the difficulties of Kripke logical relations end up being abstracted away by category theory.

Definition 49. Let $\mathcal{C}$ and $\mathcal{D}$ be two categories, and let $\mathcal{C} \xrightarrow{F} \mathcal{D}$ be a functor between them. The *Artin gluing* of $\mathcal{C}$ and $\mathcal{D}$ along F is the category $\mathcal{G}_F$ whose objects are given by triples $A = (\underline{A} \in \mathcal{C}, \overline{A} \in \mathcal{D}, \overline{A} \xrightarrow{\varphi_A} F\underline{A})$; a morphism from A to B in $\mathcal{G}_F$ is given by a pair $f = (\underline{A} \xrightarrow{\underline{f}} \underline{B}, \overline{A} \xrightarrow{\overline{f}} \overline{B})$ such that the following square commutes in $\mathcal{D}$:

$$\begin{tikzcd}
\overline{A} \arrow[r, "\overline{f}"] \arrow[d, "\varphi_A"'] & \overline{B} \arrow[d, "\varphi_B"] \\
F\underline{A} \arrow[r, "F\underline{f}"'] & F\underline{B}
\end{tikzcd}$$

Moreover, we have a functor $\mathcal{G}_F \xrightarrow{\mathsf{gl}_F} \mathcal{C}$ sending each A to $\underline{A}$.

We must spend a bit of time coming to understand this definition and how it relates to conventional logical relations. Returning to Example 47, we consider the fact that C is the category of contexts of the λ-calculus and $\mathcal{D}$ is the category of sets, with F the global sections functor that takes each context to its set of equivalence classes of closing evironments.

In this case, an object of $\mathcal{G}_F$ is given by a context $\underline{\Gamma}$, a set $\overline{\Gamma}$ and a function $\overline{\Gamma} \xrightarrow{\varphi_\Gamma} (\cdot \vdash \underline{\Gamma})$. To make contact with a more familiar form of logical relation, we consider the special case that φ_Γ is an *injective* function; then φ_Γ is determined by its image, which is a *subset* of $(\cdot \vdash \underline{\Gamma})$. Allowing φ_Γ to *not* to be injective corresponds to the *proof relevant* generalization of logical relations that we have mentioned before; indeed, in this case $\overline{\Gamma} \xrightarrow{\varphi_\Gamma} (\cdot \vdash \underline{\Gamma})$ is determined by its *fibers* $\varphi_\Gamma^{-1}\gamma \in \mathbf{Set}$ for each environment $\gamma \in (\cdot \vdash \Gamma)$. In this case, the fiber $\varphi_\Gamma^{-1}\gamma$ is the set of **proofs** that γ "in" the logical relation.

Lemma 50. *Let C be a category with finite cartesian products and let $\mathcal{D}$ be a category with binary cartesian products; let $C \xrightarrow{F} \mathcal{D}$ be a functor that preserves binary products. Then $\mathcal{G}_F$ has finite products and gl_F preserves them; moreover, if C has* chosen *finite cartesian products, the finite cartesian products in $\mathcal{G}_F$ can be chosen such that gl_F preserves the choices strictly.*

It is worth paying attention to the (surprising) fact that we do not need $\mathcal{D}$ to have a terminal object.

Proof. Let $\mathbf{1}_C$ be a given terminal object in C; we define $\mathbf{1}_{\mathcal{G}_F}$ to be the object $(\mathbf{1}_C, F\mathbf{1}_C, F\mathbf{1}_C \xrightarrow{F\mathbf{id}_{\mathbf{1}_C}} F\mathbf{1}_C)$. Given an object $A = (\underline{A}, \overline{A}, \varphi_A)$ of $\mathcal{G}_F$, we define universal map $A \longrightarrow \mathbf{1}_{\mathcal{G}_F}$ to be the following square:

$$\begin{tikzcd}
\overline{A} \arrow[r, "F!_{\underline{A}} \circ \varphi_A"] \arrow[d, "\varphi_A"'] & F\mathbf{1}_C \arrow[d, "F\mathbf{id}_{\mathbf{1}_C}"] \\
F\underline{A} \arrow[r, "F!_{\underline{A}}"'] & F\mathbf{1}_C
\end{tikzcd}$$

To see that the map above is unique, we will use the fact that f preserves binary cartesian products. Fix $u, v : A \longrightarrow \mathbf{1}_{\mathcal{G}_F}$; clearly $\underline{u} = \underline{v}$, and to check that $\overline{u} = \overline{v}$ is the same as to check that the following diagram commutes:

$$\begin{tikzcd}
\overline{A} \arrow[r, "\langle \overline{u}, \overline{v} \rangle"] & F\mathbf{1}_C \times F\mathbf{1}_C \arrow[r, bend left, "\pi_1"] \arrow[r, bend right, "\pi_2"'] & F\mathbf{1}_C
\end{tikzcd}$$

Because F preserves binary products, this follows from the fact that the following diagram evidently commutes since $F\pi_1 = F\pi_2$:

$$\begin{tikzcd}
\overline{A} \arrow[r, "\langle \overline{u}, \overline{v} \rangle"] & F\mathbf{1}_C \times F\mathbf{1}_C \arrow[r, "\cong"] & F(\mathbf{1}_C \times \mathbf{1}_C) \arrow[r, bend left, "F\pi_1"] \arrow[r, bend right, "F\pi_2"'] & F\mathbf{1}_C
\end{tikzcd}$$

For binary products, let A and B be objects of $\mathcal{G}_F$, and let $\underline{A} \xleftarrow{\pi_{\underline{A}}} \underline{A} \times \underline{B} \xrightarrow{\pi_{\underline{B}}} \underline{B}$ be a chosen cartesian product span in $\mathcal{C}$. We now consider the following commuting diagram in $\mathcal{D}$:

$$\begin{tikzcd}
\overline{A} \arrow[d, "\varphi_A"'] & \overline{A} \times \overline{B} \arrow[l, "\pi_{\overline{A}}"'] \arrow[r, "\pi_{\overline{B}}"] \arrow[d, "\varphi_A \times \varphi_B"] & \overline{B} \arrow[d, "\varphi_B"] \\
F\underline{A} & F\underline{A} \times F\underline{B} \arrow[l, "\pi_{F\underline{A}}"'] \arrow[r, "\pi_{F\underline{B}}"] & F\underline{B} \\
 & F(\underline{A} \times \underline{B}) \arrow[ul, "F\pi_{\underline{A}}"] \arrow[u, "\cong"] \arrow[ur, "F\pi_{\underline{B}}"'] &
\end{tikzcd}$$

Forgetting the interior, we now have a span in $\mathcal{G}_F$ that we shall claim is a cartesian product span, defining $\overline{A \times B} :\equiv \overline{A} \times \overline{B}$.

$$\begin{tikzcd}
\overline{A} \arrow[d, "\varphi_A"'] & \overline{A \times B} \arrow[l, "\pi_{\overline{A}}"'] \arrow[r, "\pi_{\overline{B}}"] \arrow[d, "\varphi_{A\times B}"] & \overline{B} \arrow[d, "\varphi_B"] \\
F\underline{A} & F(\underline{A} \times \underline{B}) \arrow[l, "F\pi_{\underline{A}}"] \arrow[r, "F\pi_{\underline{B}}"'] & F\underline{B}
\end{tikzcd}$$

We will write $\mathbf{P}$ for the span above; to prove that $\mathbf{P}$ is in fact a cartesian product span, we fix another span $\mathbf{Q} = (A \xleftarrow{q_A} Q \xrightarrow{q_B} B)$ in $\mathcal{G}_F$; then we must show that there is exactly *one* morphisms of spans $\mathbf{Q} \xrightarrow{h} \mathbf{P}$. First of all, we have a unique morphism of spans $\underline{\mathbf{Q}} \xrightarrow{\underline{h}} \underline{\mathbf{P}}$ because $\underline{\mathbf{P}}$ is a cartesian product span in $\mathcal{C}$; upstairs we also have a unique morphism of spans $\overline{\mathbf{Q}} \xrightarrow{\overline{h}} \overline{\mathbf{P}}$ because $\overline{\mathbf{P}}$ is a cartesian product span in $\mathcal{D}$. Therefore it remains only to show $(\underline{h}, \overline{h})$ together comprise a morphism in $\mathcal{G}_F$, which amounts to checking that the following square commutes:

$$\begin{tikzcd}
\overline{Q} \arrow[r, "\overline{h}"] \arrow[d, "\varphi_Q"'] & \overline{A} \times \overline{B} \arrow[d, "\varphi_{A\times B}"] \\
F\underline{Q} \arrow[r, "F\underline{h}"'] & F(\underline{A} \times \underline{B})
\end{tikzcd}$$

Because F preserves cartesian products, it suffices to check that the following modified diagram commutes:

$$\begin{tikzcd}
\overline{Q} \arrow[r, "\overline{h}"] \arrow[d, "\varphi_Q"'] & \overline{A} \times \overline{B} \arrow[d, "\varphi_A \times \varphi_B"] \\
F\underline{Q} \arrow[r, "\langle F\underline{q}_A, F\underline{q}_B \rangle"'] & F\underline{A} \times F\underline{B}
\end{tikzcd}$$

To check that two maps $\overline{Q} \to F\underline{A} \times F\underline{B}$ into a cartesian product are equal, it suffices to consider their behavior on projections. But this is exactly the property that $\mathbf{Q}$ forms a span in $\mathcal{G}_F$, so we are done. □

3.3 A logical relation for weak products

We will now begin to establish the *closure* of logical relations under various forms of type structure, now in a more modular way than was possible for the introduction of categorical structure.

Lemma 51. *Let C be a category with finite cartesian products and let $\mathcal{D}$ be a category with binary cartesian products, and let $C \xrightarrow{F} \mathcal{D}$ be a functor preserving binary cartesian products. Let A and B be objects of C, and let*

$$\underline{\mathbf{P}} = (\underline{A} \times \underline{B} \xrightarrow{\underline{\mathsf{pair}}} \underline{A} * \underline{B} \xrightarrow{\underline{\mathsf{split}}} \underline{A} \times \underline{B})$$

be a given weak product structure in C on $\underline{A}$ and $\underline{B}$. Then there exists a weak product structure $\mathbf{P}$ in $\mathcal{G}_F$ such that $\mathsf{gl}_F\mathbf{P} = \underline{\mathbf{P}}$.

Proof. We shall define $A * B$ to be the following object of $\mathcal{G}_F$:

$$A * B :\equiv (\underline{A} * \underline{B}, \overline{A} \times \overline{B}, \overline{A} \times \overline{B} \xrightarrow{\varphi_{A\times B}} F(\underline{A} \times \underline{B}) \xrightarrow{F\,\underline{\mathsf{pair}}} F(\underline{A} * \underline{B}))$$

Note that we have used above the cartesian product $\overline{A} \times \overline{B} \xrightarrow{\varphi_{A\times B}} F(\underline{A} \times \underline{B})$ from Lemma 50. The pairing map in $\mathcal{G}_F$ is defined to be $\mathsf{pair} :\equiv (\underline{\mathsf{pair}}, \mathbf{id}_{\overline{A}\times\overline{B}})$; the necessary square commutes by definition:

$$\begin{tikzcd}
\overline{A} \times \overline{B} \arrow[r, "\mathbf{id}_{\overline{A}\times\overline{B}}"] \arrow[d, "\varphi_{A\times B}"'] & \overline{A} \times \overline{B} \arrow[d, "\varphi_{A*B}"] \\
F(\underline{A} \times \underline{B}) \arrow[r, "F(\underline{\mathsf{pair}})"'] & F(\underline{A} * \underline{B})
\end{tikzcd}$$

The splitting map is defined conversely to be $\mathsf{split} :\equiv \big(\underline{\mathsf{split}}, \mathbf{id}_{\overline{A}\times\overline{B}}\big)$; we must check that the following square commutes:

$$\begin{tikzcd}
\overline{A}\times\overline{B} \arrow[r, "\mathbf{id}_{\overline{A}\times\overline{B}}"] \arrow[d, "\varphi_{A*B}"'] & \overline{A}\times\overline{B} \arrow[d, "\varphi_{A\times B}"] \\
F(\underline{A} * \underline{B}) \arrow[r, "F\,\underline{\mathsf{split}}"'] & F(\underline{A}\times\underline{B})
\end{tikzcd}$$

To see that this commutes, we unfold the definition of φ_{A*B} and compute the lower-left composite; the equation follows from the fact that $\underline{\mathsf{split}}\circ\underline{\mathsf{pair}} = \mathbf{id}_{\underline{A}\times\underline{B}}$:

$$\begin{aligned}
&\overline{A}\times\overline{B} \xrightarrow{\varphi_{A*B}} F(\underline{A}*\underline{B}) \xrightarrow{F\,\underline{\mathsf{split}}} F(\underline{A}\times\underline{B})\\
&= \overline{A}\times\overline{B} \xrightarrow{\varphi_A\times\varphi_B} F\underline{A}\times F\underline{B} \xrightarrow{\cong} F(\underline{A}\times\underline{B}) \xrightarrow{F\,\underline{\mathsf{pair}}} F(\underline{A}*\underline{B}) \xrightarrow{F\,\underline{\mathsf{split}}} F(\underline{A}\times\underline{B})\\
&= \overline{A}\times\overline{B} \xrightarrow{\varphi_A\times\varphi_B} F\underline{A}\times F\underline{B} \xrightarrow{\cong} F(\underline{A}\times\underline{B})\\
&= \overline{A}\times\overline{B} \xrightarrow{\varphi_{A\times B}} F(\underline{A}\times\underline{B})
\end{aligned}$$

It remains to check that $\mathsf{split}\circ\mathsf{pair} = \mathbf{id}_{A\times B}$, which follows immediately from $\underline{\mathsf{split}}\circ\underline{\mathsf{pair}} = \mathbf{id}_{\underline{A}\times\underline{B}}$ since the upper components in $\mathcal{D}$ are identity maps. □

3.4 A logical relation for weak function objects

Lemma 52. *Let C be a category with finite cartesian products and let $\mathcal{D}$ be a category with binary cartesian products and pullbacks, and let $C \xrightarrow{F} \mathcal{D}$ be a functor preserving binary cartesian products. Let A and B be two objects of C, and let $(\underline{A} \Rightarrow \underline{B}, \underline{\mathsf{ap}}, \underline{\mathsf{fn}})$ be a given weak function object structure in C on $\underline{A}$ and $\underline{B}$. Then there exists a weak product structure $(A \Rightarrow B, \mathsf{ap}, \mathsf{fn})$ in $\mathcal{G}_F$ on A, B that is preserved by gl_F.*

Our assertion that the weak product structure is preserved by gl_F means, explicitly, that $\mathsf{gl}_F(A \Rightarrow B) = (\underline{A} \Rightarrow \underline{B})$, and $\mathsf{gl}_F\mathsf{ap} = \underline{\mathsf{ap}}$, and for any $X\times A \xrightarrow{f} B$ we have $\mathsf{gl}_F(\mathsf{fn}\,f) = \underline{\mathsf{fn}}\,(\mathsf{gl}_F f)$.

Proof. We shall define $A \Rightarrow B$ to be the following pullback, which combines a "semantic" function and a "syntactic" function in a compatible way:

$$\begin{tikzcd}
\overline{A \Rightarrow B} \arrow[rr, dashed, "k"] \arrow[d, dashed, "\varphi_{A\Rightarrow B}"'] \arrow[dr, phantom, "\lrcorner", very near start] & & \overline{B}^{\overline{A}} \arrow[d, "(\varphi_B)^{\overline{A}}"] \\
F(\underline{A} \Rightarrow \underline{B}) \arrow[rr, "l"] \arrow[dr, "\theta"'] & & (F\underline{B})^{\overline{A}} \\
& (F\underline{B})^{F\underline{A}} \arrow[ur, "(F\underline{B})^{\varphi_A}"'] &
\end{tikzcd}$$

The map $F(\underline{A} \Rightarrow \underline{B}) \xrightarrow{\theta} (F\underline{B})^{F\underline{A}}$ employed above is defined to be the transpose of the composite $F(\underline{A} \Rightarrow \underline{B}) \times F\underline{A} \xrightarrow{F_\times} F((\underline{A} \Rightarrow \underline{B}) \times \underline{A}) \xrightarrow{F\,\underline{\mathsf{ap}}} F\underline{B}$ where $F_\times$ is the canonical isomorphism induced by F preserving cartesian products. In other words, we have $\vartheta :\equiv \lambda(F\,\underline{\mathsf{ap}} \circ F_\times)$. The right-hand map $(\varphi_B)^{\underline{A}}$ is *postcomposition* with the projection φ_B; the lower-right map $(F\underline{B})^{\varphi_A}$ is *precomposition* with the projection φ_A; these can be computed more explicitly as follows:

$$\begin{aligned}
(F\underline{B})^{\varphi_A} &:\equiv \lambda\big((F\underline{B})^{F\underline{A}} \times \overline{A} \xrightarrow{\mathbf{id}_{(F\underline{B})^{F\underline{A}}} \times \varphi_A} (F\underline{B})^{F\underline{A}} \times F\underline{A} \xrightarrow{\epsilon} F\underline{B}\big) \\
(\varphi_B)^{\overline{A}} &:\equiv \lambda\big(\overline{B}^{\overline{A}} \times \overline{A} \xrightarrow{\epsilon} \overline{B} \xrightarrow{\varphi_B} F\underline{B}\big)
\end{aligned}$$

Thus it follows that the downstairs map l can be computed as the following transpose:

$$l = \lambda\big(F(\underline{A} \Rightarrow \underline{B}) \times \overline{A} \xrightarrow{\mathbf{id}_{F(\underline{A}\Rightarrow\underline{B})} \times \varphi_A} F(\underline{A} \Rightarrow \underline{B}) \times F\underline{A} \xrightarrow{F_\times} F((\underline{A} \Rightarrow \underline{B}) \times \underline{A}) \xrightarrow{F\,\underline{\mathsf{ap}}} F\underline{B}\big)$$

We define the application map $(A \Rightarrow B) \times A \xrightarrow{\mathsf{ap}} B$ in $\mathcal{G}_F$ to be the pair $(\underline{\mathsf{ap}}, \overline{A \Rightarrow B} \times \overline{A} \xrightarrow{k \times \mathbf{id}_{\overline{A}}} \overline{B}^{\overline{A}} \xrightarrow{\epsilon} \overline{B})$; we have to check that the inner square depicted below commutes:

$$\begin{tikzcd}
& \overline{B}^{\overline{A}} \times \overline{A} \arrow[dr, "\epsilon"] & \\
\overline{A \Rightarrow B} \times \overline{A} \arrow[ur, "k \times \mathbf{id}_{\overline{A}}"] \arrow[rr, "\overline{\mathsf{ap}}"] \arrow[d, "\varphi_{(A\Rightarrow B)\times A}"'] & & \overline{B} \arrow[d, "\varphi_B"] \\
F((\underline{A} \Rightarrow \underline{B}) \times \underline{A}) \arrow[rr, "F\,\underline{\mathsf{ap}}"'] & & F\underline{B}
\end{tikzcd}$$

We will compute $\varphi_B \circ \overline{\mathsf{ap}}$ using our original pullback square.

$$\begin{tikzcd}
\overline{A \Rightarrow B} \times \overline{A} \arrow[r, "k \times \mathbf{id}_{\overline{A}}"] \arrow[d, "\varphi_{A\Rightarrow B} \times \mathbf{id}_{\overline{A}}"'] & \overline{B}^{\overline{A}} \times \overline{A} \arrow[r, "\epsilon"] \arrow[d, "(\varphi_B)^{\overline{A}} \times \overline{A}"] & \overline{B} \arrow[d, "\varphi_B"] \\
F(\underline{A} \Rightarrow \underline{B}) \times \overline{A} \arrow[r, "l \times \mathbf{id}_{\overline{A}}"'] & (F\underline{B})^{\overline{A}} \times \overline{A} \arrow[r, "\epsilon"'] & F\underline{B}
\end{tikzcd}$$

We may now verify by equational computation the well-definedness property for the map $\mathsf{ap} = (\underline{\mathsf{ap}}, \overline{\mathsf{ap}})$:

$$\begin{aligned}
\varphi_B \circ \overline{\mathsf{ap}} &= \varphi_B \circ \epsilon \circ (k \times \mathbf{id}_{\overline{A}}) \\
&= \epsilon \circ (l \times \mathbf{id}_{\overline{A}}) \circ (\varphi_{A\Rightarrow B} \times \mathbf{id}_{\overline{A}}) \\
&= F\,\underline{\mathsf{ap}} \circ F_{\times} \circ (\mathbf{id}_{F(\underline{A}\Rightarrow\underline{B})} \times \varphi_A) \circ (\varphi_{A\Rightarrow B} \times \mathbf{id}_{\overline{A}}) \\
&= F\,\underline{\mathsf{ap}} \circ F_{\times} \circ (\varphi_{A\Rightarrow B} \times \varphi_A) \\
&= F\,\underline{\mathsf{ap}} \circ \varphi_{A\Rightarrow B\times A}
\end{aligned}$$

For the function abstraction, fix $X \times A \xrightarrow{f} B$ in order to define $X \xrightarrow{\mathsf{fn}\, f} A \Rightarrow B$ in $\mathcal{G}_F$. Our goal is to construct the dotted map depicted below.

$$\begin{tikzcd}
\overline{X} \arrow[r, dashed, "\overline{\mathsf{fn}\, f}"] \arrow[d, "\varphi_X"'] & \overline{A \Rightarrow B} \arrow[r, "k"] \arrow[d, "\varphi_{A\Rightarrow B}"] \arrow[dr, phantom, "\lrcorner", very near start] & \overline{B}^{\overline{A}} \arrow[d, "(\varphi_B)^{\overline{A}}"] \\
F\underline{X} \arrow[r, "F(\underline{\mathsf{fn}\, f})"'] & F(\underline{A} \Rightarrow \underline{B}) \arrow[r, "l"'] & (F\underline{B})^{\overline{A}}
\end{tikzcd}$$

By the universal property of the pullback, it suffices to construct the dotted map depicted below such that the outermost square commutes:

$$\begin{tikzcd}
\overline{X} \arrow[rr, dashed, bend left, "h :\equiv \lambda\overline{f}"] \arrow[r, "\overline{\mathsf{fn}\, f}"] \arrow[d, "\varphi_X"'] & \overline{A \Rightarrow B} \arrow[r, "k"] \arrow[d, "\varphi_{A\Rightarrow B}"] \arrow[dr, phantom, "\lrcorner", very near start] & \overline{B}^{\overline{A}} \arrow[d, "(\varphi_B)^{\overline{A}}"] \\
F\underline{X} \arrow[r, "F(\underline{\mathsf{fn}\, f})"'] & F(\underline{A} \Rightarrow \underline{B}) \arrow[r, "l"'] & (F\underline{B})^{\overline{A}}
\end{tikzcd}$$

To check that the outermost square commutes, we will use the universal property of the exponential $(F\underline{B})^{\overline{A}}$: for two maps $\overline{X} \xrightarrow{u,v} (F\underline{B})^{\overline{A}}$ to be equal is equivalent to the composite maps $\overline{X}\times\overline{A} \xrightarrow{u\times\mathbf{id}_{\overline{A}},\, v\times\mathbf{id}_{\overline{A}}} (F\underline{B})^{\overline{A}}\times\overline{A} \xrightarrow{\epsilon} F\underline{B}$ are equal.

$$\begin{aligned}
&\epsilon\circ((l\circ F(\underline{\mathsf{fn}\,f})\circ\varphi_X)\times\mathbf{id}_{\overline{A}})\\
&\quad=\epsilon\circ(l\times\mathbf{id}_{\overline{A}})\circ(F(\underline{\mathsf{fn}\,f})\times\mathbf{id}_{\overline{A}})\circ(\varphi_X\times\mathbf{id}_{\overline{A}})\\
&\quad=F\,\underline{\mathsf{ap}}\circ F_\times\circ(\mathbf{id}_{F(\underline{A\Rightarrow B})}\times\varphi_A)\circ(F(\underline{\mathsf{fn}\,f})\times\mathbf{id}_{\overline{A}})\circ(\varphi_X\times\mathbf{id}_{\overline{A}})\\
&\quad=F\,\underline{\mathsf{ap}}\circ F_\times\circ(F(\underline{\mathsf{fn}\,f})\times\mathbf{id}_{F\underline{A}})\circ(\varphi_X\times\varphi_A)\\
&\quad=F\,(\underline{\mathsf{ap}}\circ(\underline{\mathsf{fn}\,f}\times\mathbf{id}_{\underline{A}}))\circ(\varphi_X\times\varphi_A)\\
&\quad=F\underline{f}\circ\varphi_{X\times A}
\end{aligned}$$

We compute the other composite as follows:

$$\begin{aligned}
&\epsilon\circ(((\varphi_B)^{\overline{A}}\circ\lambda\overline{f})\times\mathbf{id}_{\overline{A}})\\
&\quad=\epsilon\circ((\varphi_B)^{\overline{A}}\times\mathbf{id}_{\overline{A}})\circ(\lambda\overline{f}\times\mathbf{id}_{\overline{A}})\\
&\quad=\varphi_B\circ\epsilon\circ(\lambda\overline{f}\times\mathbf{id}_{\overline{A}})\\
&\quad=\varphi_B\circ\overline{f}
\end{aligned}$$

That $F\underline{f}\circ\varphi_{X\times A}=\varphi_B\circ\overline{f}$ is exactly the defining condition for the morphism $X\times A\xrightarrow{f}B$ in $\mathcal{G}_F$:

$$\begin{tikzcd}
\overline{X}\times\overline{A} \arrow[r, "\overline{f}"] \arrow[d, "\varphi_{X\times A}"'] & \overline{B} \arrow[d, "\varphi_B"] \\
F(\underline{X}\times\underline{A}) \arrow[r, "F\underline{f}"'] & F\underline{B}
\end{tikzcd}$$

Next we check the β-law, namely that $\mathsf{ap}\circ(\mathsf{fn}\,f\times\mathbf{id}_A)=f$; this follows easily from the corresponding law for the weak product in $\mathcal{C}$ and the exponential in $\mathcal{D}$. The only thing to check is the upper component:

$$\begin{aligned}
\overline{\mathsf{ap}\circ(\mathsf{fn}\,f\times\mathbf{id}_A)}&=\overline{\mathsf{ap}}\circ(\overline{\mathsf{fn}\,f}\times\mathbf{id}_A)\\
&=\epsilon\circ(k\times\mathbf{id}_{\overline{A}})\circ(\overline{\mathsf{fn}\,f}\times\mathbf{id}_{\overline{A}})\\
&=\epsilon\circ((k\circ\overline{\mathsf{fn}\,f})\times\mathbf{id}_{\overline{A}})\\
&=\epsilon\circ(\lambda\overline{f}\times\mathbf{id}_{\overline{A}})\\
&=\overline{f}
\end{aligned}$$

Finally we must check the *naturality* or *substitution* law for fn from Definition 36. Fixing $Y\xrightarrow{u}X$, we must check that $Y\xrightarrow{u}X\xrightarrow{\mathsf{fn}\,f}A\Rightarrow B$ is equal to $\mathsf{fn}(Y\times A\xrightarrow{u\times\mathbf{id}_A}X\times A\xrightarrow{f}B)$. This follows almost immediately from the corresponding conditions for the weak function object in $\mathcal{C}$ and the exponential object in $\mathcal{D}$, via the universal property of the pullback that defines $\overline{A\Rightarrow B}$. □

3.4.1 Types in the logical relation

TODO

3.4.2 A logical relation for weak boolean types

TODO

3.4.3 Case study: deducing the termination theorem

TODO

3.4.4 Towards synthetic Tait computability: make the pain stop!

The verifications involved in the proof of Lemma 52 were admittedly quite painful! It is a recurring theme that although category theory is an excellent tool for modularly organizing the *large-scale* structure of mathematical developments, it tends to fall down when trying to do explicit constructions. From a category theorist's viewpoint, the entire purpose of tools like type theory and λ-calculus is to provide a more expedient language for deducing theorems in categories that have the requisite structure.

Indeed, since λ-calculus can be interpreted quite transparently in any cartesian closed category (such as $\mathcal{D}$), we can use λ-terms as an alternative to the very painful categorical combinators that we have suffered through in Lemma 52. The main subtlety is that we have to account for in this new λ-calculus for the F functor, but this is not easy at all; in particular, while it is clear enough that F should be some kind of modality satisfying the rules of an *applicative functor* [12], the difficulty is that things *underneath* this modality come from a different language (the language of $\mathcal{C}$).

The solution to this problem is to replace $\mathcal{G}_F$ with a more structured category $\mathcal{G}_{\hat{F}}$, obtained by taking the Artin gluing of a functor $\hat{F}$ derived from F, such that $\mathcal{G}_F$ embeds into $\mathcal{G}_{\hat{F}}$ as a full subcategory. It will so happen that standard/off-the-shelf results of category theory will ensure that $\mathcal{G}_{\hat{F}}$ is a *topos*, *i.e.* a model of the most powerful possible dependent type theory. Then it will happen that the dependently typed λ-calculus of $\mathcal{G}_{\hat{F}}$ contains exactly the operations we need to construct the original logical relation without any fuss. The multi-page verification of Lemma 52 can be replaced by approximately two inches of text.

The idea to use the dependent type theory of a topos $\mathcal{G}_{\hat{F}}$ into which $\mathcal{G}_F$ embeds to define logical relations is named **synthetic Tait computability** [18, 16]. One way to think about synthetic Tait computability is that it leverages a *general* logical relation on topoi to construct *all* other logical relations; it turns out that it is considerably easier to establish this "general" logical relation on topoi than for any particular programming language, in part because topoi have certain advantageous structures available in them that are not available in more rarefied categories like the ones we have used for models of λ-calculus.

Changelog

1. October 15th, 2022
   (a) corrected the mistaken definition of $[\![-]\!]$ on terms, which failed to satisfy the law I imposed.
   (b) fixed errors in the definition of $\mathbf{HT}_{\sigma\Rightarrow\tau}$.
   (c) explained in more detail the well-definedness of $[\![\mathsf{if}]\!]$.
   (d) explained what the point of defining $[\![M]\!]$ is, given that its behavior is already forced by Specification 13; to this end, added discussion of proof relevant logical relations.
2. October 16th, 2022
   (a) pointed out that converse evaluation is needed for λ, not just if.
   (b) reformatted into chapters.
   (c) added product types to the first lecture.
   (d) started to write the second lecture.
3. October 17th, 2022
   (a) spelled out in Chapter 1 that the recursive definition is the FTLR.
   (b) spelled out proof of termination.
   (c) Chapter 2: spell out a bunch of basic category theory, and categorical definitions of weak product and weak function objects.
4. October 18th, 2022
   (a) replaced detour through contextual equivalence with application of Plotkin's standardization lemma.
   (b) fixed duplicate β-rule for products.
   (c) add boolean type structure to categorical semantics.
   (d) add discussion of soundness and completeness of cateogrical semantics.
   (e) added section on functorial semantics, homomorphisms of models
   (f) fixed typos in the definition of the boolean type structure
5. October 18th, 2022
   (a) got rid of obscure phrase "on the nose", clarify.

   (b) started working on a “general” logical relation.

6. October 21, 2022

   (a) fixed typos in Chapter 1

Acknowledgments

Thanks to Michael Arntzenius, Miëtek Bak, Derek Dreyer, Paolo Giarrusso, Andrew K. Hirsch, Matthew McQuaid, Max S. New, Nathan Sorenson, and Sam Tobin-Hochstadt their suggestions and corrections to earlier versions of this tutorial. Thanks to Bob Harper and Carlo Angiuli for many long talks about how to teach logical relations.

This work is funded by the European Union under the Marie Skłodowska-Curie Actions Postdoctoral Fellowship project *TypeSynth: synthetic methods in program verification*. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Bibliography

[1] Stuart Frazier Allen. “A Non-Type-Theoretic Definition of Martin-Löf’s Types”. In: *Proceedings of the Symposium on Logic in Computer Science (LICS ’87)*. Ithaca, NY: IEEE Computer Society, 1987, pp. 215–221.

[2] Thorsten Altenkirch and Ambrus Kaposi. “Normalisation by Evaluation for Dependent Types”. In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*. Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 6:1–6:16. ISBN: 978-3-95977-010-1. DOI: `10.4230/LIPIcs.FSCD.2016.6`. URL: `http://drops.dagstuhl.de/opus/volltexte/2016/5972`.

[3] Robert Atkey. “Relational Parametricity for Higher Kinds”. In: *Proceedings of the 21st EACSL Annual Conference / 26th International Workshop on Computer Science Logic*. Vol. 16. Leibniz International Proceedings in Informatics. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2012, pp. 46–61. ISBN: 978-3-939897-42-2. DOI: `10.4230/LIPIcs.CSL.2012.46`.

[4] Nick Benton, Martin Hofmann, and Vivek Nigam. “Proof-Relevant Logical Relations for Name Generation”. In: *Typed Lambda Calculi and Applications*. Ed. by Masahito Hasegawa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 48–60. ISBN: 978-3-642-38946-7.

[5] R. L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. New York: Cambridge University Press, 1993. ISBN: 978-0-521-45701-9.

[6] Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 2009.

[7] Marcelo Fiore. “Semantic Analysis of Normalisation by Evaluation for Typed Lambda Calculus”. In: *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP ’02. Pittsburgh, PA, USA: Association for Computing Machinery, 2002, pp. 26–37. ISBN: 1-58113-528-9. DOI: `10.1145/571157.571161`.

[8] Robert Harper. “Constructing Type Systems over an Operational Semantics”. In: *Journal of Symbolic Computation* 14.1 (July 1992), pp. 71–84. ISSN: 0747-7171.

[9] Robert Harper. ***PFPL** Supplement: How to (Re)Invent Tait’s Method*. Nov. 29, 2019. URL: `http://www.cs.cmu.edu/~rwh/pfpl/supplements/tait.pdf`.

[10] Achim Jung and Jerzy Tiuryn. "A new characterization of lambda definability". In: *Typed Lambda Calculi and Applications*. Ed. by Marc Bezem and Jan Friso Groote. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 245–257. ISBN: 978-3-540-47586-6.

[11] Paul Levy. *Call-by-Push-Value: A Functional/Imperative Synthesis*. Kluwer, Semantic Structures in Computation, 2, Jan. 1, 2003. ISBN: 1-4020-1730-8.

[12] Conor McBride and Ross Paterson. "Applicative Programming with Effects". In: *Journal of Functional Programming* 18.1 (Jan. 2008), pp. 1–13. ISSN: 0956-7968. DOI: 10.1017/S0956796807006326.

[13] Eugenio Moggi. "Notions of computation and monads". In: *Information and Computation* 93.1 (1991). Selections from 1989 IEEE Symposium on Logic in Computer Science, pp. 55–92. ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90052-4.

[14] Gordon D. Plotkin. "Call-by-Name, Call-by-Value and the lambda-Calculus". In: *Theor. Comput. Sci.* 1.2 (1975), pp. 125–159. DOI: 10.1016/0304-3975(75)90017-1.

[15] Andreas Rossberg, Claudio Russo, and Derek Dreyer. "F-ing modules". In: *Journal of Functional Programming* 24.5 (2014), pp. 529–607. DOI: 10.1017/S0956796814000264.

[16] Jonathan Sterling. "First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory". Version 1.1, revised May 2022. PhD thesis. Carnegie Mellon University, 2021. DOI: 10.5281/zenodo.6990769.

[17] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. "Cubical Syntax for Reflection-Free Extensional Equality". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 31:1–31:25. ISBN: 978-3-95977-107-8. DOI: 10.4230/LIPIcs.FSCD.2019.31. arXiv: 1904.08562 [cs.LO]. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10538.

[18] Jonathan Sterling and Robert Harper. "Logical Relations as Types: Proof-Relevant Parametricity for Program Modules". In: *Journal of the ACM* 68.6 (Oct. 2021). ISSN: 0004-5411. DOI: 10.1145/3474834. arXiv: 2010.08599 [cs.PL].

[19] W. W. Tait. "Intensional Interpretations of Functionals of Finite Type I". In: *The Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. ISSN: 00224812. URL: http://www.jstor.org/stable/2271658.