Under consideration for publication in J. Functional Programming

THEORETICAL PEARLS

Correctness of Compiling Polymorphism to Dynamic Typing

Kuen-Bang Hou (Favonia)∗
Carnegie Mellon University
(*e-mail:* favonia@cs.cmu.edu)

Nick Benton
Microsoft Research
(*e-mail:* nick@microsoft.com)

Robert Harper∗
Carnegie Mellon University
(*e-mail:* rwh@cs.cmu.edu)

Abstract

The connection between polymorphic and dynamic typing was originally considered by Curry *et al.* (1972) in the form of "polymorphic type assignment" for untyped λ-terms. Types are assigned after the fact to what is, in modern terminology, a dynamic language. Interest in type assignment was revitalized by the proposals of Bracha *et al.* (1998) and Bank *et al.* (1997) to enrich Java with polymorphism (generics), which in turn sparked the development of other languages, such as Scala, with similar combinations of features. In such a setting, where the target language already has a monomorphic type system, it is desirable to compile polymorphism to dynamic typing in such a way that as much static typing as possible is preserved, relying on dynamics only insofar as genericity is actually required.

The basic approach is to compile polymorphism using embeddings from each type into a universal 'top' type, $\mathbb{D}$, and partial projections that go in the other direction. This scheme is intuitively reasonable, and, indeed, has been used in practice many times (Igarashi *et al.*, 2001). Proving its correctness, however, is non-trivial. This paper studies the compilation of System F to an extension of Moggi's computational metalanguage with a dynamic type and shows how the compilation may be proved correct using a logical relation.

∗ This research is sponsored in part by the National Science Foundation under Grant Number 1116703. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

1.1 Polymorphism and Dynamic Typing

The connection between polymorphism and dynamic typing was first explored by Curry *et al.* (1972) under the name *polymorphic type assignment*. Types are assigned to untyped λ-terms using rules such as these:

$$\frac{M : A}{M : \forall X.A} \qquad \frac{M : \forall X.A}{M : A[B/X]}$$

According to this view, a term, M, already has an operational meaning, and types are assigned *after* the fact to express some aspects of their behavior. For example, the expression $\lambda x.x$ can be assigned with the types $\mathbb{N} \to \mathbb{N}$, $\mathbb{B} \to \mathbb{B}$ (booleans to booleans), or more generally $\forall X.X \to X$. This approach to typing inspired much work on programming languages, including the original formulation of the ML type system by Milner (1978).

The type assignment viewpoint, however, is not the only possible way to understand polymorphism in programming languages. The Girard-Reynolds polymorphic typed λ-calculus, known as System F (Girard, 1972; Reynolds, 1974), expresses a subtly different concept characterized by these rules:

$$\frac{M : A}{\Lambda X.M : \forall X.A} \qquad \frac{M : \forall X.A}{M\,B : A[B/X]}$$

Here type abstraction and application are an explicit part of the construction of the term. More generally, according to this *intrinsic* view, types are seen as defining what terms exist, rather than describing the behavior of pre-existing terms. See Figure 1 for comparison between two regimes.

Polymorphism in System F (Source)	Dynamic typing (Target)
Before operational semantics	*After* operational semantics
Church style	Curry style
Type checking	Type assignments
Intrinsic typing	Extrinsic typing

Fig. 1. Polymorphism and dynamic typing

A key difference between the two approaches lies in their execution behavior when endowed with an operational semantics. Under a type assignment regime the programs are given independently of any type information. Such programs may be seen as dynamically typed, which, in the presence of multiple classes of data, implies run-time overhead to express and enforce proper classification of values. In contrast, under a type checking regime the formation of programs and their execution behavior are influenced by, and in some cases determined by, the types involved. This leads to better execution behavior, because it avoids the overhead of dynamic class checks from the outset, and hence is more amenable to modular compilation and composition.

1.2 Compilation of Polymorphism

Interest in polymorphic type assignment was revitalized by the proposals of Bracha *et al.* (1998) and Bank *et al.* (1997) to enrich Java with polymorphism ("generics"), which then inspired similar treatment of generics in languages such as C♯ and Scala. Such languages are statically typed, but feature a universal type (`Object` in Java, but herein called $\mathbb{D}$) of dynamically typed values. The question arose as to how to compile these extensions, given that little or no change could be made to the language's established monomorphic run-time structure. Abstracting from the language-specific details, the question may be re-phrased as:

> How to compile System F to a simply-typed language **D** with a type of dynamically typed values while preserving static type information as much as possible?

Classical type assignment effectively erases static types, mapping everything to the universal type $\mathbb{D}$, which is unsatisfactory. We would rather, for example, translate the monomorphic doubling function $\lambda x{:}\mathtt{nat}.x+x$ to essentially the same typed code $\lambda x{:}\mathbb{N}.x+x$ in the target, reserving dynamic typing, and its associated run-time costs, to the translation of code that actually uses polymorphism. The ideal translation of a System F type A into a **D** type $A^\dagger$ will preserve the structure of A, except that source language type variables, X, will be mapped to the target type $\mathbb{D}$. This immediately raises the question of how to relate $(A[B/X])^\dagger$ to $A^\dagger$, which is to say how to manage polymorphic instantiation. Indeed, this is the heart of the translation given by the aforementioned authors.

The translation relies on the existence of an embedding, i, of each **D** type into the type $\mathbb{D}$, equipped with a corresponding projection, j, which recovers the embedded object, which is to say that j is post-inverse (left-inverse) to i, up to observational equivalence, $j \circ i \cong \mathsf{id}$. In the case of compiling to the JVM, i would be realized by an upcast to `Object` and j by a (possibly failing) downcast from `Object`. Notice that j is not also pre-inverse (right-inverse) to i, that is, $i \circ j \ncong \mathsf{id}$, because there is no reason to expect that an arbitrary value of type $\mathbb{D}$ lies in the image of the embedding i. In order-theoretic terms, every **D** type is a retract of $\mathbb{D}$, with retraction given by the idempotent composition $i \circ j : \mathbb{D} \to \mathbb{D}$.

The embedding of every **D** type into $\mathbb{D}$ lifts functorially into an embedding, I, from $(A[B/X])^\dagger$ to $A^\dagger$, accompanied by a corresponding projection, J, going in the other direction. These lifted embeddings and projections are used to mediate polymorphic instantiation. Consider the polymorphic identity function $\Lambda X.\lambda x{:}X.x$ of type $\forall X.X \to X$ in System F which is then translated to $\lambda x{:}\mathbb{D}.x$ of type $\mathbb{D} \to \mathbb{D}$ in the target language. An instantiation of this polymorphic function at the type $\mathbb{N}$ is translated to the following function of type $\mathbb{N} \to \mathbb{N}$:

$$\lambda x{:}\mathbb{N}.j_{\mathtt{nat}}((\lambda x{:}\mathbb{D}.x)(i_{\mathtt{nat}}(x))),$$

where $i_{\mathtt{nat}}$ and $j_{\mathtt{nat}}$ are, respectively, the embedding and the projection for $\mathbb{N}$. The pre- and post-compositions with the embeddings and projections arise from the functorial action of the type constructor $X \to X$, thought of as a function of X. The projection to type $\mathbb{N} \to \mathbb{N}$ requires that we embed the argument into $\mathbb{D}$, execute the translation of the polymorphic identity, and project the result back to $\mathbb{N}$. This function is observationally equivalent to the

identity on $\mathbb{N}$, because the context may only provide natural numbers as arguments, and expect natural numbers as results.

1.3 Our Contribution: Correctness Proof

At a very high level, the form of our proof is that of an adequacy theorem for a paradigmatic denotational metalanguage with dynamic typing (which we call **D**) with respect to an operational semantics (represented by conversion rules) of a paradigmatic polymorphic calculus (which is System F).

Using embeddings and projections, as sketched above, we can give a straightforward translation of System F into **D**. The goal of the correctness proof is to show that an expression and its compilation are appropriately related. The contribution of this work is in the method of proof. In the literature we identified two relevant results, neither of which are readily applicable to the present problem:

- Meyer & Wand (1985) give a logical relation argument for correctness of CPS translation for the simply-typed lambda calculus, but our projection j is not an pre-inverse (right-inverse) of i as in their work.
- Igarashi *et al.* (2001) show the correctness of compiling generics in (core) Java, but their treatment seems inextricable from the source language, Featherweight Java, which involves a number of object-oriented concepts such as a class table.

Here we present a carefully formulated parametric logical relation that directly relates terms with their translations, together with a key lemma that captures the way in which the relation respects the embeddings and projections. This is, to our knowledge, the first correctness proof of this method of compiling polymorphism in System F to dynamic typing. Compared to the bisimulation theorem by Igarashi *et al.* (2001), our logical relation (*cf.* Lemma 7) additionally permits foreign functions as long as they follow the embedding and projection invariants specified in the logical relation, which may be seen as a technical advantage.

2 Languages

Throughout the paper we work modulo α-conversion. So bound variables are assumed to be renamed if collision would happen in substitutions, and variables appearing in contexts are always distinct.

2.1 Source Language

Our source language is System F, the Girard-Reynolds polymorphic lambda calculus. As in Girard's original formulation, we include a base type of natural numbers. In our case they provide the observable outcomes that are used to distinguish programs. The syntax, the typing rules, and the conversion rules are shown in Figure 2. We say a type A is *closed* if $\cdot \vdash A$. The calculus is presented with β-conversions ($\equiv_\beta$), a thin abstraction over reductions or the operational semantics. It is compatible with both the call-by-value and call-by-name reductions, or any reasonable operational semantics because the calculus is strongly normalizing.

Syntax

(x is a variable name.)

Types	$A,B ::= X \mid \mathtt{nat} \mid A \to B \mid \forall X.A$
Terms	$M,N ::= x \mid \mathtt{z} \mid \mathtt{suc}(M) \mid \mathtt{ifz}(M;N_0;x.N_1) \mid \lambda x{:}A.M \mid MN \mid \Lambda X.M \mid MA$
Type Context	$\Delta ::= \cdot \mid \Delta,X$
Term Context	$\Gamma ::= \cdot \mid \Gamma,x{:}A$

$\boxed{\Delta \vdash A}$

$$\Delta,X,\Delta' \vdash X \qquad \Delta \vdash \mathtt{nat} \qquad \frac{\Delta \vdash A \qquad \Delta \vdash B}{\Delta \vdash A \to B} \qquad \frac{\Delta,X \vdash A}{\Delta \vdash \forall X.A}$$

$\boxed{\Delta;\Gamma \vdash M : A}$

$$\Delta;\Gamma,x{:}A,\Gamma' \vdash x : A \qquad \Delta;\Gamma \vdash \mathtt{z} : \mathtt{nat} \qquad \frac{\Delta;\Gamma \vdash M : \mathtt{nat}}{\Delta;\Gamma \vdash \mathtt{suc}(M) : \mathtt{nat}}$$

$$\frac{\Delta;\Gamma \vdash M : \mathtt{nat} \qquad \Delta;\Gamma \vdash N_0 : A \qquad \Delta;\Gamma,x{:}\mathtt{nat} \vdash N_1 : A}{\Delta;\Gamma \vdash \mathtt{ifz}(M;N_0;x.N_1) : A} \qquad \frac{\Delta;\Gamma \vdash M : A \to B \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash MN : B}$$

$$\frac{\Delta;\Gamma,x{:}A \vdash M : B}{\Delta;\Gamma \vdash \lambda x{:}A.M : A \to B} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta,X \vdash A \qquad \Delta,X;\Gamma \vdash M : A}{\Delta;\Gamma \vdash \Lambda X.M : \forall X.A} \qquad \frac{\Delta;\Gamma \vdash M : \forall X.A \qquad \Delta \vdash B}{\Delta;\Gamma \vdash MB : A[B/X]}$$

Conversions

$$(\Lambda X.M)\,B \equiv_\beta M[B/X]$$
$$(\lambda x{:}A.M)\,N \equiv_\beta M[N/x]$$
$$\mathtt{ifz}(\mathtt{z};N_0;x.N_1) \equiv_\beta N_0$$
$$\mathtt{ifz}(\mathtt{suc}(M);N_0;x.N_1) \equiv_\beta N_1[M/x]$$

Fig. 2. Syntax, type rules, and conversion rules for System F

2.2 Target Language

Our source language is pure and strongly normalizing, but the target language has to have some effects. Firstly, the presence of a universal type that will essentially be a model of an untyped lambda calculus means, because one can express fixed point combinators, that the target has to include non-terminating expressions. Secondly, there needs to be some error mechanism in the target, which we can invoke when projections from the universal type should fail.

Once the target has side-effects, we have to decide how eager to make the translation. For System F, any sensible translation will have a corresponding correctness theorem that shows, amongst other things, that the translation of a closed source program actually never exhibits any effects, but different translations can nevertheless translate a source term into ones that can be distinguished in the target. Here we choose to work with a call-by-value

translation, as that is what one would want to use in the most common real-world situation, in which the source language also has some effects, and has a call-by-value semantics.

We could take the target to be a monomorphic ML-like language with a particular universal type and notion of error. That would work out perfectly well, but we instead translate directly into a slightly more explicit metalanguage for the *semantics* of such a language, namely a version of Moggi's (1991) computational metalanguage, λML_T. The computational metalanguage is a simply-typed lambda calculus with a type constructor, $\mathrm{T}(\cdot)$, corresponding to a strong monad with an injective unit; we further add a universal type and errors. See Figure 3 for the relevant fragment of its syntax, typing rules, and equations.

The equations for **D** should be understood as real denotational equalities. The source language System F is treated more syntactically: the translation is defined structurally on actual terms (just modulo α-conversion) and we only later show that it respects β-conversions in the source language as a separate lemma. (We say a relation is *admissible* if it respects β-conversions (Equation (1) on page 12), and the admissibility of the translation relation is stated as Lemma 4.)

Taking **D** to be the computational metalanguage is largely a matter of taste, but has some advantages. We will be doing a great deal of equational reasoning and, unlike a call-by-value lambda calculus, λML_T satisfies unrestricted β and η laws. The fastidious distinction between value types, D, and computation types, $\mathrm{T}(D)$, means that the type system makes it clear where there is a possibility of an error or divergence and where there is not, so various erroneous definitions one might make simply will not typecheck. And finally, we can be generic in exactly what the monad is—the proof will work for any $\mathrm{T}(\cdot)$ that satisfies the equations we use.

As mentioned above, the monad is used to account for the possibility of divergence that is forced by the presence of a universal type, and also for the runtime errors that should arise, for example, when one injects a function value into the universal type and then attempts to project (cast) it back out as a natural number. We will require that $\mathrm{T}(\cdot)$ comes equipped with a polymorphic constant $\mathtt{err} : \mathrm{T}(D)$ for any D, but there are many concrete examples of monads that will suit our purposes. For example, in the category of ω-cpos (predomains):

1. Take $\mathrm{T}(D) = D_\perp$, the lifting monad, and $\mathtt{err} = \perp$, so dynamic errors are just modelled by divergence.
2. Take $\mathrm{T}(D) = (1+D)_\perp$, the lifted error (maybe, option) monad, and $\mathtt{err} = [\mathtt{inl}(*)]$, so dynamic errors are modelled by a terminating, failing computation.

But everything that follows works for any monad that satisfies our conditions. We write $[\cdot]$ for the unit of the monad and also abbreviate the usual monadic bind construct $\mathtt{let}\ x \Leftarrow d\ \mathtt{in}\ e$ by just $x \leftarrow d; e$.

The target language **D** has a base type $\mathbb{N}$ for the natural numbers. If $n : \mathbb{N}$ is a numeral, we write $\overline{n}$ for the corresponding System F normal form $\mathtt{suc}(\ldots(\mathtt{suc}(\mathtt{z}))\ldots)$.

The language **D** also has a universal type $\mathbb{D}$ equipped with operations

$$\mathtt{num}\,(d : \mathbb{N}) : \mathbb{D} \qquad \mathtt{num?}\,(d : \mathbb{D}) : \mathrm{T}(\mathbb{N})$$
$$\mathtt{fun}\,(d : \mathbb{D} \to \mathrm{T}(\mathbb{D})) : \mathbb{D} \qquad \mathtt{fun?}\,(d : \mathbb{D}) : \mathrm{T}(\mathbb{D} \to \mathrm{T}(\mathbb{D}))$$

Syntax

(x is a variable name.)

Types $D,E ::= \mathbb{N} \mid D \to E \mid \mathrm{T}(D) \mid \mathbb{D} \mid \ldots$

Classes $c ::= \mathtt{num} \mid \mathtt{fun} \mid \ldots$

Terms $d,e ::= x \mid 0 \mid d+1 \mid \mathtt{ifz}(d;e_0;x.e_1) \mid \lambda x{:}D.d \mid d\,e \mid [d] \mid \mathtt{err} \mid x \leftarrow d;e \mid c\,(d) \mid c?\,(d) \mid \ldots$

Contexts $\Gamma ::= \cdot \mid \Gamma, x{:}D$

$\Gamma \vdash d : D$

$$\Gamma, x{:}D, \Gamma' \vdash x : D \qquad \Gamma \vdash 0 : \mathbb{N} \qquad \frac{\Gamma \vdash d : \mathbb{N}}{\Gamma \vdash d+1 : \mathbb{N}} \qquad \frac{\Gamma \vdash d : \mathbb{N} \qquad \Gamma \vdash e_0 : D \qquad \Gamma, x{:}\mathbb{N} \vdash e_1 : D}{\Gamma \vdash \mathtt{ifz}(d;e_0;x.e_1) : D}$$

$$\frac{\Gamma, x{:}D \vdash d : E}{\Gamma \vdash \lambda x{:}D.d : D \to E} \qquad \frac{\Gamma \vdash d : D \to E \qquad \Gamma \vdash e : D}{\Gamma \vdash d\,e : E} \qquad \frac{\Gamma \vdash d : D}{\Gamma \vdash [d] : \mathrm{T}(D)} \qquad \Gamma \vdash \mathtt{err} : \mathrm{T}(D)$$

$$\frac{\Gamma \vdash d : \mathrm{T}(D) \qquad \Gamma, x{:}D \vdash e : \mathrm{T}(E)}{\Gamma \vdash x \leftarrow d;e : \mathrm{T}(E)} \qquad \frac{\Gamma \vdash d : \mathbb{N}}{\Gamma \vdash \mathtt{num}\,(d) : \mathbb{D}} \qquad \frac{\Gamma \vdash d : \mathbb{D}}{\Gamma \vdash \mathtt{num}?\,(d) : \mathrm{T}(\mathbb{N})}$$

$$\frac{\Gamma \vdash d : \mathbb{D} \to \mathrm{T}(\mathbb{D})}{\Gamma \vdash \mathtt{fun}\,(d) : \mathbb{D}} \qquad \frac{\Gamma \vdash d : \mathbb{D}}{\Gamma \vdash \mathtt{fun}?\,(d) : \mathrm{T}(\mathbb{D} \to \mathrm{T}(\mathbb{D}))} \qquad \ldots$$

Equations

$$\begin{aligned}
(\lambda x{:}D.d)\,e &= d[e/x] \\
x \notin \mathrm{FV}(d) &\implies (\lambda x{:}D.d\,x) = d \\
\mathtt{ifz}(0;d_0;x.d_1) &= d_0 \\
\mathtt{ifz}(n+1;d_0;x.d_1) &= d_1[n/x] \\
\mathtt{ifz}(d;e\,0;x.e\,(x+1)) &= e\,d \\
x \leftarrow [d];e &= e[d/x] \\
x \leftarrow d;[x] &= d \\
x_2 \leftarrow (x_1 \leftarrow d_1;d_2);e &= x_1 \leftarrow d_1;x_2 \leftarrow d_2;e \\
[d] = [e] &\implies d = e \\
c?\,(c\,(d)) &= [d] \\
c \neq c' &\implies c?\,(c'\,(d)) = \mathtt{err} \\
&\ldots
\end{aligned}$$

Fig. 3. Syntax, type rules, and equations for the relevant fragment of the metalanguage **D**

subject to the equations listed in Figure 3. We know these requirements are consistent, as they can be canonically satisfied by taking $\mathbb{D}$ to be the least solution to the recursive predomain equation

$$\mathbb{D} \cong \mathrm{T}(\mathbb{N} + (\mathbb{D} \to \mathrm{T}(\mathbb{D})))$$

$$\boxed{A^\dagger}$$

$$\begin{aligned}
\mathtt{nat}^\dagger &= \mathbb{N} \\
X^\dagger &= \mathbb{D} \\
(A \to B)^\dagger &= A^\dagger \to \mathrm{T}(B^\dagger) \\
(\forall X.A)^\dagger &= A^\dagger
\end{aligned}$$

$$\boxed{\Gamma^\dagger}$$

$$\begin{aligned}
\cdot^\dagger &= \cdot \\
(\Gamma, x{:}A)^\dagger &= \Gamma^\dagger, x{:}A^\dagger
\end{aligned}$$

Fig. 4. Translation of types and contexts

with

$$\begin{aligned}
\mathtt{num}(d) &= \mathtt{roll}([\mathtt{inl}(d)]) \\
\mathtt{num?}(d) &= d' \leftarrow \mathtt{unroll}(d); \mathtt{case}\ d'\ \mathtt{of}\ \mathtt{inl}(n) \Rightarrow [n] \mid \mathtt{inr}(_) \Rightarrow \mathtt{err} \\
\mathtt{fun}(d) &= \mathtt{roll}([\mathtt{inr}(d)]) \\
\mathtt{fun?}(d) &= d' \leftarrow \mathtt{unroll}(d); \mathtt{case}\ d'\ \mathtt{of}\ \mathtt{inl}(_) \Rightarrow \mathtt{err} \mid \mathtt{inr}(e) \Rightarrow [e]
\end{aligned}$$

where $\mathtt{roll}(\cdot)$ and $\mathtt{unroll}(\cdot)$ are the components of the isomorphism in the solution of the equation for $\mathbb{D}$. However, nothing that follows relies on any domain theory: we just need the equations.

It is interesting to observe that the correctness of the translation does not actually *require* any interesting properties of errors. In particular, we do not need to specify that $\mathtt{err}$ is natural, that the monad is strict in errors (i.e. that $x \leftarrow \mathtt{err}; d = \mathtt{err}$), or even that errors are disjoint from values ($\forall d, \mathtt{err} \neq [d]$), though these properties do hold for our examples of concrete monads. Indeed, one could remove errors entirely, replacing them with arbitrary default values, without materially affecting what follows. The reason for this is that the correctness theorem only talks about error-free behaviour—if everything in the context is error-free then the translated term is also error-free—so the precise nature of errors is not very important. But in a more practical setting, one would want to use a well-structured error mechanism.

3 Translation

The translation and interpretation of types follows Moggi's (1991) call-by-value translation, with type variables interpreted as the universal type, $\mathbb{D}$. This is shown in Figure 4.

3.1 Embeddings and Projections

Before we can define the translation of terms, we need some auxiliary definitions on the target side. First we have an embedding, i, and a projection, j, mapping between $A^\dagger$ and $\mathbb{D}$ for each source type A. The definitions are shown in Figure 5. Note that the

$$\boxed{i_A : A^\dagger \to \mathbb{D}} \text{ and } \boxed{j_A : \mathbb{D} \to \mathrm{T}(A^\dagger)}$$

$$\begin{aligned}
i_{\mathtt{nat}}(x) &= \mathtt{num}(x) \\
j_{\mathtt{nat}}(x) &= \mathtt{num?}(x) \\
i_X(x) &= x \\
j_X(x) &= [x] \\
i_{A\to B}(f) &= \mathtt{fun}(\lambda d{:}\mathbb{D}.v \leftarrow j_A(d); r \leftarrow f\,v; [i_B(r)]) \\
j_{A\to B}(f) &= f' \leftarrow \mathtt{fun?}(f); [\lambda a{:}A^\dagger.r \leftarrow f'(i_A(a)); j_B(r)] \\
i_{\forall X.A}(x) &= i_A(x) \\
j_{\forall X.A}(x) &= j_A(x)
\end{aligned}$$

Fig. 5. Embeddings and projections

embedding is total (any value of type $A^\dagger$ can be mapped into the universal domain), although the projection is partial, which is why the monad appears in the return type. Only well-behaved elements of $\mathbb{D}$ may be mapped back to $A^\dagger$; projecting an ill-behaved value may fail immediately or, in the case of function types, when the projected value is later actually applied.

An embedding followed by the corresponding projection is always morally the identity (actually, the unit of the monad):

Lemma 1

For any System F type A and **D** term $x : A^\dagger$,

$$j_A(i_A(x)) = [x].$$

Proof

Induction on the structure of A.

- Case `nat`:

$$j_{\mathtt{nat}}(i_{\mathtt{nat}}(x)) = \mathtt{num?}(\mathtt{num}(x)) = [x]$$

- Case X:

$$j_X(i_X(x)) = [x]$$

- Case $A \to B$:

$$\begin{aligned}
&j_{A\to B}(i_{A\to B}(f)) \\
&= f' \leftarrow \mathtt{fun?}(\mathtt{fun}\,(\lambda d{:}\mathbb{D}.v \leftarrow j_A(d); r \leftarrow f\,v; [i_B(r)])); [\lambda a{:}A^\dagger.r \leftarrow f'(i_A(a)); j_B(r)] \\
&= f' \leftarrow [\lambda d{:}\mathbb{D}.v \leftarrow j_A(d); r \leftarrow f\,v; [i_B(r)]]; [\lambda a{:}A^\dagger.r \leftarrow f'(i_A(a)); j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow (\lambda d{:}\mathbb{D}.v \leftarrow j_A(d); r \leftarrow f\,v; [i_B(r)])\,(i_A(a)); j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow (v \leftarrow j_A(i_A(a)); r \leftarrow f\,v; [i_B(r)]); j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow (v \leftarrow [a]; r \leftarrow f\,v; [i_B(r)]); j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow (r \leftarrow f\,a; [i_B(r)]); j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow f\,a; r \leftarrow [i_B(r)]; j_B(r)] \\
&= [\lambda a{:}A^\dagger.r \leftarrow f\,a; j_B(i_B(r))] \\
&= [\lambda a{:}A^\dagger.r \leftarrow f\,a; [r]] \\
&= [\lambda a{:}A^\dagger.f\,a] \\
&= [f]
\end{aligned}$$

- Case $\forall X.A$:

$$j_{\forall X.A}(i_{\forall X.A}(x)) = j_A(i_A(x)) = [x]$$

□

3.2 Lifted Embeddings and Projections

The translation $A^\dagger$ of a type A with a free type variable X has $\mathbb{D}$ in positions corresponding to the occurrences of X in A. Type application in the source involves substitution of a type B for those occurrences of X; translating the application requires the use of functions $J^B_{X.A}$ from $A^\dagger$ to $\mathrm{T}(A[B/X]^\dagger)$, the monad applied to the translation of the substituted type. The result is wrapped in the monad because $\mathbb{D}$-values produced by the argument in places corresponding to positive occurrences of X in A are not necessarily well-behaved. Just as with the embeddings and projections of the previous section, the definition of $J^B_{X.A}$ is not only inductive on A, but mutually inductive with that of a function going in the other direction, $I^B_{X.A}$ from $A[B/X]^\dagger$ to $A^\dagger$. The definitions are shown in Figure 6.

The following is a lifted version of Lemma 1.

Lemma 2
If $\Delta, X \vdash A$ and $\Delta \vdash B$, then for any **D** term $d : A^\dagger$,

$$J^B_{X.A}(I^B_{X.A}(d)) = [d].$$

Proof
Induction on the structure of A.

- Case $X.X$:

$$J^B_{X.X}(I^B_{X.X}(d)) = j_B(i_B(d)) = [d]$$

- Case $X.Y$ (where X and Y are different variables):

$$J^B_{X.Y}(I^B_{X.Y}(d)) = J^B_{X.Y}(d) = [d]$$

$\boxed{I^B_{X.A} : (A[B/X])^\dagger \to A^\dagger}$ and $\boxed{J^B_{X.A} : A^\dagger \to \mathrm{T}(A[B/X]^\dagger)}$

$$
\begin{aligned}
I^B_{X.X}(x) &= i_B(x) \\
J^B_{X.X}(x) &= j_B(x) \\
I^B_{X.Y}(x) &= x \qquad (X \text{ and } Y \text{ are different variables}) \\
J^B_{X.Y}(x) &= [x] \qquad (X \text{ and } Y \text{ are different variables}) \\
I^B_{X.\mathtt{nat}}(x) &= x \\
J^B_{X.\mathtt{nat}}(x) &= [x] \\
I^B_{X.A_1 \to A_2}(f) &= \lambda a{:}A_1^\dagger.a' \leftarrow J^B_{X.A_1}(a); r \leftarrow f\,a'; [I^B_{X.A_2}(r)] \\
J^B_{X.A_1 \to A_2}(f) &= [\lambda a{:}(A_1[B/X])^\dagger.r \leftarrow f(I^B_{X.A_1}(a)); J^B_{X.A_2}(r)] \\
I^B_{X.\forall Y.A}(x) &= I^B_{X.A}(x) \\
J^B_{X.\forall Y.A}(x) &= J^B_{X.A}(x)
\end{aligned}
$$

Fig. 6. Lifted embeddings and projections

- Case $X.\mathtt{nat}$:

$$J^B_{X.\mathtt{nat}}(I^B_{X.\mathtt{nat}}(d)) = J^B_{X.\mathtt{nat}}(d) = [d]$$

- Case $X.A_1 \to A_2$:

$$
\begin{aligned}
& J^B_{X.A_1 \to A_2}(I^B_{X.A_1 \to A_2}(f)) \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow I^B_{X.A_1 \to A_2}(f)(I^B_{X.A_1}(a)); J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow (\lambda a{:}A_1^\dagger.a' \leftarrow J^B_{X.A_1}(a); r \leftarrow f\,a'; [I^B_{X.A_2}(r)])(I^B_{X.A_1}(a)); J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow (a' \leftarrow J^B_{X.A_1}(I^B_{X.A_1}(a)); r \leftarrow f\,a'; [I^B_{X.A_2}(r)]); J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow (a' \leftarrow [a]; r \leftarrow f\,a'; [I^B_{X.A_2}(r)]); J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow (r \leftarrow f\,a; [I^B_{X.A_2}(r)]); J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow f\,a; r \leftarrow [I^B_{X.A_2}(r)]; J^B_{X.A_2}(r)] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow f\,a; J^B_{X.A_2}(I^B_{X.A_2}(r))] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.r \leftarrow f\,a; [r]] \\
=& [\lambda a{:}(A_2[B/X])^\dagger.f\,a] \\
=& [f]
\end{aligned}
$$

- Case $X.\forall Y.A$:

$$J^B_{X.\forall Y.A}(I^B_{X.\forall Y.A}(x)) = J^B_{X.A}(I^B_{X.A}(x)) = [x]$$

□

$$\boxed{(\Delta;\Gamma \vdash M : A)^*}$$

$$\begin{aligned}
(\Delta;\Gamma,x{:}A \vdash x : A)^* &= [x] \\
(\Delta;\Gamma \vdash \mathtt{z} : \mathtt{nat})^* &= [0] \\
(\Delta;\Gamma \vdash \mathtt{suc}(M) : \mathtt{nat})^* &= d \leftarrow (\Delta;\Gamma \vdash M : \mathtt{nat})^*; [d+1] \\
(\Delta;\Gamma \vdash \mathtt{ifz}(M;N_0;x.N_1) : A)^* &= d \leftarrow (\Delta;\Gamma \vdash M : \mathtt{nat})^*; \\
&\qquad \mathtt{ifz}(d;(\Delta;\Gamma \vdash N_0 : A)^*;x.(\Delta;\Gamma,x{:}\mathtt{nat} \vdash N_1 : A)^*) \\
(\Delta;\Gamma \vdash \lambda x{:}A.M : A \to B)^* &= [\lambda x{:}A^\dagger.(\Delta;\Gamma,x{:}A \vdash M : B)^*] \\
(\Delta;\Gamma \vdash MN : B)^* &= e \leftarrow (\Delta;\Gamma \vdash M : A \to B)^*; d \leftarrow (\Delta;\Gamma \vdash N : A)^*; e\,d \\
(\Delta;\Gamma \vdash \Lambda X.M : \forall X.A)^* &= (\Delta,X;\Gamma \vdash M : A)^* \\
(\Delta;\Gamma \vdash M\,B : A[B/X])^* &= d \leftarrow (\Delta;\Gamma \vdash M : \forall X.A)^*; J^B_{X.A}(d)
\end{aligned}$$

Fig. 7. Term translation

3.3 Term Translation

Just as was the case for types, the translation $(\cdot)^*$ of terms in context is Moggi's usual CBV translation, extended to use $J^B_{X.A}$ to translate type application. The formal definition is shown in Figure 7.

Note that uniqueness of typing in the source language ensures that the type A appearing on the right hand side of the application case is uniquely determined, so this is indeed a good definition. It is also appropriately typed:

Lemma 3
If $\Delta;\Gamma \vdash M : A$ then $\Gamma^\dagger \vdash (\Delta;\Gamma \vdash M : A)^* : \mathrm{T}(A^\dagger)$. □

4 Logical Relation

If B is a closed source type, write $\mathtt{CT}(B) = \{M \mid \cdot;\cdot \vdash M : B\}$ for the set of closed terms of type B. Given $\mathcal{R} \subseteq \mathtt{CT}(B) \times \mathbb{D}$, a relation between closed source terms of type B and elements of $\mathbb{D}$, then we say $\mathcal{R}$ is *admissible* if it respects the equivalence of the source language; that is

$$(M,d) \in \mathcal{R} \ \wedge\ M \equiv_\beta M' \implies (M',d) \in \mathcal{R}. \tag{1}$$

We write $\Delta \vdash w$ to mean that the *type environment* w is a map from the finite set of type variables Δ to pairs comprising a closed type and an admissible relation on that type. Formally

$$w : \Delta \to \mathcal{F}, \text{ where } \mathcal{F} = \sum_{\{B \mid \cdot \vdash B\}} \{\mathcal{R} \subseteq \mathtt{CT}(B) \times \mathbb{D} \mid \mathcal{R} \text{ is admissible}\}.$$

If $\Delta = \cdot, X_1, \ldots, X_n$ and $w(X_j) = (B_j, \mathcal{R}_j)$ for each $1 \le j \le n$, then we define the relations $\mathcal{R}^w_A \subseteq \mathtt{CT}(A[B_j/X_j]) \times A^\dagger$ and $\mathcal{TR}^w_A \subseteq \mathtt{CT}(A[B_j/X_j]) \times \mathrm{T}(A^\dagger)$, for each A such that $\Delta \vdash A$, by mutual induction on A as shown in Figure 8. The relation $\mathcal{TR}^w_A$ is a particular choice of "monadic lifting" of the relation $\mathcal{R}^w_A$. Figure 8 also defines the shorthand $\widehat{\mathcal{R}}^w_A$, which relates source terms to target values of type $\mathbb{D}$. We will have $(A[B_j/B_j], \widehat{\mathcal{R}}^w_A) \in \mathcal{F}$.

$$\boxed{\mathcal{R}^w_A \subseteq \mathtt{CT}(A[B_j/X_j]) \times A^\dagger} \text{ and } \boxed{\mathcal{TR}^w_A \subseteq \mathtt{CT}(A[B_j/X_j]) \times \mathrm{T}(A^\dagger)}$$

$$\begin{aligned}
\mathcal{R}^w_{X_i} &= \mathcal{R}_i \qquad (\text{where } w(X_i) = (B_i, \mathcal{R}_i)) \\
\mathcal{R}^w_{\mathtt{nat}} &= \{(M,n) \mid M \equiv_\beta \overline{n}\} \\
\mathcal{R}^w_{A_1 \to A_2} &= \{(M,d) \mid M \equiv_\beta \lambda x{:}A_1[B_j/X_j].M' \wedge \\
&\qquad \forall (M_2,d_2) \in \mathcal{R}^w_{A_1}, (M'[M_2/x], d\,d_2) \in \mathcal{TR}^w_{A_2}\} \\
\mathcal{R}^w_{\forall X.A} &= \{(M,d) \mid M \equiv_\beta \Lambda X.M' \wedge \forall (B,\mathcal{R}) \in \mathcal{F}, (M'[B/X], d) \in \mathcal{R}^{w,X \mapsto (B,\mathcal{R})}_A\} \\
\mathcal{TR}^w_A &= \{(M,[d]) \mid (M,d) \in \mathcal{R}^w_A\}
\end{aligned}$$

$$\boxed{\widehat{\mathcal{R}}^w_A \subseteq \mathtt{CT}(A[B_j/X_j]) \times \mathbb{D}}$$

$$\widehat{\mathcal{R}}^w_A = \{(M,d) \mid (M, j_A(d)) \in \mathcal{TR}^w_A\}$$

Fig. 8. Logical relations

Observe that, as in previous work on relationally parametric models of polymorphism (Reynolds, 1983), the clause for polymorphic types involves quantification over *all* relations from a pre-defined set. This enforces parametricity and also avoids the potential circularity due to impredicativity, which would arise were one to consider instantiating just with $\mathcal{R}^w_B$ for each B; an instance of $\forall X.A$, say $A[B/X]$, could be "larger" than $\forall X.A$ and break the naive induction ordering. A more detailed discussion about the potential impredicativity issues can be found in Chapter 48 of the third author's text (Harper, 2012).

Lemma 4 (*Admissibility*)
For all w and A, $\mathcal{R}^w_A$ and $\mathcal{TR}^w_A$ are admissible. □

Lemma 5 (*Weakening*)
If $\Delta \vdash A$ and $\Delta \vdash w$ then for any B and $\mathcal{R}$,

$$\mathcal{R}^{w,X \mapsto (B,\mathcal{R})}_A = \mathcal{R}^w_A \quad \text{and} \quad \mathcal{TR}^{w,X \mapsto (B,\mathcal{R})}_A = \mathcal{TR}^w_A. \qquad \square$$

The crucial lemma is the following, which connects the logical relation at a substituted type, $A[B/X]$, with the relation at the type A in an extended type environment, mediated by the lifted embeddings and projections. The statement involves instantiating a type variable with a particular, well chosen, relation.

Lemma 6 (*Type substitution*)
Let $\Delta \vdash B$, $\Delta \vdash w$, and $w(X_j) = (B_j, \mathcal{R}_j)$ for each j. Define the extended type environment $w' = w, X \mapsto (B[B_j/X_j], \widehat{\mathcal{R}}^w_B)$. Then for any A and M, with $\Delta, X \vdash A$, $\cdot;\cdot \vdash M : (A[B/X])[B_j/X_j]$, the following hold:

1. For any d, $(M,d) \in \mathcal{R}^w_{A[B/X]}$ implies $(M, I^B_{X.A}(d)) \in \mathcal{R}^{w'}_A$.
2. For any d, $(M,d) \in \mathcal{R}^{w'}_A$ implies $(M, J^B_{X.A}(d)) \in \mathcal{TR}^w_{A[B/X]}$.

A natural first attempt at a logical relations proof would replace "implies" by "iff" in the above, strengthening the lemma significantly. Our proof of Lemma 6 almost works for this

stronger version, except for the second case of function types. That is, it is unclear how to show the following statement:

$$(M, J^B_{X.A_1\to A_2}(d)) \in \mathcal{TR}^w_{(A_1\to A_2)[B/X]} \text{ implies } (M,d) \in \mathcal{R}^{w'}_{A_1\to A_2}.$$

Ignoring monads for the moment, the problem is that at some point we want $I(J(d)) = d$, which is false in general. Lemma 6 is carefully formulated so that we no longer need this false statement, and yet is still strong enough to derive the correctness theorem for the translation. Here is a failed proof attempt of the strengthened version of Lemma 6.

Proof Attempt
From the assumption $(M, J^B_{X.A_1\to A_2}(d)) \in \mathcal{TR}^w_{(A_1\to A_2)[B/X]}$ we know

$$M \equiv_\beta \lambda x{:}A_1[B/X][B_j/X_j].M'$$

for some M'. By the definition of $(M,d) \in \mathcal{R}^{w'}_{A_1\to A_2}$, it is sufficient to show that

$$(M'[M_2/x],\, d\, d_2) \in \mathcal{TR}^{w'}_{A_2}$$

for any $(M_2, d_2) \in \mathcal{R}^{w'}_{A_1}$. By the definition of $\mathcal{TR}^{w'}_{A_2}$ there will be r such that $d\, d_2 = [r]$ and $(M'[M_2/x], r) \in \mathcal{R}^{w'}_{A_2}$. By inductive hypothesis applied to $(M'[M_2/x], r) \in \mathcal{R}^{w'}_{A_2}$ and the equations in **D**, it is equivalent to show that

$$(M'[M_2/x], (r \leftarrow d\, d_2; J^B_{A_2}(r))) \in \mathcal{TR}^w_{A_2[B/X]}.$$

Again by the assumption $(M, J^B_{X.A_1\to A_2}(d)) \in \mathcal{TR}^w_{(A_1\to A_2)[B/X]}$, and the definition of $J^B_{X.A_1\to A_2}$, we have

$$(M,\, [\lambda a{:}(A_1[B/X])^\dagger.r \leftarrow d(I^B_{X.A_1}(a)); J^B_{X.A_2}(r)]) \in \mathcal{TR}^w_{(A_1\to A_2)[B/X]},$$

which by the definition of $\mathcal{TR}^w_{(A_1\to A_2)[B/X]}$ and the injectivity of $[\cdot]$ is

$$(M,\, (\lambda a{:}(A_1[B/X])^\dagger.r \leftarrow d(I^B_{X.A_1}(a)); J^B_{X.A_2}(r))) \in \mathcal{R}^w_{(A_1\to A_2)[B/X]}.$$

Moreover, by inductive hypothesis applied to $(M_2, d_2) \in \mathcal{R}^{w'}_{A_1}$, we know $(M_2, J^B_{X.A_1}(d_2)) \in \mathcal{TR}^w_{A_1[B/X]}$, which means there is d_2' such that $J^B_{X.A_1}(d_2) = [d_2']$ and $(M_2, d_2') \in \mathcal{R}^w_{A_1[B/X]}$. Thus by the definition of $\mathcal{R}^w_{(A_1\to A_2)[B/X]}$ and the equations in **D**,

$$(M'[M_2/x], (d_2' \leftarrow J^B_{X.A_1}(d_2); r \leftarrow d(I^B_{X.A_1}(d_2')); J^B_{X.A_2}(r))) \in \mathcal{TR}^w_{A_2[B/X]}.$$

Comparing this to the goal, we wish to show the following equation:

$$r \leftarrow d\, d_2; J^B_{A_2}(r) = d_2' \leftarrow J^B_{X.A_1}(d_2); r \leftarrow d(I^B_{X.A_1}(d_2')); J^B_{X.A_2}(r)$$

which would be true if $I^B_{X.A_1}$ were a post-inverse (left-inverse) of $J^B_{X.A_1}$, or that $I^B_{X.A_1}(d_2') = d_2$, which does not hold.

We now present the proof of the correct version of Lemma 6, which evades the difficulty and yet is sufficient for our main result.

Proof
The two parts are proved by simultaneous induction on A. Note that the type environment w remains free (universally quantified) in the induction hypothesis because in the case $A = \forall Y.A'$ the environment will be extended.

- Case X:
  1. By assumption $(M,d) \in \mathcal{R}^w_B$, and by the definition of $\mathcal{TR}^w_B$, $(M,[d]) \in \mathcal{TR}^w_B$. Then by Lemma 1, $j_B(i_B(d)) = [d]$, and thus
  $$(M, j_B(i_B(d))) \in \mathcal{TR}^w_B.$$
  Therefore by the definition of $\widehat{\mathcal{R}}^w_B$, $(M,i_B(d)) \in \widehat{\mathcal{R}}^w_B$. Then by the construction of w', $\widehat{\mathcal{R}}^w_B = \mathcal{R}^{w'}_X$, and also $I^B_{X.X} = i_B$, and thus
  $$(M,I^B_{X.X}(d)) = (M,i_B(d)) \in \widehat{\mathcal{R}}^w_B = \mathcal{R}^{w'}_B.$$
  2. By the construction of w', $\mathcal{R}^{w'}_X = \widehat{\mathcal{R}}^w_B$, and thus
  $$(M,d) \in \widehat{\mathcal{R}}^w_B.$$
  By the definition of $\widehat{\mathcal{R}}^w_B$, and also the fact that $J^B_{X.X}(d) = j_B(d)$,
  $$(M,J^B_{X.X}(d)) = (M,j_B(d)) \in \mathcal{TR}^w_B = \mathcal{TR}^w_{X[B/X]}.$$
- Case Y (a variable different from X):
  1. By Lemma 5 (weakening) $\mathcal{R}^{w'}_Y = \mathcal{R}^w_Y$. Also $I^B_{X.Y}(d) = d$. Therefore
  $$(M,I^B_{X.Y}(d)) = (M,d) \in \mathcal{R}^w_Y = \mathcal{R}^{w'}_Y.$$
  2. By Lemma 5 (weakening) $\mathcal{R}^{w'}_Y = \mathcal{R}^w_Y$. Also $J^B_{X.Y}(d) = [d]$. Therefore by the definition of $\mathcal{TR}^w_Y$,
  $$(M,J^B_{X.Y}(d)) = (M,[d]) \in \mathcal{TR}^w_Y = \mathcal{TR}^w_{Y[B/X]}.$$
- Case $\mathtt{nat}$:
  1. By Lemma 5 (weakening) $\mathcal{R}^{w'}_{\mathtt{nat}} = \mathcal{R}^w_{\mathtt{nat}}$. Also $I^B_{X.\mathtt{nat}}(d) = d$. Therefore
  $$(M,I^B_{X.\mathtt{nat}}(d)) = (M,d) \in \mathcal{R}^w_{\mathtt{nat}} = \mathcal{R}^{w'}_{\mathtt{nat}}.$$
  2. By Lemma 5 (weakening) $\mathcal{R}^{w'}_{\mathtt{nat}} = \mathcal{R}^w_{\mathtt{nat}}$. Also $J^B_{X.\mathtt{nat}}(d) = [d]$. Therefore by the definition of $\mathcal{TR}^w_{\mathtt{nat}}$,
  $$(M,J^B_{X.\mathtt{nat}}(d)) = (M,[d]) \in \mathcal{TR}^w_{\mathtt{nat}} = \mathcal{TR}^w_{\mathtt{nat}[B/X]}.$$
- Case $A_1 \to A_2$:
  In either part $M \equiv_\beta \lambda x{:}A_1[B/X][B_j/X_j].M'$ for some M'.
  1. It is sufficient to show that, for any $(M_2,d_2) \in \mathcal{R}^{w'}_{A_1}$,
  $$(MM_2, I^B_{X.A_1\to A_2}(d)(d_2)) \in \mathcal{TR}^{w'}_{A_2}.$$
  Expanding the definition of $I^B_{X.A_1\to A_2}$ together with the conversion $MM_2 \equiv_\beta M'[M_2/x]$, this is equivalent to
  $$(M'[M_2/x], (\lambda a{:}A_1^\dagger.a' \leftarrow J^B_{X.A_1}(a); r \leftarrow d\,a'; [I^B_{X.A_2}(r)])(d_2)) \in \mathcal{TR}^{w'}_{A_2}.$$
  By the equations in **D** this is the same as
  $$(M'[M_2/x], (a' \leftarrow J^B_{X.A_1}(d_2); r \leftarrow d\,a'; [I^B_{X.A_2}(r)])) \in \mathcal{TR}^{w'}_{A_2}.$$

By inductive hypothesis applied to $(M_2, d_2) \in \mathcal{R}^{w'}_{A_1}$, $(M_2, J^B_{X.A_1}(d_2)) \in \mathcal{TR}^{w}_{A_1[B/X]}$, which means there is d_2' such that $J^B_{X.A_1}(d_2) = [d_2']$ and $(M_2, d_2') \in \mathcal{R}^{w}_{A_1[B/X]}$. Then we can simplify the **D** expression further:

$$\begin{aligned} & a' \leftarrow J^B_{X.A_1}(d_2); r \leftarrow d\,a'; [I^B_{X.A_2}(r)] \\ = {} & a' \leftarrow [d_2']; r \leftarrow d\,a'; [I^B_{X.A_2}(r)] \\ = {} & r \leftarrow d\,d_2'; [I^B_{X.A_2}(r)] \end{aligned}$$

Because $(M, d) \in \mathcal{R}^{w}_{(A_2 \to A_1)[B/X]}$ and $(M_2, d_2') \in \mathcal{R}^{w}_{A_1[B/X]}$, by definition

$$(M'[M_2/x], d\,d_2') \in \mathcal{TR}^{w}_{A_2[B/X]},$$

which is to say there exists r' such that $d\,d_2' = [r']$ and $(M'[M_2/x], r') \in \mathcal{R}^{w}_{A_2[B/X]}$. We can then simplify the expression even more:

$$\begin{aligned} & r \leftarrow d\,d_2'; [I^B_{X.A_2}(r)] \\ = {} & r \leftarrow [r']; [I^B_{X.A_2}(r)] \\ = {} & [I^B_{X.A_2}(r')]. \end{aligned}$$

The goal becomes

$$(M'[M_2/x], I^B_{X.A_2}(r')) \in \mathcal{R}^{w'}_{A_2},$$

which is exactly the inductive hypothesis applied to $(M'[M_2/x], r') \in \mathcal{R}^{w}_{A_2[B/X]}$.

2. By the definition of $\mathcal{TR}^{w}_{(A_1 \to A_2)[B/X]}$, it is equivalent to show that there is d' such that $[d'] = J^B_{X.A_1 \to A_2}(d)$ and $(M, d') \in \mathcal{R}^{w}_{(A_1 \to A_2)[B/X]}$, which is to say for any $(M_2, d_2) \in \mathcal{R}^{w}_{A_1[B/X]}$,

$$(M\,M_2, d'\,d_2) \in \mathcal{TR}^{w}_{A_2[B/X]}.$$

Expanding the definition of $J^B_{X.A_1 \to A_2}$, the equation $[d'] = J^B_{X.A_1 \to A_2}(d)$ means

$$[d'] = [\lambda a{:}(A_1[B/X])^\dagger.r \leftarrow d(I^B_{X.A_1}(a)); J^B_{X.A_2}(r)]$$

and we will show this obvious choice of d' works:

$$d' = \lambda a{:}(A_1[B/X])^\dagger.r \leftarrow d(I^B_{X.A_1}(a)); J^B_{X.A_2}(r).$$

With the conversion $M\,M_2 \equiv_\beta M'[M_2/x]$ the goal becomes

$$(M'[M_2/x], (\lambda a{:}(A_1[B/X])^\dagger.r \leftarrow d(I^B_{X.A_1}(a)); J^B_{X.A_2}(r))(d_2)) \in \mathcal{TR}^{w}_{A_2[B/X]}.$$

Then by the equations in **D** this is the same as

$$(M'[M_2/x], (r \leftarrow d(I^B_{X.A_1}(d_2)); J^B_{X.A_2}(r))) \in \mathcal{TR}^{w}_{A_2[B/X]}.$$

By inductive hypothesis applied to $(M_2, d_2) \in \mathcal{R}^{w}_{A_1[B/X]}$,

$$(M_2, I^B_{X.A_1}(d_2)) \in \mathcal{R}^{w'}_{A_1}.$$

Then because $(M, d) \in \mathcal{R}^{w'}_{A_1 \to A_2}$,

$$(M'[M_2/x], d(I^B_{X.A_1}(d_2))) \in \mathcal{TR}^{w'}_{A_2},$$

which means there exists r' such that $d(I^B_{X.A_1}(d_2)) = [r']$ and $(M'[M_2/x], r') \in \mathcal{R}^{w'}_{A_2}$. Therefore

$$\begin{aligned} & r \leftarrow d(I^B_{X.A_1}(d_2)); J^B_{X.A_2}(r) \\ = \; & r \leftarrow [r']; J^B_{X.A_2}(r) \\ = \; & J^B_{X.A_2}(r'), \end{aligned}$$

and thus it suffices to show

$$(M'[M_2/x], J^B_{X.A_2}(r')) \in \mathcal{TR}^w_{A_2[B/X]},$$

which is the inductive hypothesis applied to $(M'[M_2/x], r') \in \mathcal{R}^{w'}_{A_2}$.

- Case $\forall Y.A'$:
  In either part $M \equiv_\beta \Lambda Y.M'$ for some M'.

  1. Expanding the definition, we know it is sufficient to show

  $$(M'[C/Y], I^B_{X.A'}(d)) \in \mathcal{R}^{w', Y \mapsto (C, \mathcal{R}_C)}_{A'}$$

  for any $(C, \mathcal{R}_C) \in \mathcal{F}$. Fix a pair $(C, \mathcal{R}_C)$. By the definition of $\mathcal{R}^w_{A'[B/X]}$,

  $$(M'[C/Y], d) \in \mathcal{R}^{w, Y \mapsto (C, \mathcal{R}_C)}_{A'[B/X]}$$

  and by induction

  $$(M'[C/Y], I^B_{X.A'}(d)) \in \mathcal{R}^{w, Y \mapsto (C, \mathcal{R}_C), X \mapsto \left(B[B_j/X_j], \widehat{\mathcal{R}}^{w, Y \mapsto (C, \mathcal{R}_C)}_B\right)}_{A'}.$$

  By Lemma 5 (weakening),

  $$\widehat{\mathcal{R}}^{w, Y \mapsto (C, \mathcal{R}_C)}_B = \widehat{\mathcal{R}}^w_B$$

  and by exchange (implicit in the treatment of type environments as maps), we have the goal

  $$(M'[C/Y], I^B_{X.A'}(d)) \in \mathcal{R}^{w', Y \mapsto (C, \mathcal{R}_C)}_{A'}.$$

  2. By the definition of $\mathcal{TR}^w_{\forall Y.A'[B/X]}$, it is equivalent to show that there is d' such that $[d'] = J^B_{X.\forall Y.A'}(d)$ and $(M, d') \in \mathcal{R}^w_{\forall Y.A'[B/X]}$. That is, there is d' such that $[d'] = J^B_{X.\forall Y.A'}(d)$ and for every $(C, \mathcal{R}_C) \in \mathcal{F}$,

  $$(M'[C/Y], d') \in \mathcal{R}^{w, Y \mapsto (C, \mathcal{R}_C)}_{A'[B/X]}.$$

  We claim that we can swap the universal quantifier of $(C, \mathcal{R}_C)$ and the existential quantifier of d'. The reason is that $[\cdot]$ is injective and so d' is uniquely determined by $J^B_{X.\forall Y.A'}(d)$. After the swapping, the goal is then for every $(C, \mathcal{R}_C) \in \mathcal{F}$, there is d' such that $[d'] = J^B_{X.\forall Y.A'}(d)$ and

  $$(M'[C/Y], d') \in \mathcal{R}^{w, Y \mapsto (C, \mathcal{R}_C)}_{A'[B/X]},$$

  which is exactly the definition of

  $$(M'[C/Y], J^B_{X.\forall Y.A'}(d)) \in \mathcal{TR}^{w, Y \mapsto (C, \mathcal{R}_C)}_{A'[B/X]}.$$

Note that $J^B_{X.\forall Y.A'} = J^B_{X.A'}$ and thus this is also equivalent to

$$(M'[C/Y], J^B_{X.A'}(d)) \in \mathcal{TR}^{w,Y\mapsto(C,\mathcal{R}_C)}_{A'[B/X]}.$$

Fix the $(C,\mathcal{R}_C) \in \mathcal{F}$. From the assumption $(M,d) \in \mathcal{R}^{w'}_{\forall Y.A'}$ and the definition of the extended type environment w' we have

$$(M'[C/Y], d) \in \mathcal{R}^{w',Y\mapsto(C,\mathcal{R}_C)}_{A'} = \mathcal{R}^{w,X\mapsto\left(B[B_j/X_j],\widehat{\mathcal{R}}^w_B\right),Y\mapsto(C,\mathcal{R}_C)}_{A'}.$$

By Lemma 5 (weakening),

$$\widehat{\mathcal{R}}^{w,Y\mapsto(C,\mathcal{R}_C)}_B = \widehat{\mathcal{R}}^w_B$$

and therefore, together with exchange,

$$(M'[C/Y], d) \in \mathcal{R}^{w,Y\mapsto(C,\mathcal{R}_C),X\mapsto\left(B[B_j/X_j],\widehat{\mathcal{R}}^{w,Y\mapsto(C,\mathcal{R}_C)}_B\right)}_{A'}.$$

Applying the inductive hypothesis, we have the desired statement

$$(M'[C/Y], J^B_{X.A'}(d)) \in \mathcal{TR}^{w,Y\mapsto(C,\mathcal{R}_C)}_{A'[B/X]}.$$

□

Armed with Lemma 6, we are now in a position to show the 'Fundamental Property': that each (open) source term is logically related to its translation. The relation is defined on closed terms, so the statement of the lemma involves substituting arbitrary types and relations for free type variables, and arbitrary—but related—closed source and target terms for free term variables.

Lemma 7 (*Fundamental Property*)
Suppose $\Delta;\Gamma \vdash M : A$, where $\Delta = \cdot, X_1, \ldots, X_m$ and $\Gamma = \cdot, x_1{:}A_1, \ldots, x_n{:}A_n$. Let w be such that $\Delta \vdash w$ and $w(X_j) = (B_j, \mathcal{R}_j)$ for each $1 \le j \le m$. Then for any list of source terms $V_i : A_i[B_j/X_j]$ and target terms $t_i : A_i^\dagger$, $1 \le i \le n$, such that $(V_i, t_i) \in \mathcal{R}^w_{A_i}$ for each i, we have

$$(M[B_j/X_j][V_i/x_i], (\Delta;\Gamma \vdash M : A)^*[t_i/x_i]) \in \mathcal{TR}^w_A.$$

Proof
Induction on the derivation of $\Delta;\Gamma \vdash M : A$. We first define some abbreviations, writing $\widehat{w}$ for the type substitution $[B_j/X_j]$, $\widehat{V}$ for the source term substitution $[V_i/x_i]$, and $\hat{t}$ for the target term substitution $[t_i/x_i]$.

- Case x: Directly from the assumption.
- Case $\mathtt{z}$: Directly from the definition.
- Case $\mathtt{suc}(M)$:
  By inductive hypothesis

  $$(\widehat{V}(\widehat{w}(M)), \hat{t}((\Delta;\Gamma \vdash M : \mathtt{nat})^*)) \in \mathcal{TR}^w_{\mathtt{nat}}.$$

  By definition of the relation, there exists n such that $\hat{t}((\Delta;\Gamma \vdash M : \mathtt{nat})^*) = [n]$ and

  $$(\widehat{V}(\widehat{w}(M)), n) \in \mathcal{R}^w_{\mathtt{nat}}.$$

So $\widehat{V}(\widehat{w}(M)) \equiv_\beta \overline{n}$ and thus $\widehat{V}(\widehat{w}(\mathtt{suc}(M))) \equiv_\beta \mathtt{suc}(\overline{n})$. Hence

$$\begin{array}{lll}
& (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), n+1) \in \mathcal{R}^w_{\mathtt{nat}} & \text{by definition} \\
\Longrightarrow & (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), [n+1]) \in \mathcal{TR}^w_{\mathtt{nat}} & \text{definition} \\
\Longrightarrow & (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), d \leftarrow [n]; [d+1]) \in \mathcal{TR}^w_{\mathtt{nat}} & \text{monad} \\
\Longrightarrow & (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), d \leftarrow \hat{t}((\Delta;\Gamma \vdash M : \mathtt{nat})^*); [d+1]) \in \mathcal{TR}^w_{\mathtt{nat}} & \\
\Longrightarrow & (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), \hat{t}(d \leftarrow (\Delta;\Gamma \vdash M : \mathtt{nat})^*; [d+1])) \in \mathcal{TR}^w_{\mathtt{nat}} & \\
\Longrightarrow & (\widehat{V}(\widehat{w}(\mathtt{suc}(M))), \hat{t}((\Delta;\Gamma \vdash \mathtt{suc}(M) : \mathtt{nat})^*)) \in \mathcal{TR}^w_{\mathtt{nat}} & \text{translation}
\end{array}$$

- Case $\mathtt{ifz}(M;N_0;x.N_1)$:
  By induction,
  $$(\widehat{V}(\widehat{w}(M)), \hat{t}((\Delta;\Gamma \vdash M : \mathtt{nat})^*)) \in \mathcal{TR}^w_{\mathtt{nat}}.$$
  Hence there exists n such that $\hat{t}((\Delta;\Gamma \vdash M : \mathtt{nat})^*) = [n]$ and $(\widehat{V}(\widehat{w}(M)), n) \in \mathcal{R}^w_{\mathtt{nat}}$, so $\widehat{V}(\widehat{w}(M)) \equiv_\beta \overline{n}$. Then
  $$\begin{array}{ll}
  & \hat{t}((\Delta;\Gamma \vdash \mathtt{ifz}(M;N_0;x.N_1) : A)^*) \\
  = & \hat{t}(d \leftarrow [n]; \mathtt{ifz}(d; (\Delta;\Gamma \vdash N_0 : A)^*; x.(\Delta;\Gamma, x{:}\mathtt{nat} \vdash N_1 : A)^*)) \\
  = & \mathtt{ifz}(n; \hat{t}((\Delta;\Gamma \vdash N_0 : A)^*); x.\hat{t}((\Delta;\Gamma, x{:}\mathtt{nat} \vdash N_1 : A)^*)) \\
  = & \begin{cases} \hat{t}((\Delta;\Gamma \vdash N_0 : A)^*) & \text{if } n = 0 \\ \hat{t}((\Delta;\Gamma, x{:}\mathtt{nat} \vdash N_1 : A)^*)[n'/x] & \text{if } n = n'+1 \end{cases}
  \end{array}$$
  — In the case that $n = 0$, $\widehat{V}(\widehat{w}(\mathtt{ifz}(M;N_0;x.N_1))) \equiv_\beta \widehat{V}(\widehat{w}(N_0))$ and since, by induction,
  $$(\widehat{V}(\widehat{w}(N_0)), \hat{t}((\Delta;\Gamma \vdash N_0 : A)^*)) \in \mathcal{TR}^w_A,$$
  we are done by Lemma 4 (admissibility).
  — If $n = n'+1$ then $\widehat{V}(\widehat{w}(\mathtt{ifz}(M;N_0;x.N_1))) \equiv_\beta \widehat{V}(\widehat{w}(N_1))[\overline{n'}/x]$. Since $(\overline{n'}, n') \in \mathcal{R}^w_{\mathtt{nat}}$, induction gives
  $$(\widehat{V}(\widehat{w}(N_1))[\overline{n'}/x], \hat{t}((\Delta;\Gamma, x{:}\mathtt{nat} \vdash N_1 : A)^*)[n'/x]) \in \mathcal{TR}^w_A$$
  we are again done by Lemma 4.
- Case $\lambda x{:}A.M$:
  Since $\hat{t}((\Delta;\Gamma \vdash \lambda x{:}A.M : A \to B)^*) = [\lambda x{:}A^\dagger.\hat{t}((\Delta;\Gamma, x{:}A \vdash M : B)^*)]$ it suffices to show
  $$(\widehat{V}(\widehat{w}(\lambda x{:}A.M)), \lambda x{:}A^\dagger.\hat{t}((\Delta;\Gamma, x{:}A \vdash M : B)^*)) \in \mathcal{R}^w_{A\to B}$$
  Suppose $(M_2, d_2) \in \mathcal{R}^w_A$, then we need to show
  $$(\widehat{V}(\widehat{w}(M))[M_2/x], \hat{t}((\Delta;\Gamma, x{:}A \vdash M : B)^*)[d_2/x]) \in \mathcal{TR}^w_B$$
  which follows by induction.
- Case MN:
  By the inductive hypotheses and the monadic relation, there are target values d and e such that
  $$\hat{t}((\Delta;\Gamma \vdash M : A \to B)^*) = [d] \quad \text{and} \quad (\widehat{V}(\widehat{w}(M)), d) \in \mathcal{R}^w_{A\to B}$$

and

$$\hat{t}((\Delta;\Gamma \vdash N : A)^*) = [e] \quad \text{and} \quad (\widehat{V}(\widehat{w}(N)), e) \in \mathcal{R}^w_A.$$

Thus we know $\widehat{V}(\widehat{w}(M)) \equiv_\beta \lambda x{:}A.M'$ for some M' such that

$$(M'[\widehat{V}(\widehat{w}(N))/x], d\,e) \in \mathcal{TR}^w_B.$$

Since $\widehat{V}(\widehat{w}(MN)) \equiv_\beta M'[\widehat{V}(\widehat{w}(N))/x]$, Lemma 4 gives

$$(\widehat{V}(\widehat{w}(MN)), d\,e) \in \mathcal{TR}^w_B.$$

And since

$$\begin{aligned}\hat{t}((\Delta;\Gamma \vdash MN : B)^*) &= d' \leftarrow \hat{t}((\Delta;\Gamma \vdash M : A \to B)^*); e' \leftarrow \hat{t}((\Delta;\Gamma \vdash N : A)^*); d'\,e' \\ &= d' \leftarrow [d]; e' \leftarrow [e]; d'\,e' \\ &= d\,e\end{aligned}$$

we are done.

- Case $\Lambda X.M$:
  We want to show

  $$(\widehat{V}(\widehat{w}(\Lambda X.M)), \hat{t}((\Delta;\Gamma \vdash \Lambda X.M : \forall X.A)^*)) \in \mathcal{TR}^w_{\forall X.A},$$

  which is to say there is d such that

  $$\hat{t}((\Delta;\Gamma \vdash \Lambda X.M : \forall X.A)^*) = [d] \quad \text{and} \quad (\widehat{V}(\widehat{w}(\Lambda X.M)), d) \in \mathcal{R}^w_{\forall X.A}.$$

  Expanding the definition and using the conversion relation $\equiv_\beta$ (since the relation is admissible), this means there is d such that

  $$\hat{t}((\Delta, X;\Gamma \vdash M : A)^*) = [d]$$

  and for any $(B, \mathcal{R}) \in \mathcal{F}$,

  $$(\widehat{V}(\widehat{w}(M)[B/X]), d) \in \mathcal{R}^{w, X \mapsto (B,\mathcal{R})}_A.$$

  Pick the $(B, \mathcal{R}) \in \mathcal{F}$. By induction hypothesis applied to the extended type environment $w' = w, X \mapsto (B, \mathcal{R})$, we have

  $$(\widehat{V}(\widehat{w}(M)[B/X]), (\Delta, X;\Gamma \vdash M : A)^*) \in \mathcal{TR}^{w, X \mapsto (B,\mathcal{R})}_A,$$

  which means there is d' such that

  $$\hat{t}((\Delta, X;\Gamma \vdash M : A)^*) = [d'] \quad \text{and} \quad (\widehat{V}(\widehat{w}(M)[B/X]), d') \in \mathcal{R}^{w, X \mapsto (B,\mathcal{R})}_A.$$

  The choice $d = d'$ meets our goal.
- Case MB:
  By induction, we know

  $$(\widehat{V}(\widehat{w}(M)), \hat{t}((\Delta;\Gamma \vdash M : \forall X.A)^*)) \in \mathcal{TR}^w_{\forall X.A}.$$

  So there is a d such that

  $$\hat{t}((\Delta;\Gamma \vdash M : \forall X.A)^*) = [d] \quad \text{and} \quad (\widehat{V}(\widehat{w}(M)), d) \in \mathcal{R}^w_{\forall X.A}.$$

Unfolding the logical relation for quantified types and instantiating with $(\widehat{w}(B), \widehat{\mathcal{R}}^w_B) \in \mathcal{F}$ yields that $\widehat{V}(\widehat{w}(M)) \equiv_\beta \Lambda X.M'$ for some M' with

$$(M'[\widehat{w}(B)/X], d) \in \mathcal{R}_A^{w, X \mapsto (\widehat{w}(B), \widehat{\mathcal{R}}^w_B)}.$$

By the second part of Lemma 6, the key type substitution property, this implies

$$(M'[\widehat{w}(B)/X], J^B_{X.A}(d)) \in \mathcal{TR}^w_{A[B/X]}.$$

Since $\widehat{V}(\widehat{w}(MB)) = \widehat{V}(\widehat{w}(M))\,\widehat{w}(B) \equiv_\beta (\Lambda X.M')\,\widehat{w}(B) \equiv_\beta M'[(\widehat{w}(B)/X]$, Lemma 4 gives

$$(\widehat{V}(\widehat{w}(MB)), J^B_{X.A}(d)) \in \mathcal{TR}^w_{A[B/X]}.$$

By definition of the translation,

$$\begin{aligned} \hat{t}((\Delta;\Gamma \vdash MB : A[B/X])^*) &= d' \leftarrow \hat{t}((\Delta;\Gamma \vdash M : \forall X.A)^*); J^B_{X.A}(d') \\ &= d' \leftarrow [d]; J^B_{X.A}(d') \\ &= J^B_{X.A}(d) \end{aligned}$$

So

$$(\widehat{V}(\widehat{w}(MB)), \hat{t}((\Delta;\Gamma \vdash MB : A[B/X])^*)) \in \mathcal{TR}^w_{A[B/X]}$$

as required.

□

An immediate consequence of Lemma 7 is that the behaviour of a program (closed term of ground type) and its translation agree:

Corollary 1
If $\cdot;\cdot \vdash M : \mathtt{nat}$ then there exists an n such that $M \equiv_\beta \overline{n}$ and $(\cdot;\cdot M : \mathtt{nat})^* = [n]$. □

5 Discussion

Using logical relations it is possible to prove the correctness of the compilation of polymorphic types to dynamic types in such a way that overhead is imposed only insofar as polymorphism is actually used. This compilation method lies at the heart of the implementation of generic extensions to Java, and of polymorphic languages such as Scala, on the Java Virtual Machine, with the type `Object` playing the role of our $\mathbb{D}$. As far as we are aware this is the first correctness proof of this compilation strategy for System F, and is novel insofar as it only relies on an embedding into $\mathbb{D}$, rather than a stronger condition such as isomorphism. In this respect the proof may be useful in other situations where the correctness of a compilation method is required.

Semantically, the underlying idea of interpreting types as retracts of a universal domain is an old one, going back to work of Scott (1976) and McCracken (1979). It has been adapted and used for various purposes in programming, including by Benton (2005) and Ramsey (2011) for interfacing typed languages with untyped ones, and by many authors studying run-time enforcement of contracts (Findler & Felleisen, 2002) in dynamic languages, and the correct assignment of blame should violations occur (Ahmed *et al.*, 2011).

The broad shape of the proof presented here is that of *adequacy*: showing agreement between an operational and a denotational (translational) semantics via a logical relation (Plotkin, 1977; Amadio, 1993). Similar logical relations have also been used for the closely-related task of establishing the correctness of compilers (Minamide *et al.*, 1996; Benton & Hur, 2010; Hur & Dreyer, 2011).

One possible extension to this work is to consider the extension of System F with general recursion at the expression level, or, more generally, with recursive types. It appears that handling general recursion is straightforward, following directly the strategy outlined in Chapter 48 of the third author's text (Harper, 2012), which requires that admissible relations be closed under limits of suitable chains, and which employs fixed point induction in establishing the main theorem. The extension to product and sum types is entirely straightforward. Recursive types require more sophisticated techniques pioneered by Pitts (1996), and adapted to the operational setting by Crary & Harper (2007). Step-indexed methods, such as those introduced by Appel & McAllester (2001); Ahmed (2006) may also be useful in this respect.

Another possible extension is to consider System F with higher-order polymorphism, namely System F_ω, which enables programmers to abstract over even type constructors, such as lists or trees which themselves are polymorphic in their element type. Such higher-order polymorphism has been materialized in dynamic typing, for example in Scala, by Moors *et al.* (2008), and it is conceivable, for studying the correctness, to migrate the method to System F_ω as we did to System F. Together with the work by Rossberg *et al.* (2010) which compiles ML modules to System F_ω, an alternative account for the dynamics of ML modules, in terms of dynamic typing, can possibly be made.

Acknowledgements

This paper originated from a homework exercise in an undergraduate programming language theory class that was developed by the third author with Roger Wolff, to whom we are grateful for many helpful discussions and suggestions. Thanks also to Yitzhak Mandelbaum for his numerous suggestions. Thanks to Carlo Angiuli, Shayak Sen and the PoP group at Carnegie Mellon who gave valuable comments on earlier drafts of this work. We also appreciate the feedback from anonymous reviewers, who carefully read our drafts and gave various insightful suggestions.

References

Ahmed, Amal. (2006). Step-indexed syntactic logical relations for recursive and quantified types. *Pages 69–83 of: Programming languages and systems*. Lecture Notes in Computer Science, vol. 3924. Springer Berlin Heidelberg.

Ahmed, Amal, Findler, Robert Bruce, Siek, Jeremy G., & Wadler, Philip. (2011). Blame for all. *Pages 201–214 of: Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)*. ACM.

Amadio, Roberto. (1993). On the adequacy of PER models. *18th international symposium on mathematical foundations of computer science (MFCS)*. LNCS, vol. 711. Springer.

Appel, Andrew W., & McAllester, David. (2001). An indexed model of recursive types for foundational proof-carrying code. *ACM trans. program. lang. syst.*, **23**(5), 657–683.

Bank, Joseph A., Myers, Andrew C., & Liskov, Barbara. (1997). Parameterized types for Java. *Pages 132–145 of: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)*. ACM.

Benton, Nick. (2005). Embedded interpreters. *Journal of functional programming*, **15**(4).

Benton, Nick, & Hur, Chung-Kil. (2010). *Realizability and compositional compiler correctness for a polymorphic language*. Tech. rept. MSR-TR-2010-62. Microsoft Research.

Bracha, Gilad, Odersky, Martin, Stoutamire, David, & Wadler, Philip. (1998). Making the future safe for the past: adding genericity to the Java programming language. *Pages 183–200 of: Proceedings of the 1998 ACM SIGPLAN conference on object-oriented programming systems, languages & applications (OOPSLA '98)*, vol. 33. ACM.

Crary, Karl, & Harper, Robert. (2007). Syntactic logical relations for polymorphic and recursive types. *Electronic notes in theoretical computer science*, **172**, 259–299. Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin.

Curry, Haskell B., Hindley, J. Roger, & Seldin, Jonathan P. (1972). *Combinatory logic, volume ii*. Studies in logic and the foundations of mathematics, vol. 65. North-Holland.

Findler, Robert Bruce, & Felleisen, Matthias. (2002). Contracts for higher-order functions. *Proceedings of the international conference on functional programming (ICFP)*.

Girard, Jean-Yves. (1972). *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Ph.D. thesis, Université Paris 7.

Harper, Robert. (2012). *Practical foundations for programming languages*. Cambridge University Press.

Hur, Chung-Kil, & Dreyer, Derek. (2011). A Kripke logical relation between ML and assembly. *Pages 133–146 of: Proceedings of the 38th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)*. ACM.

Igarashi, Atsushi, Pierce, Benjamin C., & Wadler, Philip. (2001). Featherweight Java: A minimal core calculus for Java and GJ. *ACM trans. program. lang. syst.*, **23**(3), 396–450.

McCracken, Nancy. 1979 (June). *An investigation of a programming language with a polymorphic type structure*. Ph.D. thesis, Syracuse University.

Meyer, Albert R., & Wand, Mitchell. (1985). Continuation semantics in typed lambda-calculi. *Pages 219–224 of: Logics of programs*. Lecture Notes in Computer Science, vol. 193. Springer Berlin Heidelberg.

Milner, Robin. (1978). A theory of type polymorphism in programming. *Journal of computer and system sciences*, **17**(3), 348–374.

Minamide, Yasuhiko, Morrisett, Greg, & Harper, Robert. (1996). Typed closure conversion. *Pages 271–283 of: Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL)*. ACM.

Moggi, Eugenio. (1991). Notions of computation and monads. *Information and computation*, **93**(1), 55–92. Selections from 1989 IEEE Symposium on Logic in Computer Science.

Moors, Adriaan, Piessens, Frank, & Odersky, Martin. (2008). Generics of a higher kind. *Pages 423–438 of: Proceedings of the 23rd ACM SIGPLAN conference on object-oriented programming systems languages and applications (OOPSLA)*. ACM.

Pitts, Andrew M. (1996). Relational properties of domains. *Information and computation*, **127**(2), 66–90.

Plotkin, Gordon D. (1977). LCF considered as a programming language. *Theoretical computer science*, **5**, 223–255.

Ramsey, Norman. (2011). Embedding an interpreted language using higher-order functions and types. *Journal of functional programming*, **21**(6), 585–615.

Reynolds, John. (1983). Types, abstraction and parametric polymorphism. *Pages 513–523 of: Information processing 83*. IFIP congress series, vol. 9. North-Holland.

Reynolds, John C. (1974). Towards a theory of type structure. *Pages 408–423 of: Programming symposium, proceedings colloque sur la programmation*. Springer-Verlag.

Rossberg, Andreas, Russo, Claudio V., & Dreyer, Derek. (2010). F-ing modules. *Pages 89–102 of: Proceedings of the 5th ACM SIGPLAN workshop on types in language design and implementation (TLDI)*. ACM.

Scott, Dana. (1976). Data types as lattices. *SIAM journal of computing*, **5**(3), 522–587.