Type systems for programming languages

Didier Rémy

Academic year 2020-2021
Version of October 12, 2021

Contents

Chapter 8

Logical Relations

8.1 Introduction

So far, most proofs involving terms have proceeded by induction on typing derivations, or equivalently, on the structure of *terms*.

Logical relations are relations between well-typed terms defined inductively on the structure of *types*. They allow proofs between terms by induction on the structure of *types*.

Logical relations may be n-ary. However, we mostly use unary and binary logical relations. Unary logical relations are typed-indexed predicates on terms or, equivalently, type-indexed sets of terms. They are typically used to give the semantics of types as sets of terms, and as a particular case, prove type soundness or termination of reduction. Binary logical relations are type-indexed relations, or type-indexed sets of pairs of terms. They are typically used to prove equivalence of programs and non-interference properties.

Logical relations are a common proof method for programming languages.

8.1.1 Parametricity

Parametricity is a consequence of polymorphism: polymorphic functions cannot examine the argument of polymorphic types, and therefore must treat them parametrically. This often implies that polymorphic functions have actually relatively few inhabitants—up to $\beta\eta$ convertibility. Thus, a polymorphic type can reveal a lot of information about the terms that inhabit it.

Finding inhabitants of polymorphic functions

For example, what can do a term of type $\forall \alpha.\, \alpha \rightarrow \mathsf{int}$? The function cannot examine its argument. Therefore, it must always return the same integer, that is, it must be a constant function. For example, it could be $\lambda x.\, n$, $\lambda x.\, (\lambda y.\, y)\; n$, $\lambda x.\, (\lambda y.\, n)\; x$, *etc.* What do they all have in common? They are all $\beta\eta$-equivalent to a term of the form $\lambda x.\, n$

What can do a term of type $\forall\alpha.\,\alpha\rightarrow\alpha$? Well it receives an argument V of type α and must return a value of type α—but cannot examine α. Thus, it must eventually return v, *i.e.* it behaves as $\lambda x.\,x$—again up to $\beta\eta$ equivalence.

A term type of type $\forall\alpha\beta.\,\alpha\rightarrow\beta\rightarrow\alpha$ is not very different, it additionally receives a value of type w of type β, but there is no way v and w can interact. So that function must also return v, *i.e.* it behaves as $\lambda x.\,\lambda y.\,x$.

Now, a term of type $\forall\alpha.\,\alpha\rightarrow\alpha\rightarrow\alpha$ receives two arguments v and w of type α and must return a value of type α. Again, the arguments cannot interact, as we do not have any operation available of type α. Hence it must return either v or w. That is, it behaves either as $\lambda x.\,\lambda y.\,x$ or as $\lambda x.\,\lambda y.\,y$—up to $\beta\eta$ conversion.

Theorems for free

The type of a polymorphic function may also reveal a *"free theorem"* about its behavior!

For example, what properties may we learn from a function whoami of type $\forall\alpha.\,\mathsf{list}\ \alpha\rightarrow\mathsf{list}\ \alpha$?

- the length of the result depends only on the length of the argument;
- all elements of the results are elements of the argument.
- the choice (i,j) of pairs such that i-th element of the result is the j-th element of the argument does not depend on the element itself;
- the function is preserved by a transformation of its argument that preserves the shape of the argument:

$$\forall f,x,\quad \texttt{whoami}\ (map\ f\ x) = map\ f\ (\texttt{whoami}\ x)$$

What property may we learn for the list sorting function? From it type:

$$\mathsf{sort}\ :\ \forall\alpha.\,(\alpha\rightarrow\alpha\rightarrow\mathsf{bool})\rightarrow\mathsf{list}\ \alpha\rightarrow\mathsf{list}\ \alpha$$

we can actually deduce that if f is order-preserving, then sorting commutes with *map*. Formally:

$$(\forall x,y,\ \mathsf{cmp}_2\ (f\ x)\ (f\ y) = \mathsf{cmp}_1\ \ x\ y)\ \Longrightarrow$$
$$\forall\ell,\ \mathsf{sort}\ \ \mathsf{cmp}_2\ (\mathsf{map}\ f\ \ell) = \mathsf{map}\ f\ (\mathsf{sort}\ \mathsf{cmp}_1\ \ell)$$

Such properties can be used to significantly reduce testing: in particular, if sort is correct on lists of integers, then it is correct on any list. Note that there are many other inhabitants of the type of sort, (e.g., a function that sorts in reverse order, or a function that removes or adds duplicates), but they all satisfy this free theorem.

A few readings

Parametricity has been widely studied by Reynolds 1983. It has been popularized by Wadler 1989; 2007, in the ML community, with his *Theorems for free paper* paper which contains the example of the list sorting function.

An account based on an operational semantics is offered by Pitts 2000.

The application to testing has been generalized by Bernardy et al. (2010) who show how testing any given polymorphic function can be restricted to testing it on (possibly infinitely many) particular values at some particular types.

8.2 Normalization of simply-typed λ-calculus

In general, types also ensure termination of programs—as long as no form of recursion in types or terms has been added. Even if one wishes to add recursion explicitly later on, it is an important property of the design that non-termination is originating from the constructs for recursion only and could not occur without them.

The simply-typed λ-calculus is also lifted at the level of types in richer type systems such as System F^ω where the language of types is itself a simply-typed lambda-calculus and the decidability of type-equality depends on the termination of the reduction at the type level.

Proving termination of reduction in fragments of the λ-calculus is often a difficult task because reduction may create new redexes or duplicate existing ones. However, the proof of termination for the simply-typed λ-calculus is simple enough and interesting to be presented here. Notice that our presentation of simply-typed λ-calculus is equipped with a call-by-value semantics, while proofs of termination are usually done with a strong evaluation strategy where reduction can occur in any context.

We follow the proof schema of Pierce (2002), which is a modern presentation in a call-by-value setting of an older proof by Hindley and Seldin (1986). The proof method, which is now a standard one, is due to Tait (1967).

The idea is to first build the set $\mathcal{E}[\![\tau]\!]$ of terminating closed terms of type τ, and then show that any term of type τ is actually in $\mathcal{E}[\![\tau]\!]$, by induction on terms. Unfortunately, stated as such, this hypothesis is too weak. The difficulty in such cases is usually to find a strong enough induction hypothesis. The solution in this case is to require that terms in $\mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]$ not only terminate but also terminate when applied to any term in $\mathcal{E}[\![\tau_1]\!]$.

We take the call-by-value simply-typed λ-calculus with primitive booleans and conditional. Write B the type of booleans and tt and ff for true and false.

Definition 4 *We recursively define $\mathcal{V}[\![\tau]\!]$ and $\mathcal{E}[\![\tau]\!]$, subsets of closed values and closed ex-*

pressions of (ground) type τ by induction on types *as follows:*

$$\begin{aligned}\mathcal{V}[\![\mathsf{B}]\!] &\triangleq \{\mathsf{tt}, \mathsf{ff}\}\\ \mathcal{V}[\![\tau_1 \to \tau_2]\!] &\triangleq \{\lambda x{:}\tau_1.\, M \mid \forall V \in \mathcal{V}[\![\tau_1]\!],\ (\lambda x{:}\tau_1.\, M)\ V \in \mathcal{E}[\![\tau_2]\!]\}\\ \mathcal{E}[\![\tau]\!] &\triangleq \{M \mid \exists V \in \mathcal{V}[\![\tau]\!],\ M \Downarrow V\}\end{aligned}$$

We write $M \Downarrow V$ as a shorthand for $M \longrightarrow^* V$. The goal is to show that any closed expression of type τ is in $\mathcal{E}[\![\tau]\!]$.

Remark Although usual with logical relations, *well-typedness is actually not required here* and omitted: otherwise, we would have to carry unnecessary type-preservation proof obligations.

The set $\mathcal{E}[\![\tau]\!]$ can be seen as a predicate, *i.e.* a unary relation. It is called a (unary) *logical relation* because it is defined inductively on the structure of types.

Immediate properties

- $\mathcal{V}[\![\tau]\!] \subseteq \mathcal{E}[\![\tau]\!]$ by definition.
- $\mathcal{E}[\![\tau]\!]$ is closed by inverse reduction—by definition, *i.e.* If $M \longrightarrow N$ and $N \in \mathcal{E}[\![\tau]\!]$ then $M \in \mathcal{E}[\![\tau]\!]$.
- $\mathcal{E}[\![\tau]\!]$ is closed by reduction. By confluence (since the reduction is deterministic), if $M \Downarrow N$ and $M \longrightarrow V$, then $N \Downarrow V$.
- For any term in $\mathcal{E}[\![\tau]\!]$, the reduction of M terminates—by definition of $\mathcal{E}[\![\tau]\!]$.

Normalization Therefore, it just remains to show that any term of type τ is in $\mathcal{E}[\![\tau]\!]$:

Lemma 37 *If $\varnothing \vdash M : \tau$, then $M \in \mathcal{E}[\![\tau]\!]$.*

The proof is by induction on (the typing derivation of) M. However, the case for abstraction requires some similar statement, but for open terms. We need to strengthen the lemma, *i.e.* also give a semantics to open terms, which can be given by abstracting over the semantics of their free variables, interpreting free term variables of type τ as *closed values* in $\mathcal{V}[\![\tau]\!]$.

We define a semantic judgment for open terms $\Gamma \vDash M : \tau$ so that $\Gamma \vdash M : \tau$ implies $\Gamma \vDash M : \tau$ and $\varnothing \vDash M : \tau$ means $M \in \mathcal{E}[\![\tau]\!]$.

We interpret environments Γ as *closing substitutions* γ, *i.e.* mappings from term variables to *closed values*: We write $\gamma \in \mathcal{G}[\![\Gamma]\!]$ to mean $\mathsf{dom}(\gamma) = \mathsf{dom}(\Gamma)$ and $\gamma(x) \in \mathcal{V}[\![\tau]\!]$ for all $x : \tau \in \Gamma$. Then, we define

$$\Gamma \vDash M : \tau \overset{\mathsf{def}}{\iff} \forall \gamma \in \mathcal{G}[\![\Gamma]\!],\ \ \gamma(M) \in \mathcal{E}[\![\tau]\!]$$

Theorem 18 (fundamental lemma) *If* $\Gamma \vdash M : \tau$ *then* $\Gamma \vDash M : \tau$*.*

Corollary 38 (termination of well-typed terms) *If* $\varnothing \vdash M : \tau$ *then* $M \in \mathcal{E}[\![\tau]\!]$*.*
That is, closed well-typed terms of type τ *evaluates to values of type* τ*.*

<u>Proof</u>: By induction on the typing derivation

Routine cases

Case $\Gamma \vdash \mathsf{tt} : \mathsf{B}$ *or* $\Gamma \vdash \mathsf{ff} : \mathsf{B}$: by definition, $\mathsf{tt}, \mathsf{ff} \in \mathcal{V}[\![\mathsf{B}]\!]$ and $\mathcal{V}[\![\mathsf{B}]\!] \subseteq \mathcal{E}[\![\mathsf{B}]\!]$.

Case $\Gamma \vdash x : \tau$: $\gamma \in \mathcal{G}[\![\Gamma]\!]$, thus $\gamma(x) \in \mathcal{V}[\![\tau]\!] \subseteq \mathcal{E}[\![\tau]\!]$

Case $\Gamma \vdash M_1\ M_2 : \tau$:

By inversion, $\Gamma \vdash M_1 : \tau_2 \to \tau$ and $\Gamma \vdash M_2 : \tau_2$.

Let $\gamma \in \mathcal{G}[\![\Gamma]\!]$. We have $\gamma(M_1\ M_2) = (\gamma M_1)\ (\gamma M_2)$. By IH, we have $\Gamma \vDash M_1 : \tau_2 \to \tau$ and $\Gamma \vDash M_2 : \tau_2$. Thus $\gamma M_1 \in \mathcal{E}[\![\tau_2 \to \tau]\!]$ (**1**) and $\gamma M_2 \in \mathcal{E}[\![\tau_2]\!]$ (**2**). By (2), there exists $V \in \mathcal{V}[\![\tau_2]\!]$ such that $\gamma M_2 \Downarrow V$. Thus $(\gamma M_1)\ (\gamma M_2) \Downarrow (\gamma M_1)\ V \in \mathcal{E}[\![\tau]\!]$ by (1). Then, $(\gamma M_1)\ (\gamma M_2)$, *i.e.* $\gamma(M_1\ M_2)$ is in $\mathcal{E}[\![\tau]\!]$ by closure by inverse reduction.

Case $\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : \tau$: By cases on the evaluation of γM for γ in $\mathcal{G}[\![\Gamma]\!]$.

The interesting case

Case $\Gamma \vdash \lambda x{:}\tau_1.\, M : \tau_1 \to \tau$:

Assume $\gamma \in \mathcal{G}[\![\Gamma]\!]$. We must show that $\gamma(\lambda x : \tau_1.\, M) \in \mathcal{E}[\![\tau_1 \to \tau]\!]$ (1) That is, $\lambda x : \tau_1.\, \gamma M \in \mathcal{V}[\![\tau_1 \to \tau]\!]$ (we may assume $x \notin \mathsf{dom}(\gamma)$ *w.l.o.g.*) Let $V \in \mathcal{V}[\![\tau_1]\!]$, it suffices to show $(\lambda x : \tau_1.\, \gamma M)\ V \in \mathcal{E}[\![\tau]\!]$ (2). We have $(\lambda x{:}\tau_1.\, \gamma M)\ V \Downarrow (\gamma M)[x \mapsto V] = \gamma' M$ where γ' is $\gamma[x \mapsto V] \in \mathcal{G}[\![\Gamma, x : \tau_1]\!]$ (3). Since $\Gamma, x : \tau_1 \vdash M : \tau$, we have $\Gamma, x : \tau_1 \vDash M : \tau$ by IH. Therefore by (3), we have $\gamma' M \in \mathcal{E}[\![\tau]\!]$. Since $\mathcal{E}[\![\tau]\!]$ is closed by inverse reduction, this proves (2) which finishes the proof of (1).

Variations We have shown both *termination* and *type soundness*, simultaneously. If we had a fix point, termination would not hold, but type soundness would still hold. The proof could then be modified by choosing:

$$\mathcal{E}[\![\tau]\!] = \left\{ M : \tau \mid \forall N, M \Downarrow N \implies \left(N \in \mathcal{V}[\![\tau]\!] \ \vee\ \exists N', N \longrightarrow N' \right) \right\}$$

Exercise 48 *Show type soundness with this semantics.* □

8.3 Observational equivalence in simply-typed λ-calculus

The rest of these course notes are largely inspired by course notes *Practical foundations for programming languages* by Harper (2012) and the *Introduction to Logical Relations* by Skorstengaard (2019). You may also read earlier reference papers:

- *Types, Abstraction and Parametric Polymorphism* Reynolds (1983)
- *Parametric Polymorphism and Operational Equivalence* Pitts (2000).
- *Theorems for free!* Wadler (1989).

We assume a call-by-value operational semantics (instead of call-by-name in Harper (2012)).

Program equivalence Observational equivalence is answering the question: when are two programs M and N are equivalent?

We should at least include the case where one program reduces to the other, or even, more generally, when both programs reduce to the same value. But is this sufficient? Unfortunately not: what if M and N are functions—hence values: Aren't $\lambda x.\,(x+x)$ and $\lambda x.\,2 * x$ also equivalent? Yes, they are. Indeed, two functions are observationally equivalent if when applied to *equivalent arguments*, they lead to observationally *equivalent results*. Still, are we general enough? How can we tell?

We can only *observe* the behavior of full *programs*, *i.e.* closed terms of some computational type, such as Booleans B (the only one in our setting). We thus define:

Definition 5 *(Behavioral equivalence) Two closed programs M and N of the same base type are* behaviorally equivalent *and we write $M \simeq N$ iff there exists V such that $M \Downarrow V$ and $N \Downarrow V$.*

To compare programs at other types, we place them in arbitrary *closing* contexts. Since we often manipulate pairs of well-typed programs, we write $\Gamma \vdash M, N : \tau$ as an abbreviation for $\Gamma \vdash M : \tau \wedge \Gamma \vdash N : \tau$.

Definition 6 *(observational equivalence) Assume $\Gamma \vdash M, N : \tau$. We say that M and N are observationally equivalent and we write $\Gamma \vdash M \cong N : \tau$ if there are behaviorally equivalent when placed in any closing context at some base type. That is,*

$$\Gamma \vdash M \cong N : \tau \;\triangleq\; \forall \mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\varnothing \rhd \mathsf{B}),\ \mathcal{C}[M] \simeq \mathcal{C}[N]$$

Definition 7 *(Typing of context)*

$$\mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\Delta \rhd \sigma) \overset{\mathit{def}}{\iff} (\forall M,\ \ \Gamma \vdash M : \tau \implies \Delta \vdash \mathcal{C}[M] : \sigma)$$

There is an equivalent definition given by a set of typing rules. This is needed to prove some properties by induction on the typing derivations.

We write $M \cong_\tau N$ as an abbreviation for $\varnothing \vdash M \cong N : \tau$

Lemma 39 *Observational equivalence is the coarsest consistent congruence, where:*

- *a relation* $\equiv$ *is consistent if* $\varnothing \vdash M \equiv N : \mathsf{B}$ *implies* $M \simeq N$.
- *a relation* $\equiv$ *is a congruence if it is an equivalence and is closed by context,* i.e.

$$\Gamma \vdash M \equiv N : \tau \;\;\wedge\;\; \mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\Delta \rhd \sigma) \quad\Longrightarrow\quad \Delta \vdash \mathcal{C}[M] \equiv \mathcal{C}[N] : \sigma$$

<ins>Proof</ins>:

Consistent: by definition, using the empty context.

Congruence: by compositionality of contexts.

Coarsest: Assume $\equiv$ is a consistent congruence. Assume $\Gamma \vdash M \equiv N : \tau$ holds and show that $\Gamma \vdash M \cong N : \tau$ holds (1).
Let $\mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\varnothing \rhd \mathsf{B})$ (2). We must show that $\mathcal{C}[M] \simeq \mathcal{C}[N]$.
This follows by consistency applied to $\Gamma \vdash \mathcal{C}[M] \equiv \mathcal{C}[N] : \mathsf{B}$ which follows by congruence from (1) and (2).

Problem with Observational Equivalence Observational equivalence is too difficult to test: Because of quantification over all contexts (too many for testing). but many contexts will do the same experiment.

The solution is to take advantage of types to reduce the number of experiments by defining and testing the equivalence on base types and propagating the definition mechanically at other types.

Logical relations provide the infrastructure for conducting such proofs.

8.4 Logical relations in simply-typed λ-calculus

8.4.1 Logical equivalence for closed terms

Unary logical relations interpret types by predicates on (*i.e.* sets of) closed values of that type. Binary relations interpret types by binary relations on closed values of that type, *i.e.* sets of pairs of related values of that type.

That is, $\mathcal{V}[\![\tau]\!]$ is a subset of $\mathsf{Val}(\tau) \times \mathsf{Val}(\tau)$ and $\mathcal{E}[\![\tau]\!]$, the closure of $\mathcal{V}[\![\tau]\!]$ by inverse reduction is a subset of $\mathit{Expressions}\tau \times \mathsf{Exp}(\tau)$.

Definition 8 *(Logical equivalence for closed terms) We recursively define two relations $\mathcal{V}[\![\tau]\!]$ and $\mathcal{E}[\![\tau]\!]$ between values of type τ and expressions of type τ by*

$$
\begin{aligned}
\mathcal{V}[\![\mathsf{B}]\!] &\triangleq \{(\mathsf{tt},\mathsf{tt}),(\mathsf{ff},\mathsf{ff})\}\\
\mathcal{V}[\![\tau \to \sigma]\!] &\triangleq \{(V_1,V_2) \mid V_1,V_2 \vdash \tau \to \sigma \wedge\\
&\qquad \forall (W_1,W_2) \in \mathcal{V}[\![\tau]\!],\ (V_1\, W_1, V_2\, W_2) \in \mathcal{E}[\![\sigma]\!]\}\\
\\
\mathcal{E}[\![\tau]\!] &\triangleq \{(M_1,M_2) \mid M_1,M_2 : \tau \wedge\\
&\qquad \exists (V_1,V_2) \in \mathcal{V}[\![\tau]\!],\ M_1 \Downarrow V_1 \wedge M_2 \Downarrow V_2\}
\end{aligned}
$$

Notice the (highlighted) mutual recursion between $\mathcal{V}[\![\cdot]\!]$ and $\mathcal{E}[\![\cdot]\!]$. In the following we will leave the typing constraint in gray implicit, *i.e.* we will treat them as global conditions for sets $\mathcal{V}[\![\cdot]\!]$ and $\mathcal{E}[\![\cdot]\!]$. We also write $M_1 \sim_\tau M_2$ for $(M_1,M_2) \in \mathcal{E}[\![\tau]\!]$ and $V_1 \approx_\tau V_2$ for $(V_1,V_2) \in \mathcal{V}[\![\tau]\!]$.

Non-termination In a language with non-termination, we change the definition of $\mathcal{E}[\![\tau]\!]$ to

$$
\begin{aligned}
\mathcal{E}[\![\tau]\!] \triangleq \Big\{(M_1,M_2) \Big| & \ M_1,M_2 : \tau \wedge\\
& \big(\forall V_1,\ M_1 \Downarrow V_1 \implies \exists V_2,\ M_2 \Downarrow V_2 \wedge (V_1,V_2) \in \mathcal{V}[\![\tau]\!]\big)\\
\wedge & \big(\forall V_2,\ M_2 \Downarrow V_2 \implies \exists V_1,\ M_1 \Downarrow V_1 \wedge (V_1,V_2) \in \mathcal{V}[\![\tau]\!]\big)\Big\}
\end{aligned}
$$

Remark In $\mathcal{V}[\![\sigma \to \sigma]\!]$, all values are functions. Hence, we could have *equivalently* defined:

$$
\begin{aligned}
\mathcal{V}[\![\tau \to \sigma]\!] \triangleq \{((\lambda x{:}\tau.\, M_1),(\lambda x{:}\tau.\, M_2)) \mid & (\lambda x{:}\tau.\, M_1),(\lambda x{:}\tau.\, M_2) \vdash \tau \to \sigma \wedge\\
& \forall (W_1,W_2) \in \mathcal{V}[\![\tau]\!],\ ((\lambda x{:}\tau.\, M_1)\, W_1,(\lambda x{:}\tau.\, M_2)\, W_2) \in \mathcal{E}[\![\sigma]\!]\}
\end{aligned}
$$

This formulation is more explicit, but less concise.

Properties of logical equivalence for closed terms

Closure by reduction By definition, since reduction is deterministic: Assume $M_1 \Downarrow N_1$ and $M_2 \Downarrow N_2$ and $(M_1,M_2) \in \mathcal{E}[\![\tau]\!]$, *i.e.* there exists $(V_1,V_2) \in \mathcal{V}[\![\tau]\!]$ (**1**) such that $M_i \Downarrow V_i$. Since reduction is deterministic, we must have $M_i \Downarrow N_i \Downarrow V_i$. This, together with (1), implies $(N_1,M_2) \in \mathcal{E}[\![\tau]\!]$.

Closure by inverse reduction Immediate, by construction of $\mathcal{E}[\![\tau]\!]$.

Corollaries

- If $(M_1,M_2) \in \mathcal{E}[\![\tau \to \sigma]\!]$ and $(N_1,N_2) \in \mathcal{E}[\![\tau]\!]$, then $(M_1\, N_1, M_2\, N_2) \in \mathcal{E}[\![\sigma]\!]$.

- To prove $(M_1, M_2) \in \mathcal{E}[\![\tau \to \sigma]\!]$, it suffices to show $(M_1\ V_1, M_2\ V_2) \in \mathcal{E}[\![\sigma]\!]$ for all $(V_1, V_2) \in \mathcal{V}[\![\tau]\!]$.

Consistency $(\sim_{\mathsf{B}}) \subseteq (\simeq)$.

Immediate, by definition of $\mathcal{E}[\![\mathsf{B}]\!]$ and $\mathcal{V}[\![\mathsf{B}]\!] \subseteq (\simeq)$.

Lemma 40 (Symmetry and transitivity) *Logical equivalence is symmetric and transitive (at any given type).*

Notice that *reflexivity is not at all obvious*! This will be the fundamental lemma of logical relations.

Proof: We show it simultaneously for $\sim_\tau$ and $\approx_\tau$ by induction on type τ. For $\sim_\tau$, the proof is immediate by transitivity and symmetry of $\approx_\tau$. For $\approx_\tau$, it goes as follows:

Case τ is B: the result is immediate by symmetry and transitivity of behavioral equivalence.

Case τ is $\tau \to \sigma$:

By IH, symmetry and transitivity hold at types τ and σ.

For symmetry, assume $V_1 \approx_{\tau\to\sigma} V_2$ (**1**), we must show $V_2 \approx_{\tau\to\sigma} V_1$. Assume $W_1 \approx_\tau W_2$. We must show $V_2\ M_1 \sim_\sigma V_1\ W_2$ (**2**). We have $W_2 \approx_\tau W_1$ by symmetry at type τ. By (1), we have $V_2\ W_2 \sim_\sigma V_1\ W_1$ and (2) follows by symmetry of $\sim$ at type σ.

For transitivity, assume $V_1 \approx_\tau V_2$ (**3**) and $V_2 \approx_\tau V_3$ (**4**). To show $V_1 \approx_\tau V_3$, we assume $W_1 \sim_\tau W_3$ and show $V_1\ W_1 \sim_\sigma V_3\ W_3$ (**5**). By (3), we have $V_1\ W_1 \sim_{\tau_2} V_2\ W_3$ (**6**). By symmetry and transitivity of $\approx_\tau$, we get $W_3 \approx_\tau W_3$ (**7**). By (4), we have $V_2\ W_3 \sim_\sigma V_3\ W_3$ (**8**). Then (2) follows by transitivity of $\sim_\sigma$ applied to (6) and (8).

Remark: that (7) is not using reflexivity, which has not been proved yet: this equality follows from the fact that W_3 is already known to be in relation with W_1.

8.4.2 Logical equivalence for open terms

When $\Gamma \vdash M_1, M_2 : \tau$, we wish to define a judgment $\Gamma \vdash M_1 \sim M_2 : \tau$ to mean that the open terms M_1 and M_2 are equivalent at type τ.

The solution is to interpret program variables of $\mathsf{dom}(\Gamma)$ by pairs of related values, and typing contexts Γ by sets of bisubstitutions γ mapping variable type assignments to pairs of related values at the given type:

$$\begin{aligned} \mathcal{G}[\![\varnothing]\!] &\triangleq \{\varnothing\} \\ \mathcal{G}[\![\Gamma, x : \tau]\!] &\triangleq \{\gamma, x \mapsto (V_1, V_2) \mid \gamma \in \mathcal{G}[\![\Gamma]\!] \wedge (V_1, V_2) \in \mathcal{V}[\![\tau]\!]\} \end{aligned}$$

Given a bisubstitution γ, we write γ_i for the substitution that maps x to V_i whenever γ maps x to (V_1, V_2).

Definition 9 *(Logical equivalence for open terms)*

$$\Gamma \vdash M_1 \sim M_2 : \tau \quad \overset{\mathit{def}}{\iff} \quad \begin{cases} \Gamma \vdash M_1, M_2 : \tau \\ \forall \gamma \in \mathcal{G}[\![\Gamma]\!],\ \ (\gamma_1 M_1, \gamma_2 M_2) \in \mathcal{E}[\![\tau]\!] \end{cases}$$

We also write $\vdash M_1 \sim M_2 : \tau$ *or* $M_1 \sim_\tau M_2$ *for* $\varnothing \vdash M_1 \sim M_2 : \tau$*.*

Lemma 41 *Open logical equivalence is symmetric and transitive.*

Proof: The Proof is immediate by the definition and the symmetry and transitivity of closed logical equivalence.)

Theorem 19 (Reflexivity) *If* $\Gamma \vdash M : \tau$*, then* $\Gamma \vdash M \sim M : \tau$*.*

The is also called the fundamental lemma of logical relations. The proof is by induction on the typing derivation, using compatibility lemmas.

Compatibility lemmas

$$\textsc{C-True}\quad \Gamma \vdash \mathsf{tt} \sim \mathsf{tt} : \mathsf{bool} \qquad \textsc{C-False}\quad \Gamma \vdash \mathsf{ff} \sim \mathsf{ff} : \mathsf{bool}$$

$$\textsc{C-If}\quad \frac{\Gamma \vdash M_1 \sim M_2 : \mathsf{B} \qquad \Gamma \vdash N_1 \sim N_2 : \tau \qquad \Gamma \vdash N_1' \sim N_2' : \tau}{\Gamma \vdash \mathsf{if}\ M_1\ \mathsf{then}\ N_1\ \mathsf{else}\ N_1' \sim \mathsf{if}\ M_2\ \mathsf{then}\ N_2\ \mathsf{else}\ N_2' : \tau}$$

$$\textsc{C-Var}\quad \frac{x : \tau \in \Gamma}{\Gamma \vdash x \sim x : \tau} \qquad \textsc{C-Abs}\quad \frac{\Gamma, x : \tau \vdash M_1 \sim M_2 : \sigma}{\Gamma \vdash \lambda x{:}\tau.\, M_1 \sim \lambda x{:}\tau.\, M_2 : \tau \to \sigma} \qquad \textsc{C-App}\quad \frac{\Gamma \vdash M_1 \sim M_2 : \tau \to \sigma \qquad \Gamma \vdash N_1 \sim N_2 : \tau}{\Gamma \vdash M_1\ N_1 \sim M_2\ N_2 : \sigma}$$

Proof: Each case can be shown independently.

Rule C-Abs: Assume $\Gamma, x : \tau \vdash M_1 \sim M_2 : \sigma$ (**1**). We show $\Gamma \vdash \lambda x{:}\tau.\, M_1 \sim \lambda x{:}\tau.\, M_2 : \tau \to \sigma$. Let γ be in $\mathcal{G}[\![\Gamma]\!]$. We show $(\gamma_1(\lambda x : \tau.\, M_1), \gamma_2(\lambda x : \tau.\, M_2)) \in \mathcal{V}[\![\tau \to \sigma]\!]$. Let (V_1, V_2) be in $\mathcal{V}[\![\tau]\!]$. It suffices to show that $(\gamma_1(\lambda x{:}\tau.\, M_1)\ V_1, \gamma_2(\lambda x{:}\tau.\, M_2)\ V_2) \in \mathcal{E}[\![\sigma]\!]$ (**2**).

Let γ' be $\gamma, x \mapsto (V_1, V_2)$. We have γ' in $\mathcal{G}[\![\Gamma, x : \tau]\!]$. Thus, from (1), we have $(\gamma_1' M_1, \gamma_2' M_2)$ in $\mathcal{E}[\![\sigma]\!]$, which proves (2), since $\mathcal{E}[\![\sigma]\!]$ is closed by inverse reduction and $\gamma_i(\lambda x{:}\tau.\, M_i)\ V_i \Downarrow \gamma_i' M_i$.

Rule C-App *and* C-If: By induction hypothesis and the fact that substitution distributes over application.

We must show $\Gamma \vdash M_1\ N_1 \sim M_2\ M_2 : \sigma$ (**3**). Let γ be in $\mathcal{G}[\![\Gamma]\!]$. From the premises $\Gamma \vdash M_1 \sim M_2 : \tau \to \sigma$ and $\Gamma \vdash N_1 \sim N_2 : \tau$, we have $(\gamma_1 M_1, \gamma_2 M_2)$ in $\mathcal{E}[\![\tau \to \sigma]\!]$ and $(\gamma_1 N_1, \gamma_2 N_2)$ in $\mathcal{E}[\![\tau]\!]$. Therefore $(\gamma_1 M_1\ \gamma_1 N_1, \gamma_2 M_2\ \gamma_2 N_2)$, *i.e.* $(\gamma_1(M_1\ N_1), \gamma_2(M_2\ N_2)) \in \mathcal{E}[\![\sigma]\!]$ in $\mathcal{E}[\![\sigma]\!]$, which proves (3).

Rule C-If: Similar to the case of application.
We show $\Gamma \vdash \mathsf{if}\ M_1\ \mathsf{then}\ N_1\ \mathsf{else}\ N_1' \sim \mathsf{if}\ M_2\ \mathsf{then}\ N_2\ \mathsf{else}\ N_2' : \tau$. Assume γ in $\mathcal{G}\llbracket\gamma\rrbracket$. We show $(\gamma_1(\mathsf{if}\ M_1\ \mathsf{then}\ N_1\ \mathsf{else}\ N_1'), \gamma_2(\mathsf{if}\ M_2\ \mathsf{then}\ N_2\ \mathsf{else}\ N_2'))$ in $\mathcal{E}\llbracket\tau\rrbracket$ (**1**).

From the premise $\Gamma \vdash M_1 \sim M_2 : \mathsf{B}$, we have $(\gamma_1 M_1, \gamma_2 M_2) \in \mathcal{E}\llbracket\mathsf{B}\rrbracket$. Therefore $M_1 \Downarrow V$ and $M_2 \Downarrow V$ where V is either tt or ff.

Case V is tt:. Then, $(\mathsf{if}\ \gamma_i M_i\ \mathsf{then}\ \gamma_i N_i\ \mathsf{else}\ \gamma_i N_i') \Downarrow \gamma_i N_i$, *i.e.* $\gamma_i(\mathsf{if}\ M_i\ \mathsf{then}\ N_i\ \mathsf{else}\ N_i') \Downarrow \gamma_i N_i$. From the premise $\Gamma \vdash N_1 \sim N_2 : \tau$, we have $(\gamma_1 N_1, \gamma_2 N_2)$ in $\mathcal{E}\llbracket\tau\rrbracket$ and (1) follows by closure by inverse reduction.

Case V is ff: similar.

Rule C-True, C-False, *and* C-Var: are immediate.

Proof: (of reflexivity)

By induction on the proof of $\Gamma \vdash M : \tau$. All cases are easy. We must show $\Gamma \vdash M \sim M : \tau$:

Case M is tt *or* ff: Immediate by Rule C-True or Rule C-False

Case M is x: Immediate by Rule C-Var.

Case M is $M'\ N$: By inversion of the typing rule App, induction hypothesis, and Rule C-App.

Case M is $\lambda\tau : N.$: By inversion of the typing rule Abs, induction hypothesis, and Rule C-Abs.

Properties of logical relations

Corollary 42 *(equivalence) Open logical relation is an equivalence relation*

Lemma 43 *Logical equivalence is a congruence.*
If $\Gamma \vdash M \sim M' : \tau$ *and* $\mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\Delta \rhd \sigma)$, *then* $\Delta \vdash \mathcal{C}[M] \sim \mathcal{C}[M'] : \sigma$.

Proof: By induction on the proof of $\mathcal{C} : (\Gamma \rhd \tau) \rightsquigarrow (\Delta \rhd \sigma)$.

The proof is similar to the proof of reflexivity—but we need a syntactic definition of context typing derivations (which we have omitted) to be able to reason by induction on the context typing derivations.

Theorem 20 (Soundness of logical equivalence) *Logical equivalence implies observational equivalence. If* $\Gamma \vdash M \sim M' : \tau$ *then* $\Gamma \vdash M \cong M' : \tau$.

Proof: Logical equivalence is a consistent congruence, hence included in observational equivalence which is the coarsest such relation.

Theorem 21 (Completeness of logical equivalence) *Observational equivalence of closed terms implies logical equivalence. That is* $(\cong_\tau) \subseteq (\sim_\tau)$.

Proof: Proof by induction on τ.

Case B: In the empty context, by consistency, $\cong_\mathsf{B}$ is a subrelation of $\simeq_\mathsf{B}$ which coincides with $\sim_\mathsf{B}$.

Case $\tau \to \sigma$: By congruence of observational equivalence.

By hypothesis, we have $M_1 \cong_{\tau\to\sigma} M_2$ (**1**). To show $M_1 \sim_{\tau\to\sigma} M_2$, we assume $V_1 \approx_\tau V_2$ (**2**) and then, it suffices to show $M_1\, V_1 \sim_\sigma M_2\, V_2$ (**3**).

By soundness applied to (2), we have $V_1 \cong_\tau V_2$ from (**4**). By congruence with (1), we have $M_1\, V_1 \cong_\sigma M_2\, V_2$, which implies (3) by IH at type σ.

Exercise 49 *(Application) Let not be* $\lambda x : \mathsf{B}.\, \mathsf{if}\ x\ \mathsf{then}\ \mathsf{ff}\ \mathsf{else}\ \mathsf{tt}$ *and* M *and* M' *be the expressions* $\lambda x : \mathsf{B}.\, \lambda y : \tau.\, \lambda z : \tau.\, \mathsf{if}\ \mathit{not}\ x\ \mathsf{then}\ y\ \mathsf{else}\ z$ *and* $\lambda x : \mathsf{B}.\, \lambda y : \tau.\, \lambda z : \tau.\, \mathsf{if}\ x\ \mathsf{then}\ z\ \mathsf{else}\ y$. *Show that* $M \cong_{\mathsf{B}\to\tau\to\tau\to\tau} M'$. □

Solution: It suffices to show $M\ V_0\ V_1\ V_2 \sim_\tau M'\ V_0'\ V_1'\ V_2'$ *whenever* $V_0 \approx_\mathsf{B} V_0'$ *(**1**) and* $V_1 \approx_\tau V_1'$ *(**2**) and* $V_2 \approx_\tau V_2'$ *(**3**). By inverse reduction, it suffices to show:* $\mathsf{if}\ \mathit{not}\ V_0\ \mathsf{then}\ V_1\ \mathsf{else}\ V_2 \sim_\tau \mathsf{if}\ V_0'\ \mathsf{then}\ V_2'\ \mathsf{else}\ V_1'$ *(**4**). It follows from (1) that we have only two cases:*

Case $V_0 = V_0' = \mathsf{tt}$*: Then not* $V_0 \Downarrow \mathsf{ff}$ *and thus* $M \Downarrow V_2$ *while* $M' \Downarrow V_2$. *Then (4) follows from (3) and closure by inverse reduction.*

Case $V_0 = V_0' = \mathsf{ff}$*: is symmetric.*

8.5 Logical relations in F

We now extend observational and logical equivalence to System F.

$$\tau ::= \ldots \mid \alpha \mid \forall\alpha.\,\tau \qquad\qquad M ::= \ldots \mid \Lambda\alpha.M \mid M\ \tau$$

We write typing contexts $\Delta;\Gamma$ where Δ binds type variables and Γ binds program variables. Typing of contexts becomes $\mathcal{C} : (\Delta;\Gamma \rhd \tau) \rightsquigarrow (\Delta';\Gamma' \rhd \tau')$.

Definition 10 *(observational equivalence) We defined* $\Delta;\Gamma \vdash M \cong N : \tau$ *as*

$$\forall \mathcal{C} : (\Delta;\Gamma \rhd \tau) \rightsquigarrow (\varnothing;\varnothing \rhd \mathsf{B}),\ \ \mathcal{C}[M] \simeq \mathcal{C}[N]$$

We write $M \cong_\tau N$ *for* $\varnothing;\varnothing \vdash M \cong N : \tau$ *(in particular, when* τ*,* M*, and* N *are closed).*

8.5.1 Logical equivalence for closed terms

For closed terms (no free program variables), we now need to give the semantics of polymorphic types $\forall\alpha.\tau$. Unfortunately, it cannot be defined in terms of the semantics of instances $\tau[\alpha \mapsto \sigma]$, since the semantics is defined by induction on types.

The work around is to define the semantics of terms with open types in some suitable environment that interprets type variables by relations (sets of pairs of related values) of closed types.

This relation will also be used to give the the semantics of type variables. A key point however, it to interpret type variables by heterogeneous values, relating values of different types on both sides.

We write ρ for closed types. Let $\mathcal{R}(\rho_1, \rho_2)$ be the set of relations on values of closed types ρ_1 and ρ_2, that is, $\mathcal{P}(\mathsf{Val}(\rho_1) \times \mathsf{Val}(\rho_2))$. We *optionally* restrict all such relations to be *admissible*, and we write $\mathcal{R}^\sharp(\rho_1, \rho_2)$ the subset of admissible relations, which in our setting means closed by observational equivalence, *i.e.*

$$
\begin{aligned}
R \in \mathcal{R}^\sharp(\rho_1, \rho_2) \overset{\mathsf{def}}{\iff}&\\
&\forall (V_1, V_2) \in R,\ \forall W_1, W_2,\ W_1 \cong_{\rho_1} V_1 \wedge W_2 \cong_{\rho_2} V_2 \implies (W_1, W_2) \in R
\end{aligned}
$$

Admissibility will be required for completeness of logical relations with respect to observational equivalence. However, it is not required for soundness of logical relations. Choosing relations that are not admissible is sometimes easier when one only soundness of logical relations is needed.

Example 1 *Both* $R_1 \triangleq \{(\mathsf{tt}, 0), (\mathsf{ff}, 1)\}$ *and* $R_2 \triangleq \{(\mathsf{tt}, 0)\} \cup \{(\mathsf{ff}, n) \mid n \in \mathbb{Z}^\star\}$ *are admissible relations in* $\mathcal{R}(\mathsf{B}, \mathsf{int})$*. By contrast* $R_3 \triangleq \{(\mathsf{tt}, \lambda x\!:\!\tau.\, 0), (\mathsf{ff}, \lambda x\!:\!\tau.\, 1)\}$ *is in* $\mathcal{R}(\mathsf{B}, \tau \to \mathsf{int})$ *but it is not admissible. Indeed, taking* $M_0 \triangleq \lambda x\!:\!\tau.\,(\lambda z\!:\!\mathsf{int}.\, z)\ 0$*. we have* $M_0 \cong_{\tau\to\mathsf{int}} \lambda x\!:\!\tau.\, 0$ *but* (tt, M_0) *is not in* R_3*.*

Interpretation of type environments We interpret type variables α by triples of the form (ρ_1, ρ_2, R) where $R \in \mathcal{R}(\rho_1, \rho_2)$. We write η for mappings of type variables to such triples. Given a list of type variables Δ, we define the set $\mathcal{D}[\![\Delta]\!]$ of interpretations of Δ as:

$$
\begin{aligned}
\mathcal{D}[\![\varnothing]\!] &\triangleq \{\varnothing\}\\
\mathcal{D}[\![\Delta, \alpha]\!] &\triangleq \{\eta, \alpha \mapsto (\rho_1, \rho_2, R) \mid \eta \in \mathcal{D}[\![\Delta]\!] \wedge R \in \mathcal{R}(\rho_1, \rho_2)\}
\end{aligned}
$$

Definition 11 (Logical equivalence for closed terms)

$$
\begin{aligned}
\mathcal{V}[\![\alpha]\!]_\eta &\triangleq \eta_R(\alpha)\\
\mathcal{V}[\![\forall\alpha.\tau]\!]_\eta &\triangleq \{(V_1,V_2) \mid V_1 \vdash \eta_1(\forall\alpha.\tau) \wedge V_2 \vdash \eta_2(\forall\alpha.\tau) \wedge\\
&\qquad \forall \rho_1,\rho_2, R \in \mathcal{R}(\rho_1,\rho_2), (V_1\ \rho_1, V_2\ \rho_2) \in \mathcal{E}[\![\tau]\!]_{\eta,\alpha\mapsto(\rho_1,\rho_2,R)}\}\\
\mathcal{V}[\![\mathsf{B}]\!]_\eta &\triangleq \{(\mathsf{tt},\mathsf{tt}),(\mathsf{ff},\mathsf{ff})\}\\
\mathcal{V}[\![\tau\to\sigma]\!]_\eta &\triangleq \{(V_1,V_2) \mid V_1 \vdash \eta_1(\tau\to\sigma) \wedge V_2 \vdash \eta_2(\tau\to\sigma) \wedge\\
&\qquad \forall (W_1,W_2) \in \mathcal{V}[\![\tau]\!]_\eta,\ (V_1\ W_1, V_2\ W_2) \in \mathcal{E}[\![\sigma]\!]_\eta\}\\
\mathcal{E}[\![\tau]\!]_\eta &\triangleq \{(M_1,M_2) \mid M_1 : \eta_1\tau \wedge M_2 : \eta_2\tau \wedge\\
&\qquad \exists (V_1,V_2) \in \mathcal{V}[\![\tau]\!]_\eta, M_1 \Downarrow V_1 \wedge M_2 \Downarrow V_2\}\\
\mathcal{G}[\![\varnothing]\!]_\eta &\triangleq \{\varnothing\}\\
\mathcal{G}[\![\Gamma, x:\tau]\!]_\eta &\triangleq \{\gamma, x\mapsto(V_1,V_2)\ \mid\ \gamma\in\mathcal{G}[\![\Gamma]\!]_\eta \wedge (V_1,V_2)\in\mathcal{V}[\![\tau]\!]_\eta\}
\end{aligned}
$$

Notice that there are really just two new cases $\mathcal{V}[\![\alpha]\!]_\eta$ and $\mathcal{V}[\![\forall\alpha.\tau]\!]_\eta$, as the other cases are just adjusting the previous definition to carry around the environment η (which we have here typeset in highlighted to emphasize the minor difference).

Notice again that $\forall\alpha.\tau$ is interpreted by choosing two different types ρ_1 and ρ_2 and therefore heterogeneous pairs of types in $\mathcal{R}(\rho_1,\rho_2)$ to interpret α.

Definition 12 *(Logical equivalence for open terms) We say* $\Delta;\Gamma \vdash M \sim M' : \tau$ *as*

$$
\wedge\left\{\begin{array}{l}
\Delta;\Gamma \vdash M, M' : \tau\\
\forall \eta \in \mathcal{D}[\![\Delta]\!],\ \forall\gamma\in\mathcal{G}[\![\Gamma]\!]_\eta,\ (\eta_1(\gamma_1 M_1), \eta_2(\gamma_2 M_2)) \in \mathcal{E}[\![\tau]\!]_\eta
\end{array}\right.
$$

We also write $M_1 \sim_\tau M_2$ for $\vdash M_1 \sim M_2 : \tau$ (*i.e.* $\varnothing;\varnothing \vdash M_1 \sim M_2 : \tau$). In this case, τ is a closed type and M_1 and M_2 are closed terms of type τ; hence, this coincides with the previous definition (M_1, M_2) in $\mathcal{E}[\![\tau]\!]_\varnothing$, which may still be used as a shorthand for $\mathcal{E}[\![\tau]\!]$.

Lemma 44 (Compositionality)
Assume $\Delta \vdash \sigma$ *and* $\Delta, \alpha \vdash \tau$ *and* $\eta \in \mathcal{D}[\![\Delta]\!]$. *Then,* $\mathcal{V}[\![\tau[\alpha\mapsto\sigma]]\!]_\eta = \mathcal{V}[\![\tau]\!]_{\eta,\alpha\mapsto(\eta_1\sigma,\eta_2\sigma,\mathcal{V}[\![\sigma]\!]_\eta)}$.

Proof: Let us write θ for $[\alpha\mapsto\sigma]$ and η^α for $\eta,\alpha\mapsto(\eta_1\sigma,\eta_2\sigma,R)$. We show $\mathcal{V}[\![\tau\theta]\!]_\eta = \mathcal{V}[\![\tau]\!]_{\eta^\alpha}$. by induction on τ.

Case τ is α: The right-hand side $\mathcal{V}[\![\alpha]\!]_{\eta^\alpha}$ is by definition $\eta^\alpha_R(\alpha)$, which is $R(\alpha)$, *i.e.* $\mathcal{V}[\![\sigma]\!]_\eta$ by hypothesis.

Case τ is $\sigma\to\sigma'$: Since $(\sigma\to\sigma')\theta$ is $\sigma\theta\to\sigma'\theta$, the left-hand side is $\mathcal{V}[\![\sigma\theta\to\sigma'\theta]\!]_\eta$, *i.e.* by definition:

$$\{(V_1,V_2) \mid \forall (W_1,W_2)\in\mathcal{V}[\![\sigma\theta]\!]_\eta, (V_1\ W_1, V_2\ W_2)\in\mathcal{E}[\![\sigma'\theta]\!]_\eta\}$$

By induction hypothesis, we may replaced $\mathcal{V}[\![\sigma\theta]\!]_\eta$ by $\mathcal{V}[\![\sigma]\!]_{\eta^\alpha}$ and $\mathcal{E}[\![\sigma'\theta]\!]_\eta$ by $\mathcal{E}[\![\sigma']\!]_{\eta^\alpha}$ which gives exactly the definition of the right-hand side $\mathcal{V}[\![\sigma \to \sigma']\!]_{\eta^\alpha}$.

Case τ is B: Both sides are equal to $\mathcal{V}[\![\mathsf{B}]\!]$.

Case τ is $\forall\beta.\,\sigma$: Assume $\alpha \ne \beta$. Since $\theta(\forall\alpha.\,\sigma)$ is then $\forall\alpha.\,\theta\sigma$, the left-hand side is $\mathcal{V}[\![\forall\alpha.\,\sigma\theta]\!]_\eta$ which is, by definition:

$$\{(V_1, V_2) \mid \forall\rho_1, \rho_2, \forall S \in \mathcal{R}(\rho_1, \rho_2), (V_1\ \rho_1, V_2\ \rho_2) \in \mathcal{E}[\![\sigma\theta]\!]_{\eta,\beta\mapsto(\rho_1,\rho_2,S)}\}$$

Since R and S are relations between closed types the substitutions $\alpha \mapsto (\tau_1, \tau_2, R)$ and $\beta \mapsto (\rho_1, \rho_2, S)$ commute. Thus, by induction hypothesis, we may replace $\mathcal{E}[\![\sigma\theta, \beta]\!]_\eta$ by $\mathcal{E}[\![\sigma]\!]_{\eta^\alpha,\beta\mapsto(\rho_1,\rho_2,S)}$, which gives the definition of the right-hand side.

Theorem 22 (Reflexivity, also called the fundamental lemma)
If $\Delta; \Gamma \vdash M : \tau$ then $\Delta; \Gamma \vdash M \sim M : \tau$.

Admissibility is not required for the fundamental lemma.

Proof: By induction on the typing derivation of $\Delta; \Gamma \vdash M : \tau$, using compatibility lemmas.

Lemma 45 (Compatibility lemmas) *We redefined previous the lemmas to work in a typing context of the form Δ, Γ instead of Γ. In addition, we have:*

$$\textsc{C-Tabs}\ \frac{\Delta, \alpha; \Gamma \vdash M_1 \sim M_2 : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.M_1 \sim \Lambda\alpha.M_2 : \forall\alpha.\,\tau} \qquad \textsc{C-Tapp}\ \frac{\Delta; \Gamma \vdash M_1 \sim M_2 : \forall\alpha.\,\tau \qquad \Delta \vdash \sigma}{\Delta; \Gamma \vdash M_1\ \sigma \sim M_2\ \sigma : \tau[\alpha \mapsto \sigma]}$$

Proof: We show each rule independently. In each case, the typing conditions follow immediately from the mimicking of the typing rules.

Rule C-Tabs: Assume $\Delta, \alpha; \Gamma \vdash M_1 \sim M_2 : \tau$ (**1**). We show $\Delta; \Gamma \vdash \Lambda\alpha.N \sim \Lambda\alpha.N : \forall\alpha.\,\tau$.

Let $\eta \in \mathcal{D}[\![\Delta]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\eta$. We show $(\eta_1(\gamma_1(\Lambda\alpha.M_1)), \eta_2(\gamma_2(\Lambda\alpha.M_2))) \in \mathcal{E}[\![\forall\alpha.\,\tau]\!]_\eta$, *i.e.* $((\eta_1(\gamma_1(\Lambda\alpha.M_1)))\ \rho_1, (\eta_2(\gamma_2(\Lambda\alpha.M_2)))\ \rho_2) \in \mathcal{E}[\![\tau]\!]_{\eta,\alpha\mapsto(\rho_1,\rho_2,R)}$ (**2**), for any ground types ρ_1 and ρ_2 and $R \in \mathcal{R}(\rho_1, \rho_2)$.

We may assume $\alpha \notin \mathsf{dom}(\gamma)$ *w.l.o.g.*. Then $(\eta_i(\gamma_i(\Lambda\alpha.M_i)))\ \rho_i$ is equal to $\eta_i((\Lambda\alpha.\gamma_i M_i)\ \rho_i)$ which reduces to $\eta_1(\gamma_i(M_i[\alpha \mapsto \rho_i]))$, *i.e.* $\eta_1(\gamma_i'(M_i))$ where γ_i' is $\gamma_i, \alpha \mapsto \rho_i$.

Since $\gamma_i' \in \mathcal{D}[\![\Delta, \alpha]\!]$, we have by $(\eta_1(\gamma_1'(M_1)), \eta_2(\gamma_2'(M_2))) \in \mathcal{E}[\![\tau]\!]_{\eta,\alpha\mapsto(\rho_1,\rho_2,R)}$ by IH applied to (1), from which (2) follows by closure under inverse reduction.

Rule C-Tapp: Assume $\Delta; \Gamma \vdash M_1 \sim M_2 : \forall\alpha.\,\tau$ (**1**) and $\Delta \vdash \sigma$. We show $\Delta; \Gamma \vdash M_1\ \sigma \sim M_2\ \sigma : \tau[\alpha \mapsto \sigma]$. Let $\eta \in \mathcal{D}[\![\Delta]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\eta$. We just need to show $(\eta_1\gamma_1(M_1\ \sigma), \eta_2\gamma_2(M_2\ \sigma))$

in $\mathcal{E}[\![\tau[\alpha \mapsto \sigma]]\!]_\eta$ (**2**). From (1), we have $(\eta_1\gamma_1 M_1, \eta_2\gamma_2 M_2)$ in $\mathcal{E}[\![\forall\alpha.\tau]\!]_\eta$. By definition, this implies $((\eta_1\gamma_1 M_1)\ (\eta_1\sigma), (\eta_2\gamma_2 M_2)\ (\eta_2\sigma))$, *i.e.*, $(\eta_1\gamma_1(M_1\ \sigma), \eta_2\gamma_2(M\ \sigma))$ is in $\mathcal{E}[\![\tau]\!]_{\eta'}$ where η' is $\eta, \alpha \mapsto (\eta_1\sigma, \eta_2\sigma, \mathcal{V}[\![\sigma]\!]_\eta)$, which exactly proves (2) *by compositionality.* (Notice, that by corollary 47 this relation is admissible if we are working under the admissibility assumption.)

Other rules : their proof is quite similar to the same corresponding rule for closed types.

Theorem 23 (Soundness of logical equivalence) *Logical equivalence implies implies observational equivalence. That is, if* $\Delta; \Gamma \vdash M_1 \sim M_2 : \tau$ *then* $\Delta; \Gamma \vdash M_1 \cong M_2 : \tau$.

Lemma 46 (Respect for observational equivalence) *Under the admissibility condition, If* $(M_1, M_2) \in \mathcal{E}[\![\tau]\!]^\sharp_\eta$ *and* $M_1 \cong_{\eta_1\tau} N_1$ *and* $M_2 \cong_{\eta_2\tau} N_2$*, then* $(N_1, N_2) \in \mathcal{E}[\![\tau]\!]^\sharp_\eta$.

Proof: By symmetry, we may just show it when N_2 is M_2, the case when N_1 is M_1 is symmetric and the general case follows by two applications of of the lemma that falls in the two previous cases.

We assume $(M_1, M_2) \in \mathcal{E}[\![\tau]\!]^\sharp_\eta$ (**1**) and $M_1 \cong_{\eta_1\tau} N_1$ (**2**). We show $(N_1, M_2) \in \mathcal{E}[\![\tau]\!]^\sharp_\eta$ (**3**) by induction on τ.

Case τ is $\forall\alpha.\sigma$: Assume $R \in \mathcal{R}(\rho_1, \rho_2)$. Let η^α be $\eta, \alpha \mapsto (\rho_1, \rho_2, R)$. It suffices to show $(M_1\ \rho_1, M_2\ \rho_2)$ in $\mathcal{E}[\![\sigma]\!]^\sharp_{\eta_\alpha}$ (**4**). We have $(M_1\ \rho_1, M_2\ \rho_2)$ in $\mathcal{E}[\![\sigma]\!]^\sharp_{\eta_\alpha}$, from (1). By congruence applied to (2), we have $N_1\rho_1 \cong_{\eta_1^\alpha\sigma} M_1\ \rho_1$. Then (4) follows by induction hypothesis at type σ.

Case τ is α: We know that (M_1, M_2) reduces to a pair (V_1, V_2) in $\mathcal{V}[\![\alpha]\!]^\sharp_\eta$, *i.e.* $\eta_R(\alpha)$, which is by assumption is admissible, *i.e.* closed by observational equivalence (between values). Therefore, we just need to show that $V \cong_{\eta_1\alpha} V_1$ where V is such that $N_1 \Downarrow V$. This follows from $N_1 \cong_{\eta_1\alpha} M_1$ since observational equivalence of closed terms is closed by reduction.

Case τ is B: By definition $\mathcal{E}[\![\mathsf{B}]\!]^\sharp_\eta$ does not depend on η and is equal to $\simeq_{\mathsf{B}}$, which is included in $\cong_{\mathsf{B}}$ and closed by transitivity.

Case τ is $\sigma' \to \sigma$: Assume V_1, V_2 is in $\mathcal{E}[\![\sigma']\!]^\sharp_\eta$ (**5**). It suffices to show that $(N_1\ V_1, M_2\ V_2)$ is in $\mathcal{E}[\![\sigma]\!]^\sharp_\eta$ (**6**). By (1), we have $(M_1\ V_1, M_2\ V_2)$ in $\mathcal{E}[\![\sigma']\!]^\sharp_\eta$. By congruence applied to (2), we have $N_1V_1 \cong_{\eta_1(\sigma)} M_1V_1$. Then (6) follows by IH, since then $\mathcal{E}[\![\sigma]\!]^\sharp_\eta$ respects observational equivalence..

Corollary 47 *Under the admissibility condition, the relation* $\mathcal{V}[\![\tau]\!]^\sharp_\eta$ *is an admissible relation in* $\mathcal{R}(\eta_1\tau, \eta_2\tau)$.

This may be useful to build admissibility relations, when admissibility is required.

Lemma 48 (Closure by observational equivalence) *Under the admissibility condition, if* $\Delta; \Gamma \vdash M_1 \sim_\sharp M_2 : \tau$ *and* $\Delta; \Gamma \vdash M_1 \cong N_1 : \tau$ *and* $\Delta; \Gamma \vdash M_2 \cong N_2 : \tau$*, then* $\Delta; \Gamma \vdash N_1 \sim_\sharp N_2 : \tau$

This lemma is use in the proof of correctness of logical equivalence.

Proof: By symmetry, we may just show it when N_2 is M_2, the case when N_1 is M_1 is symmetric and the general case follows by two applications of of the lemma that falls in the two previous cases.

The proof is by induction on τ.

Assume that $\Delta, \Gamma \vdash M_1 \sim_\sharp M_2 : \tau$ (**1**) and $\Delta; \Gamma \vdash N_1 \cong_{\eta_1 \tau} M_1$ (**2**). Assume η in $\mathcal{D}[\![\Delta]\!]$ and γ in $\mathcal{G}[\![\Gamma]\!]^\sharp_\eta$. We are to show that $(\eta_1\gamma_1 N_1, \eta_2\gamma_2 M_2)$ is in $\mathcal{E}[\![\tau]\!]^\sharp_\eta$ (**3**).

Let $\mathcal{C}$ be the context $(\Lambda\Delta.\lambda\Gamma.[\,])\ \eta_1(\Delta)\ \gamma_1(\Gamma)$ where $\eta_1\Delta$ and $\gamma_1\Gamma$ are sequences of ground types and of closed values of ground types taken in the appropriate (*i.e.* reverse) order. We have $\mathcal{C}(\Delta; \Gamma \rhd \tau) \rightsquigarrow (\varnothing; \varnothing \rhd \eta_1\tau)$. It then follows from (2) that $\mathcal{C}[N_1] \cong_{\eta_1\tau} \mathcal{C}[M_1]$, which implies $\eta_1\gamma_1 N_1 \cong_\tau \eta_1\gamma_1 M_1$, since observational equivalence of closed terms is closed by reduction. From (1), we have $(\eta_1\gamma_1 M_1, \eta_2\gamma_2 M_2)$ in $\mathcal{E}[\![\tau]\!]^\sharp_\eta$. Then, the conclusion (3) follows by respect for observational equivalence.

Theorem 24 (Completeness of logical equivalence) *Under the admissibility condition, observational equivalence implies logical equivalence.*
If $\Delta; \Gamma \vdash M_1 \cong M_2 : \tau$ *then* $\Delta; \Gamma \vdash M_1 \sim^\sharp M_2 : \tau$.

In particular, $(\cong_\tau) \subseteq (\sim^\sharp_\tau)$ for closed types τ.

Proof: Assume $\Delta; \Gamma \vdash M_1 \cong M_2 : \tau$. The conclusion $\Delta, \Gamma \vdash M_1 \sim M_2 : \tau$. follows from the fundamental lemma, $\Delta, \Gamma \vdash M_1 \sim M_1 : \tau$ and respect for observation equivalence.

Remark Admissibility is required for completeness, but not for soundness. ($\sim^\sharp$ means $\sim$ when admissibility is required—for all relations.)

As a particular case, for closed terms, we have $M_1 \sim^\sharp_\tau M_2$ iff $M_1 \cong_\tau M_2$.

Lemma 49 (Extensionality)

- $M_1 \cong_{\tau\to\sigma} M_2 \iff \forall V \in \mathsf{Val}(\tau),\ M_1\, V \cong_\sigma M_2\, V \iff \forall N \in \mathsf{Exp}(\tau),\ M_1\, N \cong_\sigma M_2\, N$
- $M_1 \cong_{\forall\alpha.\tau} M_2 \iff \forall \rho,\ M_1\, \rho \cong_{\tau[\alpha\mapsto\rho]} M_2\, \rho.$

Extensionality does not require admissibility—since it does not refer to logical equivalence, but we need admissibility to conduct the proof, which relies on *respect for observational equivalence.*

Proof: We reason under admissibility (left implicit in notations).q The right most equivalence for value abstractions results from the closure of $\mathcal{E}[\![\tau]\!]$ by reduction and anti-reduction.

The forward direction follows in both cases from the congruence of $\cong$. The backward is as follows:

Value abstraction: It suffices to show $M_1 \sim_{\tau\to\sigma} M_2$. That is, assuming $V_1 \approx_\tau V_2$ (**1**), we show $M_1\ V_1 \sim_\sigma M_2\ V_2$ (**2**). By assumption, we have $M_1\ V_1 \cong_\sigma M_2\ V_1$ (**3**). By the *fundamental lemma*, we have $M_2 \sim_{\tau\to\sigma} M_2$. Hence, from (1), read as a logical equivalence, we deduce $M_2\ V_1 \sim_\sigma M_2\ V_2$, We conclude (2) by *respect for observational equivalence* with (3).

Type abstraction: It suffices to show $M_1 \sim_{\forall\alpha.\tau} M_2$. That is, given $R \in \mathcal{R}(\rho_1,\rho_2)$, we show $(M_1\ \rho_1, M_2\ \rho_2) \in \mathcal{E}[\![\tau]\!]_{\alpha\mapsto(\rho_1,\rho_2,R)}$ (**4**). By assumption, we have $M_1\ \rho_1 \cong_{\tau[\alpha\mapsto\rho_1]} M_2\ \rho_1$ (**5**). By the *fundamental lemma*, we have $M_2 \sim_{\forall\alpha.\tau} M_2$. Hence, we have $(M_2\ \rho_1, M_2\ \rho_2) \in \mathcal{E}[\![\tau]\!]_{\alpha\mapsto(\rho_1,\rho_2,R)}$ We conclude (4) by *respect for observational equivalence* with (5).

Identity extension Let θ be a substitution of variables for ground types. Let R be the restriction of $\cong_{\alpha\theta}$ to $\mathsf{Val}(\alpha\theta) \times \mathsf{Val}(\alpha\theta))$ and $\eta : \alpha \mapsto (\alpha\theta, \alpha\theta, R)$. Then $\mathcal{E}[\![\tau]\!]_\eta$ is equal to $\cong_{\tau\theta}$—assuming admissibiliy.

The proof uses respects for observational equivalence.

8.6 Applications

Exercise 50 (Inhabitants of $\forall\alpha.\alpha\to\alpha$) *If $M : \forall\alpha.\alpha \to \alpha$, then $M \cong_{\forall\alpha.\alpha\to\alpha} id$ where $id \triangleq \Lambda\alpha.\lambda x{:}\alpha.\,x$.*

Solution: By *extensionality*, it suffices to show that for any ρ and $V : \rho$ we have $M\ \rho\ V \cong_\rho id\ \rho\ V$. In fact, by closure by inverse reduction, it suffices to show $M\ \rho\ V \cong_\rho V$ (**1**).

By *parametricity*, we have $M \sim_{\forall\alpha.\alpha\to\alpha} M$ (**2**). Consider R in $\mathcal{R}(\rho,\rho)$ equal to $\{(V,V)\}$ and η be $[\alpha \mapsto (\rho,\rho,R)]$. Since $R(V,V)$, we have $(V,V) \in \mathcal{V}[\![\alpha]\!]_\eta$ by definition. Hence, from (2), we have $(M\ \rho\ V, M\ \rho\ V) \in \mathcal{E}[\![\alpha]\!]_\eta$, which means that the pair of expressions $(M\ \rho\ V, M\ \rho\ V)$ reduces to a pair of values in R and, in particular, $M\ \rho\ V$ reduces to V, which implies (1). □

Exercise 51 (Inhabitants of $\forall\alpha.\alpha\to\alpha\to\alpha$) *If $M : \forall\alpha.\alpha \to \alpha \to \alpha$, then either $M \cong_\sigma W_1 \triangleq \Lambda\alpha.\lambda x_1{:}\alpha.\,\lambda x_2{:}\alpha.\,x_1$ or $M \cong_\sigma W_2 \triangleq \Lambda\alpha.\lambda x_1{:}\alpha.\,\lambda x_2{:}\alpha.\,x_2$*

Solution: By *extensionality*, it suffices to show that for either $i = 1$ or $i = 2$, for any closed type ρ and $V_1, V_2 : \rho$, we have $M\ \rho\ V_1\ V_2 \cong_\rho W_i\ \rho\ V_1\ V_2$, or just $M\ \rho\ V_1\ V_2 \cong_\sigma V_i$ (**1**) by closure by inverse reduction, since $W_i\ \rho\ V_1\ V_2$ reduces to V_i

Let ρ and $V_1, V_2 : \rho$ be fixed. Consider R equal to $\{(\mathsf{tt}, V_1), (\mathsf{ff}, V_2)\}$ in $\mathcal{R}(\mathsf{B},\rho)$ and η be $\alpha \mapsto (\mathsf{B},\rho,R)$. We have $(\mathsf{tt}, V_1) \in \mathcal{V}[\![\alpha]\!]_\eta$ since $R(\mathsf{tt}, V_1)$ and, similarly, $(\mathsf{ff}, V_2) \in \mathcal{V}[\![\alpha]\!]_\eta$. By parametricity. we have $(M, M) \in \mathcal{E}[\![\sigma]\!]$. Hence, $(M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff}, M\ \rho\ V_1\ V_2) \in \mathcal{E}[\![\alpha]\!]_\eta$, which means that $(M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff}, M\ \rho\ V_1\ V_2)$ reduces to a pair of values in R, *i.e.* either (tt, V_1) or (ff, V_2), which implies:

$$\begin{array}{rl} \text{either} & M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{tt}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_1 \\ \text{or} & M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{ff}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_2 \end{array}$$

In summary, we have shown

$$\forall \rho, V_1, V_2, \quad \bigvee \begin{cases} M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{tt}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_1 \\ M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{ff}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_2 \end{cases}$$

However, since $M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff}$ is independent of ρ, V_1, and V_2 and the two branches are incompatible as $\mathsf{tt} \ncong \mathsf{ff}$, the choice is actually independent of ρ, V_1 and V_2. Therefore, we also have:

$$\bigvee \begin{cases} \forall \rho, V_1, V_2,\ \ M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{tt}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_1 \\ \forall \rho, V_1, V_2,\ \ M\ \mathsf{B}\ \mathsf{tt}\ \mathsf{ff} \cong_{\mathsf{B}} \mathsf{ff}\ \wedge\ M\ \rho\ V_1\ V_2 \cong_\rho V_2 \end{cases}$$

that is (1). □

Remark *Notice that the proof could have been conducted by choosing* 0 *and* 1 *of type* nat*, or even* W_1 *and* W_2 *of type* σ*, instead of* tt *and* ff *of type* B*.*

Exercise 52 (Inhabitants of $\forall\alpha.\,(\alpha\to\alpha)\to\alpha\to\alpha$) *Let* nat *be* $\forall\alpha.\,(\alpha\to\alpha)\to\alpha\to\alpha$*. If* $M : \mathsf{nat}$*, then* $M \cong_{\mathsf{nat}} N_n$ *for some integer* n*, where* $N_n \triangleq \Lambda\alpha.\lambda f{:}\alpha\to\alpha.\,\lambda x{:}\alpha.\,f^n\ x$*. (That is, the inhabitants of* nat *are the Church naturals.)*

Solution: By *extensionality*, it suffices to show that there exists n such for any closed type ρ and closed values $V_1 : \rho\to\rho$ and $V_2 : \rho$, we have $M\ \rho\ V_1\ V_2 \cong_\rho N_n\ \rho\ V_1\ V_2$, or, by closure by inverse reduction, $M\ \rho\ V_1\ V_2 \cong_\rho V_1^n\ V_2$ (**1**).

Let ρ and $V_1 : \rho\to\rho$ and $V_2 : \rho$ be fixed. Let Z and S be $M_0\ \mathsf{nat}$ and $M_2\ \mathsf{nat}$. Let R be $\{(W_1, W_2) \mid \exists k \in \mathbb{N}, \mathsf{S}^k\ \mathsf{Z} \cong_{\mathsf{nat}} W_1 \wedge V_1^k\ V_2 \cong_\rho W_2\}$ in $\mathcal{R}(\mathsf{nat}, \rho)$ and η be $\alpha \mapsto (\mathsf{nat}, \rho, R)$.

We have $(\mathsf{Z}, V_2) \in \mathcal{V}[\![\alpha]\!]_\eta$ (**2**) since $R(\mathsf{Z}, V_2)$ (reduce both sides for $k = 0$). We also have $(\mathsf{S}, V_1) \in \mathcal{V}[\![\alpha\to\alpha]\!]_\eta$ (**3**), (which is a key to the proof). Indeed, assume (W_1, W_2) in $\mathcal{V}[\![\alpha]\!]_\eta$, *i.e.* R. There exists k such that $\mathsf{S}^k\ \mathsf{Z} \cong_{\mathsf{nat}} W_1$ and $V_1^k\ V_2 \cong_\rho W_2$. By congruence $\mathsf{S}\ W_1 \cong_{\mathsf{nat}} \mathsf{S}^{k+1}\ \mathsf{Z}$ and $V_1\ W_2 \cong_\rho V_1^{k+1}\ V_2$. Since $(\mathsf{S}^{k+1}\ \mathsf{Z}, V_1^{k+1}\ V_2)$ is in $\mathcal{E}[\![\alpha]\!]_\eta$, so is $(\mathsf{S}\ W_1, V_1\ W_2)$ by closure by observational equivalence.

By parametricity, we have $M \sim_{\mathsf{nat}} M$. Hence, $(M\ \mathsf{nat}\ \mathsf{S}\ \mathsf{Z}, M\ \rho\ V_1\ V_2) \in \mathcal{E}[\![\alpha]\!]_\eta$. Thus, the pair must reduce to a pair in R, there exists n such that $M\ \mathsf{nat}\ \mathsf{S}\ \mathsf{Z} \cong_{\mathsf{nat}} \mathsf{S}^n\ \mathsf{Z}$ and $M\ \rho\ V_1\ V_2 \cong_\rho V_1^n\ V_2$ We have shown,

$$\forall\rho,\ \forall V_1 \in \mathsf{Val}(\rho\to\rho),\ \forall V_2 \in \mathsf{Val}(\rho),\ \exists n \in \mathbb{N},\ M\ \mathsf{nat}\ \mathsf{S}\ \mathsf{Z} \cong_{\mathsf{nat}} \mathsf{S}^n\ \mathsf{Z} \wedge M\ \rho\ V_1\ V_2 \cong_\rho V_1^n\ V_2$$

Since, $M\ \mathsf{nat}\ \mathsf{S}\ \mathsf{Z}$ is independent of n, and all $\mathsf{S}^n\ \mathsf{Z}$ are in different observational equivalence classes (which is easy to prove by appplying, *e.g.*, to the successor function and primitive integer 0), n is actually independent of V_1 and V_2. Hence, we have:

$$\exists n \in \mathbb{N},\ \forall\rho,\ \forall V_1 \in \mathsf{Val}(\rho\to\rho),\ \forall V_2 \in \mathsf{Val}(\rho),\ M\ \mathsf{nat}\ \mathsf{S}\ \mathsf{Z} \cong_{\mathsf{nat}} \mathsf{S}^n\ \mathsf{Z} \wedge M\ \rho\ V_1\ V_2 \cong_\rho V_1^n\ V_2$$

which implies (1). □

Exercise 53 (sort)
Assume $\mathsf{sort} : \forall\alpha.\,(\alpha \to \alpha \to \mathsf{B}) \to \mathsf{list}\,\alpha \to \mathsf{list}\,\alpha$ *(1). Then for all* g *of ground type* $\rho_1 \to \rho_2$*, and all (comparison) functions* cmp_1 *of type* $\rho_1 \to \rho_1 \to \mathsf{B}$ *and* cmp_2 *of type* $\rho_2 \to \rho_2 \to \mathsf{B}$ *satisfying*

$$\forall V, W \in \mathsf{Val}(\rho_1),\ \ \mathsf{cmp}_2\,(g\,V)\,(g\,W) \ \cong\ \mathsf{cmp}_1\ V\,W \qquad (2)$$

we have, for all U *in* $\mathsf{Val}(\mathsf{list}\,\rho_1)$*,*

$$\mathsf{sort}\ \rho_2\ \mathsf{cmp}_2\ (\mathsf{map}\ \rho_1\ \rho_2\ g\ U) \ \cong\ \mathsf{map}\ \rho_1\ \rho_2\ g\ (\mathsf{sort}\ \rho_1\ \mathsf{cmp}_1\ U) \qquad (3)$$

Solution: Let ρ_1 and ρ_2 be fix and g be a function g satisfying (2). We show (3) as follows.

Let R in $\mathcal{R}(\rho_1, \rho_2)$ be the graph of the function g up to observational equivalence, *i.e.* composed of all pairs (W_1, W_2) such that $W_2 \cong f\ W_1$ and let η be $\alpha \mapsto (\rho_1, \rho_2, R)$.

We have $\mathsf{sort} \sim_\sigma \mathsf{sort}$ where σ is $\forall\alpha.\,(\alpha \to \alpha \to \mathsf{B}) \to \mathsf{list}\,\alpha \to \mathsf{list}\,\alpha$.

The hypothesis (2) implies $(\mathsf{cmp}_1, \mathsf{cmp}_2)$ in $\mathcal{V}[\![\alpha \to \alpha \to \mathsf{B}]\!]_\eta$. Indeed, consider (V_1, V_2) and (W_1, W_2) in $\mathcal{V}[\![\alpha_2]\!]_\eta$, *i.e.* in R. By definition of R, we have $V_2 \cong_{\eta_2} g\,V_1$ and $W_2 \cong_{\eta_2} g\,W_1$. By congruence and (2), we have

$$\mathsf{cmp}_2\ \ V_2\,W_2 \ \ \cong_{\mathsf{B}}\ \ \mathsf{cmp}_2\ \ (g\,V_1)\,(g\,W_1) \ \ \cong_{\mathsf{B}}\ \ \mathsf{cmp}_1\ \ V_1\,W_1$$

Hence, $\mathsf{cmp}_2\ \ V_2\,W_2 \cong_{\mathsf{B}} \mathsf{cmp}_1\ \ V_1\,W_1$ as expected.

Consider $\mathcal{V}[\![\mathsf{list}\ \alpha]\!]_\eta$. Informally, this is composed of all pairs (V_1, V_2) in $\mathsf{Val}(\mathsf{list}\ \rho_1) \times \mathsf{Val}(\mathsf{list}\,\rho_2)$ such that $V_2 \cong \mathsf{map}\ \rho_1\ \rho_2\ g\ V_1$. Indeed, this pointwise relates elements of the two lists. (A formal definition would require definition of logical relations for lists.)

Let U be in $\mathsf{Val}(\mathsf{list}\,\rho_1)$. We have $(U, \mathsf{map}\ \rho_1\ \rho_2\ g\ U)$ in $\mathcal{V}[\![\mathsf{list}\ \alpha]\!]_\eta$. Therefore, the pair

$$(\mathsf{sort}\ \rho_1\ \mathsf{cmp}_1\ \ U,\ \mathsf{sort}\ \rho_2\ \mathsf{cmp}_2\ \ (\mathsf{map}\ \rho_1\ \rho_2\ g\ U))$$

is in $\mathcal{V}[\![\mathsf{list}\ \alpha]\!]_\eta$, which actually means (3).

□

8.7 Extensions

8.7.1 Natural numbers

We have shown that all expressions of type nat behave as natural numbers. Hence, natural numbers are definable in System F.

Still, we can also provide a type nat of primitive natural numbers. Then we would define behavioral equivalence on nat as the relation in Val(nat) × Val(nat) by

$$M_1 \simeq_{\mathsf{nat}} M_2 \overset{\mathsf{def}}{\iff} \exists n : \mathsf{nat},\ M_1 \Downarrow n \wedge M_2 \Downarrow n$$

As for the logical equivalence, we defined

$$\mathcal{V}[\![\mathsf{nat}]\!] = \{(n, n) \mid n \in \mathsf{Val}(\mathsf{nat})\}$$

Notice that nat is another observable type. All properties are preserved.

8.7.2 Products

Encodable Given closed types τ_1 and τ_2, we defined

$$\begin{array}{rcl}\tau_1 \times \tau_2 & \triangleq & \forall\alpha.\, (\tau_1 \to \tau_2 \to \alpha) \to \alpha \\ (M_1, M_2) & \triangleq & \Lambda\alpha.\lambda x\!:\!\tau_1 \to \tau_2 \to \alpha.\, x\ M_1\ M_2 \\ M.i & \triangleq & M\ (\lambda x_1\!:\!\tau_1.\, \lambda x_2\!:\!\tau_2.\, x_i)\end{array}$$

Lemma 50
If $M : \tau_1 \times \tau_2$, then $M \cong_{\tau_1 \times \tau_2} (M_1, M_2)$ for some $M_1 : \tau_1$ and $M_2 : \tau_2$.
If $M : \tau_1 \times \tau_2$ and $M.1 \cong_{\tau_1} M_1$ and $M.2 \cong_{\tau_2} M_2$, then $M \cong_{\tau_1 \times \tau_2} (M_1, M_2)$

Primitive With primitive pairs, we define:

$$\mathcal{V}[\![\tau \times \sigma]\!]_\eta \triangleq \Big\{\big((V_1, W_1), (V_2, W_2)\big) \;\Big|\; (V_1, V_2) \in \mathcal{V}[\![\tau]\!]_\eta \wedge (W_1, W_2) \in \mathcal{V}[\![\sigma]\!]_\eta\Big\}$$

8.7.3 Sums

$$\mathcal{V}[\![\tau + \sigma]\!]_\eta \triangleq \{(\mathsf{inj}_1\ V_1, \mathsf{inj}_1\ V_2) \mid (V_1, V_2) \in \mathcal{V}[\![\tau]\!]_\eta\} \bigcup \{(\mathsf{inj}_2\ V_2, \mathsf{inj}_2\ V_2) \mid (V_1, V_2) \in \mathcal{V}[\![\sigma]\!]_\eta\}$$

8.7.4 Lists

We could extend the language with lists and define:

$$\mathcal{V}[\![\mathsf{list}\ \tau]\!]_\eta \triangleq \{([V_1^1; \ldots V_1^n], [V_2^1; \ldots V_2^n]) \mid n \in I\!N \wedge \forall k \in [1, n], (V_1^k, V_2^k) \in \mathcal{V}[\![\tau]\!]_\eta\}$$

Assume given a function g from ρ_1 to ρ_2. Let R in $\mathcal{R}(\rho_1, \rho_2)$ be the admissible relation composed of all pairs (W_1, W_2) such that $W_2 \cong g\ W_1$ and η be $\alpha \mapsto (\rho_1, \rho_2, R)$. Then $\mathcal{V}[\![\mathsf{list}\ \alpha]\!]_\eta$ is composed of all pairs (W_1, W_2) such that $W_2 \cong \mathsf{map}\ \rho_1\ \rho_2\ g\ W_1$ and $\mathcal{E}[\![\mathsf{list}\ \alpha]\!]_\eta$ is composed of all pairs (N_1, N_2) such that $N_2 \cong \mathsf{map}\ \rho_1\ \rho_2\ g\ N_1$.

8.7.5 Existential types

We define:

$$\begin{array}{rcl}\mathcal{V}[\![\exists\alpha.\,\tau]\!]_\eta & \triangleq & \Big\{(\mathsf{pack}\ V_1, \rho_1\ \mathsf{as}\ \exists\alpha.\,\tau, \mathsf{pack}\ V_2, \rho_2\ \mathsf{as}\ \exists\alpha.\,\tau) \mid V_1 \vdash \eta_1(\exists\alpha.\,\tau) \wedge V_2 \vdash \eta_2(\exists\alpha.\,\tau) \wedge \\ & & \quad \exists\rho_1, \rho_2, R \in \mathcal{R}(\rho_1, \rho_2),\ \ (V_1, V_2) \in \mathcal{E}[\![\tau]\!]_{\eta, \alpha \mapsto (\rho_1, \rho_2, R)}\Big\}\end{array}$$

Example 2 *Consider* $V_1 \triangleq (\mathsf{not}, \mathsf{tt})$*, and* $V_2 \triangleq (\mathsf{succ}, 0)$ *and* $\sigma \triangleq (\alpha \to \alpha) \times \alpha$*. Let* R *in* $\mathcal{R}(\mathsf{B}, \mathsf{nat})$ *be* $\{(\mathsf{tt}, 2n), (\mathsf{ff}, 2n+1) \mid n \in I\!N\}$ *and* η *be* $\alpha \mapsto (\mathsf{B}, \mathsf{nat}, R)$*.*

We have $(\mathsf{pack}\ V_1, \mathsf{B}\ \mathsf{as}\ \exists\alpha.\,\sigma,\ \mathsf{pack}\ V_2, \mathsf{nat}\ \mathsf{as}\ \exists\alpha.\,\sigma) \in \mathcal{V}[\![\exists\alpha.\,\sigma]\!]$*. To see this it suffices to show* $(V_1, V_2) \in \mathcal{V}[\![\sigma]\!]_\eta$*, that is* $\big((\mathsf{not}, \mathsf{tt}), (\mathsf{succ}, 0)\big) \in \mathcal{V}[\![(\alpha \to \alpha) \times \alpha]\!]_\eta$*. In turn, it suffices to show both* $(\mathsf{not}, \mathsf{succ}) \in \mathcal{V}[\![\alpha \to \alpha]\!]_\eta$ *and* $(\mathsf{tt}, 0) \in \mathcal{V}[\![\alpha]\!]_\eta$*. The latter holds by construction since* $(\mathsf{tt}, 0) \in R$*. To show the former, we assume* (W_1, W_2) *in* $\mathcal{V}[\![\alpha]\!]_\eta$*,* i.e. *in* R*. Hence, it must be either of the form*

- $(\mathsf{tt}, 2n)$*; and* $(\mathsf{not}\ W_1, \mathsf{succ}\ W_2)$ *reduces to* $(\mathsf{ff}, 2n+1)$*, or of the form*

- $(\mathsf{ff}, 2n+1)$ *and* $(\mathsf{not}\ W_1, \mathsf{succ}\ W_2)$*reduces to* $(\mathsf{tt}, 2n+2)$*.*

In both cases, $(\mathsf{not}\ W_1, \mathsf{succ}\ W_2)$ *reduces to a pair in* R*,* i.e. *in* $\mathcal{V}[\![\alpha]\!]_\eta$*, hence it is in* $\mathcal{E}[\![\alpha]\!]_\eta$*.*

Representation independence A client of an existential type $\exists\alpha.\,\tau$ should not see the difference between two implementations N_1 and N_2 of $\exists\alpha.\,\tau$ with witness types ρ_1 and ρ_2.

Assume that ρ_1 and ρ_2 are two closed representation types and R is in $\mathcal{R}(\rho_1, \rho_2)$. Let η be $\alpha \mapsto (\rho_1, \rho_2, R)$. Suppose that $N_1 : \tau[\alpha \mapsto \rho_1]$ and $N_2 : \tau[\alpha \mapsto \rho_2]$ are two equivalent implementations of the operations, *i.e.* $(N_1, N_2) \in \mathcal{E}[\![\tau]\!]_\eta$.

A client M has type $\forall\alpha.\,\tau \to \sigma$ with $\alpha \notin \mathsf{fv}(\sigma)$; it must use the argument parametrically, and the result is independent of the witness type. Indeed the client satisfies $(M, M) \in \mathcal{E}[\![\forall\alpha.\,\tau \to \sigma]\!]_\eta$ and therefore $(M\ \rho_1\ N_1, M\ \rho_2\ N_2)$ is in $\mathcal{E}[\![\sigma]\!]$ (as α is not free in σ), which implies $M\ \rho_1\ N_1 \cong_\sigma M\ \rho_2\ N_2$.

That is, the behavior with the implementation N_1 with representation type ρ_1 is indistinguishable from the behavior with implementation N_2 with representation type ρ_2.

8.7.6 Step-indexed logical relations

How do we deal with recursive types? Assume that we allow equi-recursive types.

$$\tau ::= \ldots \mid \mu\alpha.\tau$$

A naive definition would be

$$\mathcal{V}[\![\mu\alpha.\tau]\!]_\eta = \mathcal{V}[\![[\alpha \mapsto \mu\alpha.\tau]\tau]\!]_\eta$$

But this is ill-founded, because $[\alpha \mapsto \mu\alpha.\tau]\tau$ is usually larger than τ.

The solution is to use indexed-logical relations.

We use a sequence of decreasing relations indexed by integers (fuel), which is consumed during unfolding of recursive types.

Step-indexing (a taste) We define a sequence $\mathcal{V}_k[\![\tau]\!]_\eta$ indexed by natural numbers $n \in I\!N$ that relates values of type τ up to n reduction steps.

$$
\begin{array}{rcl}
\mathcal{V}_k[\![\mathsf{B}]\!]_\eta & = & \{(\mathsf{tt},\mathsf{tt}),(\mathsf{ff},\mathsf{ff})\} \\
\mathcal{V}_k[\![\tau \to \sigma]\!]_\eta & = & \{(V_1,V_2) \mid \forall j < k, \forall (W_1,W_2) \in \mathcal{V}_j[\![\tau]\!]_\eta, \\
& & \qquad (V_1\ W_1, V_2\ W_2) \in \mathcal{E}_j[\![\sigma]\!]_\eta\} \\
\mathcal{V}_k[\![\alpha]\!]_\eta & = & (\eta_R \alpha).k \\
\mathcal{V}_k[\![\forall \alpha.\, \tau]\!]_\eta & = & \big\{(V_1,V_2) \mid \forall \rho_1,\rho_2, R \in \mathcal{R}^k(\rho_1,\rho_2), \forall j < k, \\
& & \qquad (V_1\ \rho_1, V_2\ \rho_2) \in \mathcal{V}_j[\![\tau]\!]_{\eta,\alpha \mapsto (\rho_1,\rho_2,R)}\big\} \\
\mathcal{V}_k[\![\mu\alpha.\tau]\!]_\eta & = & \mathcal{V}_{k-1}[\![[\alpha \mapsto \mu\alpha.\tau]\tau]\!]_\eta \\
\mathcal{E}_k[\![\tau]\!]_\eta & = & \{(M_1,M_2) \mid \forall j < k, M_1 \Downarrow_j V_1 \\
& & \qquad \Longrightarrow \exists V_2, M_2 \Downarrow V_2 \wedge (V_1,V_2) \in \mathcal{V}_{k-j}[\![\tau]\!]_\eta\}
\end{array}
$$

By $\Downarrow_j$, we mean *reduces in j-steps*. $\mathcal{R}^j(\rho_1,\rho_2)$ is a sequence of decreasing relations between closed values of closed types ρ_1 and ρ_2 of length (at least) j.

Notice that the relation is asymmetric.

We define

$$
\Delta;\Gamma \vdash M_1 \precsim M_2 : \tau \overset{\mathsf{def}}{\iff} \begin{cases} \Delta;\Gamma \vdash M_1, M_2 : \tau. \\ \forall \eta \in \mathcal{R}^k_\Delta(\delta_1,\delta_2), \forall (\gamma_1,\gamma_2) \in \mathcal{G}_k[\![\Gamma]\!], \\ \qquad (\gamma_1(\delta_1(M_1)), \gamma_2(\delta_2(M_2)) \in \mathcal{E}_k[\![\tau]\!]_\eta \end{cases}
$$

and

$$
\Delta;\Gamma \vdash M_1 \sim M_2 : \tau \ \triangleq\ \bigwedge \begin{cases} \Delta;\Gamma \vdash M_1 \precsim M_2 : \tau \\ \Delta;\Gamma \vdash M_2 \precsim M_1 : \tau \end{cases}
$$

Notations and proofs get a bit involved.

Bibliography

▷ A tour of scala: Implicit parameters. Part of scala documentation.

▷ Martín Abadi and Luca Cardelli. A theory of primitive objects: Untyped and first-order systems. *Information and Computation*, 125(2):78–102, March 1996.

▷ Martín Abadi and Luca Cardelli. A theory of primitive objects: Second-order systems. *Science of Computer Programming*, 25(2–3):81–116, December 1995.

▷ Amal Ahmed and Matthias Blume. Typed closure conversion preserves observational equivalence. In *ACM International Conference on Functional Programming (ICFP)*, pages 157–168, September 2008.

▷ Lennart Augustsson. Implementing Haskell overloading. In *FPCA '93: Proceedings of the conference on Functional programming languages and computer architecture*, pages 65–73, New York, NY, USA, 1993. ACM. ISBN 0-89791-595-X.

▷ Nick Benton and Andrew Kennedy. Exceptional syntax journal of functional programming. *J. Funct. Program.*, 11(4):395–410, 2001.

▷ Jean-Philippe Bernardy, Patrik Jansson, and Koen Claessen. *Testing Polymorphic Properties*, pages 125–144. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-11957-6. doi: 10.1007/978-$\sigma_2$3-$\sigma_2$642-$\sigma_2$11957-$\sigma_2$6_8.

▷ Richard Bird and Lambert Meertens. Nested datatypes. In *International Conference on Mathematics of Program Construction (MPC)*, volume 1422 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 1998.

Nikolaj Skallerud Bjørner. Minimal typing derivations. In *In ACM SIGPLAN Workshop on ML and its Applications*, pages 120–126, 1994.

Daniel Bonniot. *Typage modulaire des multi-méthodes*. PhD thesis, École des Mines de Paris, November 2005.

▷ Daniel Bonniot. Type-checking multi-methods in ML (a modular approach). In *Workshop on Foundations of Object-Oriented Languages (FOOL)*, January 2002.

▷ Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticæ*, 33:309–338, 1998.

▷ Kim B. Bruce, Luca Cardelli, and Benjamin C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, November 1999.

Luca Cardelli. An implementation of f¡:. Technical report, DEC Systems Research Center, 1993.

Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science Series. Birkäuser, Boston, 1997.

▷ Henry Cejtin, Matthew Fluet, Suresh Jagannathan, and Stephen Weeks. The MLton compiler, 2007.

▷ Arthur Charguéraud and François Pottier. Functional translation of a calculus of capabilities. In *ACM International Conference on Functional Programming (ICFP)*, pages 213–224, September 2008.

▷ Juan Chen and David Tarditi. A simple typed intermediate language for object-oriented languages. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 38–49, January 2005.

▷ Adam Chlipala. A certified type-preserving compiler from lambda calculus to assembly language. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 54–65, June 2007.

▷ Karl Crary, Stephanie Weirich, and Greg Morrisett. Intensional polymorphism in type erasure semantics. *Journal of Functional Programming*, 12(6):567–600, November 2002.

Julien Crétin and Didier Rémy. Extending System F with Abstraction over Erasable Coercions. In *Proceedings of the 39th ACM Conference on Principles of Programming Languages*, January 2012.

▷ Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty, and Gabriele Keller. Modular type classes. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 63–70, New York, NY, USA, 2007. ACM. ISBN 1-59593-575-4.

Joshua Dunfield. Greedy bidirectional polymorphism. In *ML '09: Proceedings of the 2009 ACM SIGPLAN workshop on ML*, pages 15–26, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-509-3. doi: http://doi.acm.org/10.1145/1596627.1596631.

▷ Ken-etsu Fujita and Aleksy Schubert. Existential type systems with no types in terms. In *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, pages 112–126, 2009. doi: 10.1007/978-$\sigma_2$3-$\sigma_2$642-$\sigma_2$02273-$\sigma_2$9_10.

Jun Furuse. Extensional polymorphism by flow graph dispatching. In Ohori (2003), pages 376–393. ISBN 3-540-20536-5.

▷ Jun Furuse. Extensional polymorphism by flow graph dispatching. In *Asian Symposium on Programming Languages and Systems (APLAS)*, volume 2895 of *Lecture Notes in Computer Science*. Springer, November 2003b.

▷ Jacques Garrigue. Relaxing the value restriction. In *Functional and Logic Programming*, volume 2998 of *Lecture Notes in Computer Science*, pages 196–213. Springer, April 2004.

Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, June 1972.

▷ Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1990.

▷ Dan Grossman. Quantified types in an imperative language. *ACM Transactions on Programming Languages and Systems*, 28(3):429–475, May 2006.

▷ Bob Harper and Mark Lillibridge. ML with callcc is unsound. Message to the TYPES mailing list, July 1991.

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.

Robert Harper and Benjamin C. Pierce. Design considerations for ML-style module systems. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 8, pages 293–345. MIT Press, 2005.

▷ Fritz Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253–289, April 1993.

▷ J. Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

▷ Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of Haskell: being lazy with class. In *ACM SIGPLAN Conference on History of Programming Languages*, June 2007.

Gérard Huet. *Résolution d'équations dans des langages d'ordre* 1*,* 2*, …,* ω*.* PhD thesis, Université Paris 7, September 1976.

▷ John Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

▷ Mark P. Jones. Simplifying and improving qualified types. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 160–169, New York, NY, USA, 1995a. ACM. ISBN 0-89791-719-7.

Mark P. Jones. Typing Haskell in Haskell. In *In Haskell Workshop*, 1999a.

Mark P. Jones. *Qualified types: theory and practice*. Cambridge University Press, New York, NY, USA, 1995b. ISBN 0-521-47253-9.

▷ Mark P. Jones. Typing Haskell in Haskell. In *Haskell workshop*, October 1999b.

▷ Simon Peyton Jones, Mark Jones, and Erik Meijer. Type classes: an exploration of the design space. In *Haskell workshop*, 1997.

▷ Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *Journal of Functional Programming*, 17(01):1, 2006.

Stefan Kaes. Type inference in the presence of overloading, subtyping and recursive types. In *LFP '92: Proceedings of the 1992 ACM conference on LISP and functional programming*, pages 193–204, New York, NY, USA, 1992. ACM. ISBN 0-89791-481-3. doi: http://doi.acm.org/10.1145/141471.141540.

▷ Assaf J. Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. ML typability is DEXPTIME-complete. In *Colloquium on Trees in Algebra and Programming*, volume 431 of *Lecture Notes in Computer Science*, pages 206–220. Springer, May 1990.

▷ Peter J. Landin. Correspondence between ALGOL 60 and Church's lambda-notation: part I. *Communications of the ACM*, 8(2):89–101, 1965.

▷ Konstantin Läufer and Martin Odersky. Polymorphic type inference and abstract data types. *ACM Transactions on Programming Languages and Systems*, 16(5):1411–1430, September 1994.

▷ Didier Le Botlan and Didier Rémy. Recasting MLF. *Information and Computation*, 207(6): 726–785, 2009. ISSN 0890-5401. doi: 10.1016/j.ic.2008.12.006.

▷ Xavier Leroy. *Typage polymorphe d'un langage algorithmique.* PhD thesis, Université Paris 7, June 1992.

▷ Xavier Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 42–54, January 2006.

▷ Xavier Leroy and François Pessaux. Type-based analysis of uncaught exceptions. *ACM Trans. Program. Lang. Syst.*, 22(2):340–377, 2000. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/349214.349230.

▷ John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 47–57, January 1988.

▷ Harry G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 382–401, 1990.

▷ David McAllester. A logical algorithm for ML type inference. In *Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 436–451. Springer, June 2003.

Todd D. Millstein and Craig Chambers. Modular statically typed multimethods. In *ECOOP '99: Proceedings of the 13th European Conference on Object-Oriented Programming*, pages 279–303, London, UK, 1999. Springer-Verlag. ISBN 3-540-66156-5.

▷ Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.

▷ Yasuhiko Minamide, Greg Morrisett, and Robert Harper. Typed closure conversion. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 271–283, January 1996.

▷ John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76(2–3):211–249, 1988.

▷ John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, 1988.

▷ Benoît Montagu and Didier Rémy. Modeling abstract types in modules with open existential types. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 63–74, January 2009.

J. Garrett Morris and Mark P. Jones. Instance chains: type class programming without overlapping instances. In *ICFP '10: Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 375–386, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-794-3. doi: http://doi.acm.org/10.1145/1863543.1863596.

▷ Greg Morrisett and Robert Harper. Typed closure conversion for recursively-defined functions (extended abstract). In *International Workshop on Higher Order Operational Techniques in Semantics (HOOTS)*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1998.

▷ Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):528–569, May 1999.

▷ Alan Mycroft. Polymorphic type schemes and recursive definitions. In *International Symposium on Programming*, volume 167 of *Lecture Notes in Computer Science*, pages 217–228. Springer, April 1984.

▷ Matthias Neubauer, Peter Thiemann, Martin Gasbichler, and Michael Sperber. Functional logic overloading. pages 233–244, 2002. doi: http://doi.acm.org/10.1145/565816.503294.

▷ Martin Odersky, Philip Wadler, and Martin Wehr. A second look at overloading. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 135–146, New York, NY, USA, 1995. ACM. ISBN 0-89791-719-7.

▷ Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55, 1999.

▷ Martin Odersky, Matthias Zenger, and Christoph Zenger. Colored local type inference. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 41–53, 2001.

Atsushi Ohori, editor. *Programming Languages and Systems, First Asian Symposium, APLAS 2003, Beijing, China, November 27-29, 2003, Proceedings*, volume 2895 of *Lecture Notes in Computer Science*, 2003. Springer. ISBN 3-540-20536-5.

▷ Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1999.

▷ Bruno C.d.S. Oliveira, Tom Schrijvers, Wontae Choi, Wonchan Lee, and Kwangkeun Yi. The implicit calculus: a new foundation for generic programming. In *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, PLDI '12, pages 35–44, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1205-9. doi: 10.1145/2254064.2254070.

▷ Simon Peyton Jones. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. Online lecture notes, January 2009.

▷ Simon Peyton Jones and Mark Shields. Lexically-scoped type variables. Manuscript, April 2004.

▷ Simon Peyton Jones and Philip Wadler. Imperative functional programming. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 71–84, January 1993.

Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *LFP '88: Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 153–163, New York, NY, USA, 1988. ACM. ISBN 0-89791-273-X. doi: http://doi.acm.org/10.1145/62678.62697.

▷ Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

▷ Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Transactions on Programming Languages and Systems*, 22(1):1–44, January 2000.

▷ Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.

▷ François Pottier. Notes du cours de DEA "Typage et Programmation", December 2002.

François Pottier. A typed store-passing translation for general references. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages (POPL'11)*, Austin, Texas, January 2011. Supplementary material.

François Pottier. Syntactic soundness proof of a type-and-capability system with hidden state. *Journal of Functional Programming*, 23(1):38–144, January 2013.

François Pottier. Hindley-Milner elaboration in applicative style. In *Proceedings of the 2014 ACM SIGPLAN International Conference on Functional Programming (ICFP'14)*, September 2014.

▷ François Pottier and Nadji Gauthier. Polymorphic typed defunctionalization and concretization. *Higher-Order and Symbolic Computation*, 19:125–162, March 2006.

François Pottier and Jonathan Protzenko. Programming with permissions in Mezzo. Submitted for publication, October 2012.

François Pottier and Jonathan Protzenko. Programming with permissions in Mezzo. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Functional Programming (ICFP'13)*, pages 173–184, September 2013.

▷ François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.

▷ François Pottier and Didier Rémy. The essence of ML type inference. Draft of an extended version. Unpublished, September 2003.

▷ Didier Rémy. Simple, partial type-inference for System F based on type-containment. In *Proceedings of the tenth International Conference on Functional Programming*, September 2005.

▷ Didier Rémy. Programming objects with ML-ART: An extension to ML with abstract and record types. In *International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 321–346. Springer, April 1994a.

▷ Didier Rémy. Type inference for records in a natural extension of ML. In Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects Of Object-Oriented Programming: Types, Semantics and Language Design*. MIT Press, 1994b.

▷ Didier Rémy and Jérôme Vouillon. Objective ML: An effective object-oriented extension to ML. *Theory and Practice of Object Systems*, 4(1):27–50, 1998.

Didier Rémy and Boris Yakobowski. Efficient Type Inference for the MLF language: a graphical and constraints-based approach. In *The 13th ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, pages 63–74, Victoria, BC, Canada, September 2008. doi: http://doi.acm.org/10.1145/1411203.1411216.

▷ John C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, April 1974.

▷ John C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83*, pages 513–523. Elsevier Science, 1983.

▷ John C. Reynolds. Three approaches to type structure. In *International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 185 of *Lecture Notes in Computer Science*, pages 97–138. Springer, March 1985.

François Rouaix. Safe run-time overloading. In *Proceedings of the 17th ACM Conference on Principles of Programming Languages*, pages 355–366, 1990. doi: http://doi.acm.org/10.1145/96709.96746.

▷ Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. Cochis: Deterministic and coherent implicits. Technical report, KU Leuven, May 2017.

▷ Christian Skalka and François Pottier. Syntactic type soundness for HM(X). In *Workshop on Types in Programming (TIP)*, volume 75 of *Electronic Notes in Theoretical Computer Science*, July 2002.

Lau Skorstengaard. An Introduction to Logical Relations. *arXiv e-prints*, art. arXiv:1907.11133, July 2019.

Geoffrey S. Smith. Principal type schemes for functional programs with overloading and subtyping. In *Science of Computer Programming*, 1994.

Morten Heine Sørensen and Pawel Urzyczyn. *Studies in Logic and the Foundations of Mathematics*, chapter Lectures on the Curry-Howard Isomorphism. Elselvir Science Inc, 2006.

▷ Matthieu Sozeau and Nicolas Oury. First-Class Type Classes. In Sofiène Tahar, Otmame Ait-Mohamed, and César Muñoz, editors, *TPHOLs 2008: Theorem Proving in Higher Order Logics, 21th International Conference*, Lecture Notes in Computer Science. Springer, August 2008.

▷ Paul A. Steckler and Mitchell Wand. Lightweight closure conversion. *ACM Transactions on Programming Languages and Systems*, 19(1):48–86, 1997.

▷ Christopher Strachey. Fundamental concepts in programming languages. *Higher-Order and Symbolic Computation*, 13(1–2):11–49, April 2000.

▷ Peter J. Stuckey and Martin Sulzmann. A theory of overloading. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 167–178, New York, NY, USA, 2002. ACM. ISBN 1-58113-487-8.

▷ W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):pp. 198–212, 1967. ISSN 00224812.

▷ Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Information and Computation*, 11(2):245–296, 1994.

Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975.

▷ Jerzy Tiuryn and Pawel Urzyczyn. The subtyping problem for second-order types is undecidable. *Information and Computation*, 179(1):1–18, 2002.

▷ Mads Tofte, Lars Birkedal, Martin Elsman, and Niels Hallenberg. A retrospective on region-based memory management. *Higher-Order and Symbolic Computation*, 17(3):245–265, September 2004.

▷ Andrew Tolmach and Dino P. Oliva. From ML to Ada: Strongly-typed language interoperability via source translation. *Journal of Functional Programming*, 8(4):367–412, July 1998.

▷ Philip Wadler. Theorems for free! In *Conference on Functional Programming Languages and Computer Architecture (FPCA)*, pages 347–359, September 1989.

▷ Philip Wadler. The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1–3):201–226, May 2007.

▷ Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad-hoc. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 60–76, January 1989.

Mitchell Wand. Corrigendum: Complete type inference for simple objects. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, 1988.

▷ J. B. Wells. The essence of principal typings. In *International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 913–925. Springer, 2002.

▷ J. B. Wells. The undecidability of Mitchell's subtyping relation. Technical Report 95-019, Computer Science Department, Boston University, December 1995.

▷ J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.

▷ Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8 (4):343–356, December 1995.

▷ Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, November 1994.