## Contents

1 Introduction .................................................. 7
   1.1 Overview of the course ................................... 7
   1.2 Requirements ............................................. 9
   1.3 About Functional Programming ......................... 9
   1.4 About Types .............................................. 9
   1.5 Acknowledgment .......................................... 11

2 The untyped $\lambda$-calculus ................................. 13
   2.1 Syntax .................................................. 13
   2.2 Semantics ............................................... 15
      2.2.1 Strong v.s. weak reduction strategies ............. 15
      2.2.2 Call-by-value semantics ............................ 16
   2.3 Answers to exercises .................................... 18

3 Simply-typed lambda-calculus ................................. 21
   3.1 Syntax .................................................. 21
   3.2 Dynamic semantics ...................................... 21
   3.3 Type system ............................................. 22
   3.4 Type soundness .......................................... 25
      3.4.1 Proof of subject reduction ......................... 26
      3.4.2 Proof of progress .................................. 28
   3.5 Simple extensions ....................................... 30
      3.5.1 Unit ................................................ 30
      3.5.2 Boolean ............................................ 30
      3.5.3 Pairs .............................................. 31
      3.5.4 Sums .............................................. 32
      3.5.5 Modularity of extensions ......................... 32
      3.5.6 Recursive functions ............................... 33
      3.5.7 A derived construct: let-bindings ............... 33
   3.6 Exceptions .............................................. 35
      3.6.1 Semantics .......................................... 35
## 5 Type reconstruction

5.1 Introduction .............................................. 91
5.2 Type inference for simply-typed $\lambda$-calculus ......... 92
   5.2.1 Constraints ......................................... 93
   5.2.2 A detailed example .................................. 94
   5.2.3 Soundness and completeness of type inference .... 96
   5.2.4 Constraint solving ................................. 96
5.3 Type inference for ML ................................. 98

---

## 6 Fomega: higher-kinds and higher-order types

6.1 Introduction ............................................. 109
6.2 From System F to System $F^\omega$ ......................... 110
   6.2.1 Properties ........................................ 111
6.3 Expressiveness .......................................... 113
   6.3.1 Distrib pair in System $F^\omega$ ................. 113
   6.3.2 Abstracting over type operators .................. 113
   6.3.3 Encoding of existential types ...................... 114
   6.3.4 Church encoding of non-regular ADT .............. 115
   6.3.5 Encoding GADT—with explicit coercions .......... 117
6.4 Beyond $F^\omega$ ....................................... 118

---

## 7 Logical Relations

7.1 Introduction ............................................. 123
7.1.1 Parametricity ...................................... 123
7.2 Normalization of simply-typed $\lambda$-calculus .......... 125
7.3 Observational equivalence ............................... 128
7.4 Logical rel in simply-typed $\lambda$-calculus ............ 129
   7.4.1 Logical equivalence for closed terms ............ 129
   7.4.2 Logical equivalence for open terms .............. 131
7.5 Logical rel. in $F$ ..................................... 134
   7.5.1 Logical equivalence for closed terms ............ 135
7.6 Applications ........................................... 140
7.7 Extensions ............................................. 142
   7.7.1 Natural numbers .................................... 142
   7.7.2 Products ........................................... 143
   7.7.3 Sums ............................................... 143
   7.7.4 Lists ............................................... 143
   7.7.5 Existential types .................................. 143
   7.7.6 Step-indexed logical relations ...................... 144

---

## 5.3.3 Mutually recursive functions .......................... 105
5.3.1 Milner's Algorithm \( \mathcal{T} \) ............................................. 98
5.3.2 Constraints ......................................................... 99
5.3.3 Constraint solving by example .................................. 103
5.3.4 Type reconstruction .................................................. 106
5.4 Type annotations ....................................................... 109
5.4.1 Explicit binding of type variables ......................... 110
5.4.2 Polymorphic recursion .............................................. 113
5.4.3 mixed-prefix ......................................................... 114
5.5 Equi- and iso-recursive types ....................................... 115
5.5.1 Equi-recursive types .............................................. 115
5.5.2 Iso-recursive types ................................................ 117
5.5.3 Algebraic data types .............................................. 118
5.6 HM(X) .............................................................. 119
5.7 Type reconstruction in System F ..................................... 121
5.7.1 Type inference based on Second-order unification .......... 121
5.7.2 Bidirectional type inference ..................................... 122
5.7.3 Partial type inference in MLF ................................. 124
5.8 Proofs and Solution to Exercises ................................. 124

8 Overloading ................................................................. 159
8.1 An overview ........................................................... 159
8.1.1 Why use overloading? ............................................. 159
8.1.2 Different forms of overloading ................................. 160
8.1.3 Static overloading ............................................... 161
8.1.4 Dynamic resolution with a type passing semantics ........ 161
8.1.5 Dynamic overloading with a type erasing semantics ...... 162
8.2 Mini Haskell ............................................................... 162
8.2.1 Examples in MH ................................................... 163
8.2.2 The definition of Mini Haskell .............................. 164
8.2.3 Semantics of Mini Haskell ....................................... 165
8.2.4 Elaboration of expressions ..................................... 167
8.2.5 Summary of the elaboration .................................. 168
8.2.6 Elaboration of dictionaries .................................... 170
8.3 Implicitly-typed terms ................................................ 172
8.4 Variations .............................................................. 177
8.5 Omitted proofs and answers to exercises ....................... 181
Chapter 7

Logical Relations

7.1 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.

7.1.1 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 $\forall \alpha. \alpha \to \text{int}$? 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 \( \alpha \) and must return a value of type \( \alpha \) — but cannot examine \( \alpha \). Thus, it must eventually return \( v \), \textit{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 \( \beta \), but there is no way \( v \) and \( w \) can interact. So that function must also return \( v \), \textit{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 \( \alpha \) and must return a value of type \( \alpha \). Again, the arguments cannot interact, as we do not have any operation available of type \( \alpha \). 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.

\section*{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 \texttt{whoami} of type \( \forall \alpha. \text{list} \alpha \rightarrow \text{list} \alpha \)?

- 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:

\[
\forall f, x, \ \texttt{whoami} (\text{map} f x) = \text{map} f (\texttt{whoami} x)
\]

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

\[
\texttt{sort} : \ \forall \alpha. (\alpha \rightarrow \alpha \rightarrow \text{bool}) \rightarrow \text{list} \alpha \rightarrow \text{list} \alpha
\]

we can actually deduce that if \( f \) is order-preserving, then sorting commutes with \texttt{map}. Formally:

\[
(\forall x, y, \ \text{cmp}_2 (f x) (f y) = \text{cmp}_1 x y) \quad \implies \quad \forall \ell, \ \texttt{sort} \ \text{cmp}_2 (\text{map} f \ell) = \text{map} f (\texttt{sort} \ \text{cmp}_1 \ell)
\]

Such properties can be used to significantly reduce testing: in particular, if \texttt{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 \texttt{sort}, \textit{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.

7.2 Normalization of simply-typed $\lambda$-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 $\lambda$-calculus is also lifted at the level of types in richer type systems such as System $F^\omega$ 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 $\lambda$-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 $\lambda$-calculus is simple enough and interesting to be presented here. Notice that our presentation of simply-typed $\lambda$-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 $E[\tau]$ of terminating closed terms of type $\tau$, and then show that any term of type $\tau$ is actually in $E[\tau]$, 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 $E[\tau_1 \rightarrow \tau_2]$ not only terminate but also terminate when applied to any term in $E[\tau_1]$.

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

**Definition 2** We recursively define $V[\tau]$ and $E[\tau]$, subsets of closed values and closed ex-
expressions of (ground) type $\tau$ by induction on types as follows:

\[
\begin{align*}
V[B] & \triangleq \{ \text{tt, ff} \} \\
V[\tau_1 \rightarrow \tau_2] & \triangleq \{ \lambda x : \tau_1. M \mid \forall V \in V[\tau_1], (\lambda x : \tau_1. M) V \in E[\tau_2] \} \\
E[\tau] & \triangleq \{ M \mid \exists V \in V[\tau], M \downarrow V \}
\end{align*}
\]

We write $M \downarrow V$ as a shorthand for $M \rightarrow^* V$. The goal is to show that any closed expression of type $\tau$ is in $E[\tau]$.

**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 $E[\tau]$ 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**

- $V[\tau] \subseteq E[\tau]$ by definition.

- $E[\tau]$ is closed by inverse reduction—by definition, *i.e.* If $M \rightarrow N$ and $N \in E[\tau]$ then $M \in E[\tau]$.

- $E[\tau]$ is closed by reduction. By confluence (since the reduction is deterministic), if $M \downarrow N$ and $M \rightarrow V$, then $N \downarrow V$.

- For any term in $E[\tau]$, the reduction of $M$ terminates—by definition of $E[\tau]$.

**Normalization** Therefore, it just remains to show that any term of type $\tau$ is in $E[\tau]$:

**Lemma 30** If $\emptyset \vdash M : \tau$, then $M \in E[\tau]$.

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 $\tau$ as closed values in $V[\tau]$.

We define a semantic judgment for open terms $\Gamma \models M : \tau$ so that $\emptyset \vdash M : \tau$ implies $\Gamma \models M : \tau$ and $\emptyset \models M : \tau$ means $M \in E[\tau]$.

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

$$
\Gamma \models M : \tau \iff \forall \gamma \in G[\Gamma], \gamma(M) \in E[\tau]
$$

Theorem 15 (fundamental lemma) If $\Gamma \vdash M : \tau$ then $\Gamma \models M : \tau$.

Corollary 31 (termination of well-typed terms) If $\emptyset \vdash M : \tau$ then $M \in \mathcal{E}[\tau]$.

That is, closed well-typed terms of type $\tau$ evaluates to values of type $\tau$.

---

**Proof:** By induction on the typing derivation

**Routine cases**

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

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, $\Gamma \vdash M_1 : \tau_2 \rightarrow \tau$ 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 $\Gamma \models M_1 : \tau_2 \rightarrow \tau$ and $\Gamma \models M_2 : \tau_2$. Thus $\gamma M_1 \in \mathcal{E}[\tau_2 \rightarrow \tau]$ (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 $\Gamma \vdash \text{if } M \text{ then } M_1 \text{ else } M_2 : \tau$: By cases on the evaluation of $\gamma M$ for $\gamma$ in $\mathcal{G}[\Gamma]$.

**The interesting case**

Case $\Gamma \vdash \lambda x : \tau_1. M : \tau_1 \rightarrow \tau$:

Assume $\gamma \in \mathcal{G}[\Gamma]$. We must show that $\gamma(\lambda x : \tau_1. M) \in \mathcal{E}[\tau_1 \rightarrow \tau]$ (1) That is, $\lambda x : \tau_1. \gamma M \in \mathcal{V}[\tau_1 \rightarrow \tau]$ (we may assume $x \notin \text{dom}(\gamma)$ w.l.o.g.) Let $V \in \mathcal{V}[\tau_1]$, it suffices to show $(\lambda x : \tau_1. \gamma M) V \Downarrow (\gamma M)[x \mapsto V] = \gamma' M$ where $\gamma'$ is $\gamma[x \mapsto V] \in \mathcal{G}[\Gamma, x : \tau_1]$ (3). Since $\Gamma, x : \tau_1 \vdash M : \tau$, we have $\Gamma, x : \tau_1 
\models M : \tau$ 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:

$$\mathcal{E}[\tau] = \{ M : \tau \mid \forall N, M \Downarrow N \implies (N \in \mathcal{V}[\tau] \lor \exists N', N \rightarrow N') \}$$

**Exercise 41** Show type soundness with this semantics.
7.3 Observational equivalence in simply-typed $\lambda$-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 $\mathbb{B}$ (the only one in our setting). We thus define:

**Definition 3** (Behavioral equivalence) Two closed programs $M$ and $N$ of the same base type are behaviorally equivalent and we write $\equiv$ 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 $\Gamma \vdash M : \tau \land \Gamma \vdash N : \tau$.

**Definition 4** (observational equivalence) Assume $\Gamma \vdash M, N : \tau$. We say that $M$ and $N$ are observationally equivalent and we write $\equiv$ if there are behaviorally equivalent when placed in any closing context at some base type. That is,

$$\equiv \triangleq \forall C : (\Gamma \triangleright \tau) \rightsquigarrow (\emptyset \triangleright \mathbb{B}), C[M] \equiv C[N]$$

**Definition 5** (Typing of context)

$$C : (\Gamma \triangleright \tau) \rightsquigarrow (\Delta \triangleright \sigma) \iff (\forall M, \Gamma \vdash M : \tau \Longrightarrow \Delta \vdash C[M] : \sigma)$$
7.4. Logical relations in simply-typed λ-calculus

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 $M \equiv N$ as an abbreviation for $\emptyset \vdash M \equiv N : \tau$

**Lemma 32** Observational equivalence is the coarsest consistent congruence, where:

- a relation $\equiv$ is consistent if $\emptyset \vdash M \equiv N : \mathcal{B}$ implies $M \simeq N$.
- a relation $\equiv$ is a congruence if it is an equivalence and is closed by context, i.e. $\Gamma \vdash M \equiv N : \tau \land \mathcal{C} : (\Gamma \triangleright \tau) \not\rightsquigarrow (\Delta \triangleright \sigma) \implies \Delta \vdash \mathcal{C}[M] \equiv \mathcal{C}[N] : \sigma$

**Proof:**

*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 $\Gamma \vdash M \simeq N : \tau$ holds (1).

Let $\mathcal{C} : (\Gamma \triangleright \tau) \not\rightsquigarrow (\emptyset \triangleright \mathcal{B})$ (2). We must show that $\mathcal{C}[M] \simeq \mathcal{C}[N]$.

This follows by consistency applied to $\Gamma \vdash \mathcal{C}[M] \equiv \mathcal{C}[N] : \mathcal{B}$ 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.

7.4 Logical relations in simply-typed λ-calculus

7.4.1 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 $\text{Val}(\tau) \times \text{Val}(\tau)$ and $\mathcal{E}[\tau]$, the closure of $\mathcal{V}[\tau]$ by inverse reduction is a subset of $\text{Expressions}_\tau \times \text{Exp}(\tau)$. 
Definition 6 (Logical equivalence for closed terms) We recursively define two relations $\mathcal{V}[\tau]$ and $\mathcal{E}[\tau]$ between values of type $\tau$ and expressions of type $\tau$ by

$$\mathcal{V}[\mathcal{B}] \triangleq \{(tt, tt), (ff, ff)\}$$

$$\mathcal{V}[\tau \to \sigma] \triangleq \{(V_1, V_2) \mid V_1, V_2 \vdash \tau \rightarrow \sigma \land \forall (W_1, W_2) \in \mathcal{V}[\tau], (V_1 \ W_1, V_2 \ W_2) \in \mathcal{E}[\sigma]\}$$

$$\mathcal{E}[\tau] \triangleq \{(M_1, M_2) \mid M_1, M_2 : \tau \land \exists (V_1, V_2) \in \mathcal{V}[\tau], M_1 \downarrow V_1 \land M_2 \downarrow V_2\}$$

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

$$\mathcal{E}[\tau] \triangleq \{(M_1, M_2) \mid M_1, M_2 : \tau \land \forall V_1, M_1 \downarrow V_1 \implies \exists V_2, M_2 \downarrow V_2 \land (V_1, V_2) \in \mathcal{V}[\tau]\}$$

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

$$\mathcal{V}[\tau \rightarrow \sigma] \triangleq \{((\lambda x : \tau. M_1), (\lambda x : \tau. M_2)) \mid (\lambda x : \tau. M_1), (\lambda x : \tau. M_2) \vdash \tau \rightarrow \sigma \land \forall (W_1, W_2) \in \mathcal{V}[\tau], ((\lambda x : \tau. M_1) \ W_1, (\lambda x : \tau. M_2) \ W_2) \in \mathcal{E}[\sigma]\}$$

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 $(N_1, N_2) \in \mathcal{E}[\tau]$.

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

Corollaries

- If $(M_1, M_2) \in \mathcal{E}[\tau \rightarrow \sigma]$ and $(N_1, N_2) \in \mathcal{E}[\tau]$, then $(M_1 \ N_1, M_2 \ N_2) \in \mathcal{E}[\sigma]$. 
• To prove \((M_1, M_2) \in \mathcal{E}[\tau \rightarrow \sigma]\), 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** \((\sim_B) \subseteq (\approx)\).
Immediate, by definition of \(\mathcal{E}[B]\) and \(\mathcal{V}[B] \subseteq (\approx)\).

**Lemma 33 (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 \(\tau\). For \(\sim_\tau\), the proof is immediate by transitivity and symmetry of \(\approx_\tau\). For \(\approx_\tau\), it goes as follows:

**Case** \(\tau\) is \(B\): the result is immediate by symmetry and transitivity of behavioral equivalence.

**Case** \(\tau\) is \(\tau \rightarrow \sigma\):

By **IH**, symmetry and transitivity hold at types \(\tau\) and \(\sigma\). For symmetry, assume \(V_1 \approx_\tau V_2\) (1), we must show \(V_2 \approx_\tau V_1\). Assume \(W_1 \approx_\tau W_2\). We must show \(V_2 W_1 \approx_\sigma V_1 W_2\) (2). We have \(W_2 \approx_\tau W_1\) by symmetry at type \(\tau\). By (1), we have \(V_2 W_2 \approx_\sigma V_1 W_1\) and (2) follows by symmetry of \(\sim\) at type \(\sigma\).

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 \(W_1 \approx_\tau W_3\) and show \(V_1 W_1 \approx_\sigma V_3 W_3\) (5). By (3), we have \(V_1 W_1 \approx_\tau V_2 W_2\) (6). By symmetry and transitivity of \(\approx_\tau\), we get \(W_3 \approx_\tau W_3\) (7). By (4), we have \(V_2 W_3 \approx_\sigma V_3 W_3\) (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 \(W_1\).

**7.4.2 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 \(\tau\).

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

\[
\begin{align*}
\mathcal{G}[\emptyset] & \triangleq \{ \emptyset \} \\
\mathcal{G}[\Gamma, x : \tau] & \triangleq \{ \gamma, x \mapsto (V_1, V_2) \mid \gamma \in \mathcal{G}[\Gamma] \land (V_1, V_2) \in \mathcal{V}[\tau] \}
\end{align*}
\]

Given a bisubstitution \(\gamma\), we write \(\gamma_i\) for the substitution that maps \(x\) to \(V_i\) whenever \(\gamma\) maps \(x\) to \((V_1, V_2)\).
Definition 7 (Logical equivalence for open terms)

\[ \Gamma \vdash M_1 \sim M_2 : \tau \quad \text{def} \quad \left\{ \begin{array}{l} \Gamma \vdash M_1, M_2 : \tau \\
\forall \gamma \in \mathcal{G}[\Gamma], (\gamma_1 M_1, \gamma_2 M_2) \in \mathcal{E}[\tau] \end{array} \right\} \]

We also write \( \vdash M_1 \sim M_2 : \tau \) or \( M_1 \sim M_2 \) for \( \varnothing \vdash M_1 \sim M_2 : \tau \).

Lemma 34 Open logical equivalence is symmetric and transitive.

\[ \text{Proof: The Proof is immediate by the definition and the symmetry and transitivity of closed logical equivalence.} \]

Theorem 16 (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

\[
\begin{align*}
\text{C-True} & \quad \text{C-False} \\
\Gamma \vdash \mathbf{true} \sim \mathbf{true} : \mathsf{bool} & \quad \Gamma \vdash \mathbf{false} \sim \mathbf{false} : \mathsf{bool} \\
\text{C-If} & \quad \text{C-Abs} \\
\Gamma \vdash M_1 \sim M_2 : \mathsf{B} & \quad \Gamma \vdash N_1 \sim N_2 : \tau & \quad \Gamma \vdash N'_1 \sim N'_2 : \tau \\
\Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_2 : \tau & \quad \Gamma \vdash \lambda x : \tau. M_1 \sim \lambda x : \tau. M_2 : \tau \rightarrow \sigma \\
\text{C-App} & \quad \text{C-App} \\
\Gamma \vdash M_1 \sim M_2 : \tau \rightarrow \sigma & \quad \Gamma \vdash N_1 \sim N_2 : \tau \\
\Gamma \vdash M_1 N_1 \sim M_2 N_2 : \sigma &
\end{align*}
\]

\[ \text{Proof: Each case can be shown independently.} \]

\textbf{Rule} \text{C-Abs} Assume \( \Gamma, x : \tau \vdash M_1 \sim M_2 : \sigma \) (1). We show \( \Gamma \vdash \lambda x : \tau. M_1 \sim \lambda x : \tau. M_2 : \tau \rightarrow \sigma \).

Let \( \gamma \) be in \( \mathcal{G}[\Gamma] \). We show \( (\gamma_1 (\lambda x : \tau. M_1), \gamma_2 (\lambda x : \tau. M_2)) \in \mathcal{V}[\tau \rightarrow \sigma] \). 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 \( \gamma' \) be \( \gamma, x \mapsto (V_1, V_2) \). We have \( \gamma' \) 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 \( \gamma_i (\lambda x : \tau. M_i) V_i \downarrow \gamma'_i M_i \).

\textbf{Rule} \text{C-App} and \text{C-Appp} By induction hypothesis and the fact that substitution distributes over application.

We must show \( \Gamma \vdash M_1 N_1 \sim M_2 N_2 : \sigma \) (3). Let \( \gamma \) be in \( \mathcal{G}[\Gamma] \). From the premises \( \Gamma \vdash M_1 \sim M_2 : \tau \rightarrow \sigma \) and \( \Gamma \vdash N_1 \sim N_2 : \tau \), we have \( (\gamma_1 M_1, \gamma_2 M_2) \) in \( \mathcal{E}[\tau \rightarrow \sigma] \) 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 $\Gamma \vdash \text{if } M_1 \text{ then } N_1 \text{ else } N_1' \sim \text{if } M_2 \text{ then } N_2 \text{ else } N_2' : \tau$. Assume $\gamma$ in $G[\gamma]$. We show $(\gamma_1(\text{if } M_1 \text{ then } N_1 \text{ else } N_1'), \gamma_2(\text{if } M_2 \text{ then } N_2 \text{ else } N_2'))$ in $E[\tau]$ (1).

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

Case $V$ is $\text{tt}$: Then, $(\gamma_i M_i \text{ then } \gamma_i N_i \text{ else } \gamma_i N_i') \Downarrow \gamma_i N_i$, i.e. $\gamma_i((\text{if } M_i \text{ then } N_i \text{ else } N_i')) \Downarrow \gamma_i N_i$.

From the premise $\Gamma \vdash N_1 \sim N_2 : \tau$, we have $(\gamma_1 N_1, \gamma_2 N_2)$ in $E[\tau]$ and (1) follows by closure by inverse reduction.

Case $V$ is $\text{ff}$: 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 $\text{tt}$ or $\text{ff}$: 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 C-App, induction hypothesis, and Rule C-App.

Case $M$ is $\lambda x : \tau. N$: By inversion of the typing rule C-App, induction hypothesis, and Rule C-App.
Proof: Logical equivalence is a consistent congruence, hence included in observational equivalence which is the coarsest such relation.

Theorem 18 (Completeness of logical equivalence) Observational equivalence of closed terms implies logical equivalence. That is $(\equiv) \subseteq (\sim)$.

Proof: Proof by induction on $\tau$.

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

Case $\tau \rightarrow \sigma$: By congruence of observational equivalence. By hypothesis, we have $M_1 \equiv_{\tau \rightarrow \sigma} M_2$ (1). To show $M_1 \sim_{\tau \rightarrow \sigma} M_2$ (2), it suffices to show $M_1 V_1 \sim_{\tau} M_2 V_2$ (3).

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

Exercise 42 (Application) Let not be $\lambda x : \mathbb{B}. (\text{if } x \text{ then } ff \text{ else } tt)$ and $M$ and $M'$ be the expressions $\lambda x : \mathbb{B}. \lambda y : \tau. \lambda z : \tau. \text{if not } x \text{ then } y \text{ else } z$ and $\lambda x : \mathbb{B}. \lambda y : \tau. \lambda z : \tau. \text{if } x \text{ then } z \text{ else } y$. Show that $M \equiv_{\mathbb{B} \rightarrow \tau \rightarrow \tau} M'$.

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

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

Case $V_0 = V_0' = ff$: is symmetric.

7.5 Logical relations in F

We now extend observational and logical equivalence to System F.

$$\begin{align*}
\tau & ::= \ldots | \alpha | \forall \alpha. \tau \\
M & ::= \ldots | \Lambda \alpha. M | M \tau
\end{align*}$$

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

Definition 8 (observational equivalence) We defined $\Delta; \Gamma \vdash M \equiv_{\tau} N$ as

$$\forall \mathcal{C} : (\Delta; \Gamma \triangleright \tau) \sim (\emptyset; \emptyset \triangleright \mathbb{B}), \quad \mathcal{C}[M] \equiv \mathcal{C}[N]$$

We write $M \equiv_{\tau} N$ for $\emptyset; \emptyset \vdash M \equiv_{\tau} N$ (in particular, when $\tau$, $M$, and $N$ are closed).
7.5. LOGICAL REL. IN F

7.5.1 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 semantics of type variables. A key point however, is to interpret type variables by heterogeneous values, relating values of different types on both sides.

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

\[
R \in R^t(\rho_1, \rho_2) \iff \forall (V_1, V_2) \in R, \forall W_1, W_2, W_1 \approx_{\rho_1} V_1 \land W_2 \approx_{\rho_2} V_2 \implies (W_1, W_2) \in R
\]

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 \( R_1 \overset{\Delta}{=} \{ (\text{tt}, 0), (\text{ff}, 1) \} \) and \( R_2 \overset{\Delta}{=} \{ (\text{tt}, 0) \} \cup \{ (\text{ff}, n) \mid n \in \mathbb{Z}^+ \} \) are admissible relations in \( R(\text{B, int}) \). By contrast \( R_3 \overset{\Delta}{=} \{ (\text{tt}, \lambda x: \cdot. 0), (\text{ff}, \lambda x: \cdot. 1) \} \) is in \( R(\text{B, } \tau \mapsto \text{int}) \) but it is not admissible. Indeed, taking \( M_0 \overset{\Delta}{=} \lambda x: \cdot. (\lambda z: \text{int}. z) 0 \). we have \( M_0 \approx_{\tau \mapsto \text{int}} \lambda x: \cdot. 0 \) but (tt, \( M_0 \)) is not in \( R_3 \).

**Interpretation of type environments** We interpret type variables \( \alpha \) by triples of the form \( (\rho_1, \rho_2, R) \) where \( R \in R(\rho_1, \rho_2) \). We write \( \eta \) for mappings of type variables to such triples. Given a list of type variables \( \Delta \), we define the set \( D[\Delta] \) of interpretations of \( \Delta \) as:

\[
D[\emptyset] \overset{\Delta}{=} \{ \emptyset \}
D[\Delta, \alpha] \overset{\Delta}{=} \{ \eta, \alpha \mapsto (\rho_1, \rho_2, R) \mid \eta \in D[\Delta] \land R \in R(\rho_1, \rho_2) \} 
\]
Definition 9 (Logical equivalence for closed terms)
\[ \forall [\alpha]_\eta \triangleq \eta_R(\alpha) \]
\[ \forall [\forall \alpha. \tau]_\eta \triangleq \{(V_1, V_2) \mid V_1 \vdash \eta_1 (\forall \alpha. \tau) \land V_2 \vdash \eta_2 (\forall \alpha. \tau) \land \forall \rho_1, \rho_2, R \in R(\rho_1, \rho_2), (V_1 \rho_1, V_2 \rho_2) \in \mathcal{E}[^{\tau}_\eta, \alpha \leftarrow (\rho_1, \rho_2, R)]\} \]
\[ \forall [B]_\eta \triangleq \{(tt, tt), (ff, ff)\} \]
\[ \forall [\tau \rightarrow \sigma]_\eta \triangleq \{(V_1, V_2) \mid V_1 \vdash \eta_1 (\tau \rightarrow \sigma) \land V_2 \vdash \eta_2 (\tau \rightarrow \sigma) \land \forall (W_1, W_2) \in \forall [\tau]_\eta, (V_1 W_1, V_2 W_2) \in \mathcal{E}[\sigma]_\eta\} \]
\[ \mathcal{E}[\tau]_\eta \triangleq \{(M_1, M_2) \mid M_1 : \eta_1 \tau \land M_2 : \eta_2 \tau \land \exists (V_1, V_2) \in \forall [\tau]_\eta, M_1 \Downarrow V_1 \land M_2 \Downarrow V_2\} \]
\[ \mathcal{G}[\varnothing]_\eta \triangleq \{\varnothing\} \]
\[ \mathcal{G}[\Gamma, x : \tau]_\eta \triangleq \{\gamma, x \mapsto (V_1, V_2) \mid \gamma \in \mathcal{G}[\Gamma]_\eta \land (V_1, V_2) \in \forall [\tau]_\eta\} \]

Notice that there are really just two new cases \( \forall [\alpha]_\eta \) and \( \forall [\forall \alpha. \tau]_\eta \), as the other cases are just adjusting the previous definition to carry around the environment \( \eta \) (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 \( \rho_1 \) and \( \rho_2 \) and therefore heterogeneous pairs of types in \( R(\rho_1, \rho_2) \) to interpret \( \alpha \).

Definition 10 (Logical equivalence for open terms) We say \( \Delta; \Gamma \vdash M \sim M' : \tau \) as
\[ \forall \eta \in \mathcal{D}[^{\Delta}_\eta], \forall \gamma \in \mathcal{G}[\Gamma]_\eta, (\eta_1 (\gamma_1 M_1), \eta_2 (\gamma_2 M_2)) \in \mathcal{E}[\tau]_\eta \]

We also write \( M_1 \sim M_2 \) for \( \vdash M_1 \sim M_2 : \tau \) (i.e. \( \varnothing; \varnothing \vdash M_1 \sim M_2 : \tau \)). In this case, \( \tau \) is a closed type and \( M_1 \) and \( M_2 \) are closed terms of type \( \tau \); 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 37 (Compositionality)
Assume \( \Delta \vdash \sigma \) and \( \Delta, \alpha \vdash \tau \) and \( \eta \in \mathcal{D}[^{\Delta}_\eta] \). Then,
\[ \forall [\tau[\alpha \mapsto \sigma]]_\eta = \forall [\tau]_{\eta, \alpha \leftarrow (\eta_1 \sigma, \eta_2 \sigma, \nu[\sigma])}. \]

Proof: Let us write \( \theta \) for \( [\alpha \mapsto \sigma] \) and \( \eta^\alpha \) for \( \eta, \alpha \leftarrow (\eta_1 \sigma, \eta_2 \sigma, R) \). We show \( \forall [\tau \theta]_\eta = \forall [\tau]_{\eta^\alpha} \).

Case \( \tau \) is \( \alpha \): The right-hand side \( \forall [\alpha]_{\eta^\alpha} \) is by definition \( \eta^\alpha_R(\alpha) \), which is \( R(\alpha) \), i.e. \( \forall [\sigma]_\eta \) by hypothesis.

Case \( \tau \) is \( \sigma \rightarrow \sigma' \): Since \( (\sigma \rightarrow \sigma') \theta \) is \( \sigma \theta \rightarrow \sigma' \theta \), the left-hand side is \( \forall [\sigma \theta \rightarrow \sigma' \theta]_\eta \), i.e. by definition:
\[ \{(V_1, V_2) \mid \forall (W_1, W_2) \in \forall [\sigma \theta]_\eta, (V_1 W_1, V_2 W_2) \in \mathcal{E}[\sigma' \theta]_\eta\} \]
By induction hypothesis, we may replaced $\forall[\sigma\theta]_\eta$ by $\forall[\sigma]_{\eta^\sigma}$ and $E[\sigma'\theta]_\eta$ by $E[\sigma']_{\eta^\rho}$ which gives exactly the definition of the right-hand side $\forall[\sigma \rightarrow \sigma']_{\eta^\rho}$.

Case $\tau$ is $B$: Both sides are equal to $\forall[B]$.

Case $\tau$ is $\forall\beta.\sigma$: Assume $\alpha \not= \beta$. Since $\theta(\forall\alpha.\sigma)$ is then $\forall\alpha.\theta\sigma$, the left-hand side is $\forall[\forall\alpha.\sigma\theta]_\eta$ which is, by definition:

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

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 $E[\sigma\theta]_\eta$ by $E[\sigma']_{\eta^\rho}$, which gives the definition of the right-hand side.

**Theorem 19 (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 38 (Compatibility lemmas)** We redefined previous the lemmas to work in a typing context of the form $\Delta, \Gamma$ instead of $\Gamma$. In addition, we have:

**C-TABs**

$$\frac{\Delta, \alpha; \Gamma \vdash M_1 \sim M_2 : \tau}{\Delta; \Gamma \vdash \Lambda\alpha. M_1 \sim \Lambda\alpha. M_2 : \forall \alpha. \tau}$$

**C-TAPP**

$$\frac{\Delta; \Gamma \vdash M_1 \sim M_2 : \forall \alpha. \tau \quad \Delta \vdash \sigma}{\Delta; \Gamma \vdash M_1 \sigma \sim M_2 \sigma : \tau[\alpha \mapsto \sigma]}$$

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

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

We may assume $\alpha \notin \text{dom}(\gamma)$ w.l.o.g.. Then $\eta_1(\gamma_1(\Lambda\alpha. M_1))) \rho_i$ is equal to $\eta_1(\lambda \gamma_1. M_1))$ which reduces to $\eta_1(\gamma_1(M_1))$, i.e. $\eta'_1(\gamma_1(M_1))$ where $\gamma'_1$ is $\gamma_1 \mapsto \rho_i$.

Since $\gamma'_1 \in D[\Delta, \alpha]$, we have by $(\eta_1(\gamma'_1(M_1)), \eta_2(\gamma'_2(M_2))) \in E[\tau]_{\eta, \alpha \mapsto (\rho_1, \rho_2, R)}$ by IH applied to (1), from which (2) follows by closure under inverse reduction.

**Rule:** 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 $\eta \in D[\Delta]$ and $\gamma \in G[\Gamma]$. 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 $\eta'$ is $\eta, \alpha \mapsto (\eta_1 \sigma, \eta_2 \sigma, \mathcal{V}[\sigma]_{\eta})$, which exactly proves (2) by compositionality. (Notice, that by corollary 40 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 20 (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 39 (Respect for observational equivalence)** Under the admissibility condition, If $(M_1, M_2) \in \mathcal{E}[\tau]_{\eta}$ and $M_1 \cong_{\eta_1 \tau} N_1$ and $M_2 \cong_{\eta_2 \tau} N_2$, then $(N_1, N_2) \in \mathcal{E}[\tau]_{\eta}$.

**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 the lemma that falls in the two previous cases.

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

**Case $\tau$ is $\forall \alpha. \sigma$:** Assume $R \in \mathcal{R}(\rho_1, \rho_2)$. Let $\eta^a$ be $\eta, \alpha \mapsto (\rho_1, \rho_2, R)$. It suffices to show $(M_1 \rho_1, M_2 \rho_2) \in \mathcal{E}[\sigma]_{\eta^a}$ (4). We have $(M_1 \rho_1, M_2 \rho_2) \in \mathcal{E}[\sigma]_{\eta_1}$, from (1). By congruence applied to (2), we have $N_1 \rho_1 \cong_{\eta_1 \gamma_1} M_1 \rho_1$. Then (4) follows by induction hypothesis at type $\sigma$.

**Case $\tau$ is $\alpha$:** We know that $(M_1, M_2)$ reduces to a pair $(V_1, V_2)$ in $\mathcal{V}[\alpha]_{\eta}$, 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 $N_1 \not\Downarrow V$. This follows from $N_1 \cong_{\eta_1 \alpha} M_1$ since observational equivalence of closed terms is closed by reduction.

**Case $\tau$ is $B$:** By definition $\mathcal{E}[B]_{\eta}$ does not depend on $\eta$ and is equal to $\approx_B$, which is included in $\cong_B$ and closed by transitivity.

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

**Corollary 40** Under the admissibility condition, the relation $\mathcal{V}[\tau]_{\eta}$ 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 41 (Closure by observational equivalence)** Under the admissibility condition, if $\Delta; \Gamma \vdash M_1 \sim_1 M_2 : \tau$ and $\Delta; \Gamma \vdash M_1 \cong N_1 : \tau$ and $\Delta; \Gamma \vdash M_2 \cong N_2 : \tau$, then $\Delta; \Gamma \vdash N_1 \sim_1 N_2 : \tau$.
This lemma is used in the proof of correctness of logical equivalence.

**Proof**: By symmetry, we may just show it when $N_1$ is $M_1$ 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 $\tau$.

Assume that $\Delta, \Gamma \vdash M_1 \equiv M_2 : \tau$ (1) and $\Delta; \Gamma \vdash N_1 \equiv N_2 : \tau$ (2). Assume $\eta$ in $\mathcal{D}[\Delta]$ and $\gamma$ in $\mathcal{G}[\Gamma]$. We are to show that $(\eta_1 \gamma_1 N_1, \eta_2 \gamma_2 M_1)$ is in $E[\tau]$. Then, the conclusion (3) follows by respect for observational equivalence.

**Theorem 21 (Completeness of logical equivalence)** Under the admissibility condition, observational equivalence implies logical equivalence.

If $\Delta; \Gamma \vdash M_1 \equiv M_2 : \tau$ then $\Delta; \Gamma \vdash M_1 \sim M_2 : \tau$.

In particular, $(\equiv \tau) \subseteq (\sim \tau)$ for closed types $\tau$.

**Proof**: Assume $\Delta; \Gamma \vdash M_1 \equiv M_2 : \tau$. 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. ($\sim \tau$ means $\sim$ when admissibility is required—for all relations.)

As a particular case, for closed terms, we have $M_1 \sim M_2$ iff $M_1 \equiv M_2$.

**Lemma 42 (Extensionality)**

- $M_1 \equiv \forall \alpha \rightarrow \sigma M_2 \iff \forall V \in \text{Val}(\tau), M_1 V \equiv \sigma M_2 V \iff \forall N \in \text{Exp}(\tau), M_1 N \equiv \sigma M_2 N$
- $M_1 \equiv \forall \alpha \rightarrow \tau M_2 \iff \forall \rho, M_1 \rho \equiv \tau \alpha \rightarrow \rho M_2 \rho$.

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). The right most equivalence for value abstractions results from the closure of $E[\tau]$ by reduction and anti-reduction.

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

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

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

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

The proof uses respects for observational equivalence.

7.6 Applications

Exercise 43 (Inhabitants of $\forall \alpha. \alpha \rightarrow \alpha$) If $M : \forall \alpha. \alpha \rightarrow \alpha$, then $M \approx_{\forall \alpha. \alpha \rightarrow \alpha} \text{id}$ where $\text{id} \triangleq \Lambda \alpha. \lambda x : \alpha. x$.

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

By parametricity, we have $M \approx_{\forall \alpha. \alpha \rightarrow \alpha} M$ (2). Consider $R$ in $\mathcal{R}(\rho, \rho)$ equal to $\{(V, V)\}$ and $\eta$ 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 $(M \rho V, M \rho V) \in E[\alpha]_\eta$, 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 44 (Inhabitants of $\forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$) If $M : \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$, then either $M \approx_{\alpha}$

\[ W_1 \triangleq \Lambda \alpha. \lambda x_1 : \alpha. \lambda x_2 : \alpha. x_1 \text{ or } M \approx_{\alpha} W_2 \triangleq \Lambda \alpha. \lambda x_1 : \alpha. \lambda x_2 : \alpha. x_2 \]

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

Let $\rho$ and $V_1, V_2 : \rho$ be fixed. Consider $R$ equal to $\{(tt, V_1), (ff, V_2)\}$ in $\mathcal{R}(\mathcal{B}, \rho)$ and $\eta$ be $\alpha \mapsto (\mathcal{B}, \rho, R)$. We have $(tt, V_1) \in \mathcal{V}[\alpha]_\eta$ since $R(tt, V_1)$ and, similarly, $(ff, V_2) \in \mathcal{V}[\alpha]_\eta$. By parametricity: we have $(M, M) \in E[\sigma]$.

\[ (M \mathcal{B} tt \mathcal{F} f, M \rho V_1 V_2) \in E[\alpha]_\eta \], which means
Indeed, assume an integer 0), e.g., since, the pair must reduce to a pair in \( R \), observational equivalence.

That is (1).

Exercise 45 (Inhabitants of \( \forall \alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \)) Let \( \text{nat} \) be \( \forall \alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \).

If \( M : \text{nat} \), then \( M \cong_{\text{nat}} N_n \) for some integer \( n \), where \( N_n \cong_{\Delta} \Lambda \alpha. \lambda f : \alpha \rightarrow \alpha. \lambda x : \alpha. f^n x \).

(That is, the inhabitants of \( \text{nat} \) are the Church naturals.)

**Solution:** By extensionality, it suffices to show that there exists \( n \) such for any closed type \( \rho \) and closed values \( V_1 : \rho \rightarrow \rho \) 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 \( \rho \) and \( V_1 : \rho \rightarrow \rho \) and \( V_2 : \rho \) be fixed. Let \( Z \) and \( S \) be \( M_0 \text{nat} \) and \( M_2 \text{nat} \). Let \( R \) be \( \{ (W_1, W_2) \mid \exists k \in \mathbb{N}, S^k Z \cong_{\text{nat}} W_1 \land V_1^k V_2 \cong_{\rho} W_2 \} \) in \( \mathcal{R}(\text{nat}, \rho) \) and \( \eta \) be \( \alpha \mapsto (\text{nat}, \rho, R) \).

We have \((Z, V_2) \in \mathcal{V}[\alpha]_{\eta}(2) \) since \((R(Z, V_2)) \) (reduce both sides for \( k = 0 \)). We also have \((S, V_1) \in \mathcal{V}[\alpha \rightarrow \alpha]_{\eta}(3) \), (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 \( S^k Z \cong_{\text{nat}} W_1 \) and \( V_1^k V_2 \cong_{\rho} W_2 \). By congruence \( S W_1 \cong_{\text{nat}} S^{k+1} Z \) and \( V_1 W_2 \cong_{\rho} V_1^{k+1} V_2 \). Since \((S^{k+1} Z, V_1^{k+1} V_2) \) is in \( \mathcal{E}[\alpha]_{\eta} \), so is \((S W_1, V_1 W_2) \) by closure by observational equivalence.

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

\[
\forall \rho, \forall V_1 \in \text{Val}(\rho \rightarrow \rho), \forall V_2 \in \text{Val}(\rho), \exists n \in \mathbb{N}, M \text{ nat } S Z \cong_{\text{nat}} S^n Z \land M \rho V_1 V_2 \cong_{\rho} V_1^n V_2
\]

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

\[
\exists n \in \mathbb{N}, \forall \rho, \forall V_1 \in \text{Val}(\rho \rightarrow \rho), \forall V_2 \in \text{Val}(\rho), M \text{ nat } S Z \cong_{\text{nat}} S^n Z \land M \rho V_1 V_2 \cong_{\rho} V_1^n V_2
\]
which implies (1). 

Exercise 46 (sort)
Assume \( \text{sort} : \forall \alpha. (\alpha \to \alpha \to \mathbf{B}) \to \text{list} \alpha \to \text{list} \alpha \) (1). Then for all \( g \) of ground type \( \rho_1 \to \rho_2 \), and all (comparison) functions \( \text{cmp}_1 \) of type \( \rho_1 \to \rho_1 \to \mathbf{B} \) and \( \text{cmp}_2 \) of type \( \rho_2 \to \rho_2 \to \mathbf{B} \) satisfying

\[
\forall V, W \in \text{Val}(\rho_1), \quad \text{cmp}_2 \, (g \, V) \, (g \, W) \cong \text{cmp}_1 \, V \, W
\]  

we have, for all \( U \) in \( \text{Val}(\text{list} \, \rho_1) \),

\[
\text{sort} \, \rho_2 \, \text{cmp}_2 \, (\text{map} \, \rho_1 \, \rho_2 \, g \, U) \cong \text{map} \, \rho_1 \, \rho_2 \, g \, (\text{sort} \, \rho_1 \, \text{cmp}_1 \, U)
\]  

Solution: Let \( \rho_1 \) and \( \rho_2 \) 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 \( \eta \) be \( \alpha \to (\rho_1, \rho_2, R) \).

We have \( \text{sort} \sim_\sigma \text{sort} \) where \( \sigma \) is \( \forall \alpha. (\alpha \to \alpha \to \mathbf{B}) \to \text{list} \alpha \to \text{list} \alpha \).

The hypothesis (2) implies \( (\text{cmp}_1, \text{cmp}_2) \) in \( \mathcal{V}[\alpha \to \alpha \to \mathbf{B}]_\eta \). Indeed, consider \( (V_1, V_2) \) and \( (W_1, W_2) \) in \( \mathcal{V}[\alpha]_\eta \), 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

\[
\text{cmp}_2 \, V_2 \, W_2 \cong_B \text{cmp}_2 \, (g \, V_1) \, (g \, W_1) \cong_B \text{cmp}_1 \, V_1 \, W_1
\]

Hence, \( \text{cmp}_2 \, V_2 \, W_2 \cong_B \text{cmp}_1 \, V_1 \, W_1 \) as expected.

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

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

\[
(\text{sort} \, \rho_1 \, \text{cmp}_1 \, U, \text{sort} \, \rho_2 \, \text{cmp}_2 \, (\text{map} \, \rho_1 \, \rho_2 \, g \, U))
\]

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

\[\square\]

7.7 Extensions

7.7.1 Natural numbers

We have shown that all expressions of type \( \text{nat} \) behave as natural numbers. Hence, natural numbers are definable in System F.

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

\[
M_1 \cong_{\text{nat}} M_2 \overset{\text{def}}{\iff} \exists n : \text{nat}, \ M_1 \downarrow n \land M_2 \downarrow n
\]
As for the logical equivalence, we defined
\[ \mathcal{V}[\text{nat}] = \{(n, n) \mid n \in \text{Val(nat)}\} \]
Notice that \text{nat} is another observable type. All properties are preserved.

### 7.7.2 Products

**Encodable** Given closed types \( \tau_1 \) and \( \tau_2 \), we defined
\[
\tau_1 \times \tau_2 \triangleq \forall \alpha. (\tau_1 \to \tau_2 \to \alpha) \to \alpha
\]
\[
(M_1, M_2) \triangleq \Lambda \alpha. \lambda x: \tau_1 \to \tau_2 \to \alpha. x M_1 M_2
\]
\[
M.i \triangleq M (\lambda x_1: \tau_1. \lambda x_2: \tau_2. x_i)
\]

**Lemma 43**
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:
\[
\mathcal{V}[\tau \times \sigma]_\eta \triangleq \{(V_1, V_2) \mid (V_1, V_2) \in \mathcal{V}[\tau]_\eta \land (W_1, W_2) \in \mathcal{V}[\sigma]_\eta\}
\]

### 7.7.3 Sums

\[
\mathcal{V}[\tau + \sigma]_\eta \triangleq \{(\text{inj}_1 V_1, \text{inj}_1 V_2) \mid (V_1, V_2) \in \mathcal{V}[\tau]_\eta\} \cup \{(\text{inj}_2 V_2, \text{inj}_2 V_2) \mid (V_1, V_2) \in \mathcal{V}[\sigma]_\eta\}
\]

### 7.7.4 Lists

We could extend the language with lists and define:
\[
\mathcal{V}[\text{list } \tau]_\eta \triangleq \{(V_1^1; \ldots; V_n^n), (V_2^1; \ldots; V_n^n) \mid n \in \mathbb{N} \land \forall k \in [1, n], (V_1^k, V_2^k) \in \mathcal{V}[\tau]_\eta\}
\]
Assume given a function \( g \) from \( \rho_1 \) to \( \rho_2 \). 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 \( \eta \) be \( \alpha \mapsto (\rho_1, \rho_2, R) \). Then \( \mathcal{V}[\text{list } \alpha]_\eta \) is composed of all pairs \( (W_1, W_2) \) such that \( W_2 \cong \text{map } \rho_1 \rho_2 g W_1 \) and \( \mathcal{E}[\text{list } \alpha]_\eta \) is composed of all pairs \( (N_1, N_2) \) such that \( N_2 \cong \text{map } \rho_1 \rho_2 g N_1 \).

### 7.7.5 Existential types

We define:
\[
\mathcal{V}[\exists \alpha. \tau]_\eta \triangleq \{\text{pack } V_1, \rho_1 \text{ as } \exists \alpha. \tau, \text{pack } V_2, \rho_2 \text{ as } \exists \alpha. \tau \mid V_1 \vdash \eta_1(\exists \alpha. \tau) \land V_2 \vdash \eta_2(\exists \alpha. \tau) \land \exists \rho_1, \rho_2, R \in \mathcal{R}(\rho_1, \rho_2), (V_1, V_2) \in \mathcal{E}[\tau]_{\eta_1, \eta_2}(\rho_1, \rho_2, R)\}
\]
Example 2 Consider \(V_1 \triangleq (\text{not, tt})\), and \(V_2 \triangleq (\text{succ, 0})\) and \(\sigma \triangleq (\alpha \rightarrow \alpha) \times \alpha\). Let \(R\) in \(\mathcal{R}(\mathbb{B}, \text{nat})\) be \(\{(\text{tt, } 2n), (\text{ff, } 2n + 1) \mid n \in \mathbb{N}\}\) and \(\eta\) be \(\alpha \mapsto (\mathbb{B}, \text{nat}, R)\).

We have \((\text{pack } V_1 \mathbb{B} \text{ as } \exists \alpha. \sigma, \text{ pack } V_2 \text{ nat as } \exists \alpha. \sigma) \in \mathcal{V}[\exists \alpha. \sigma]\). To see this it suffices to show \((V_1, V_2) \in \mathcal{V}[\sigma]_\eta\), that is \(((\text{not, tt}), (\text{succ, 0})) \in \mathcal{V}[(\alpha \rightarrow \alpha) \times \alpha]_\eta\). In turn, it suffices to show both \((\text{not, succ}) \in \mathcal{V}[(\alpha \rightarrow \alpha)]_\eta\) and \((\text{tt, 0}) \in \mathcal{V}[\alpha]_\eta\). The latter holds by construction since \((\text{tt, 0}) \in R\). 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

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

In both cases, \((\text{not } W_1, \text{succ } W_2)\) 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 \(\rho_1\) and \(\rho_2\).

Assume that \(\rho_1\) and \(\rho_2\) are two closed representation types and \(R\) is in \(\mathcal{R}(\rho_1, \rho_2)\). Let \(\eta\) 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 \(\forall \alpha. \tau \rightarrow \sigma\) with \(\alpha \notin \text{fv}(\sigma)\); it must use the argument parametrically, and the result is independent of the witness type. Indeed the client satisfies \((M, M) \in \mathcal{E}[\forall \alpha. \tau \rightarrow \sigma]_\eta\) and therefore \((M\rho_1 N_1, M\rho_2 N_2)\) is in \(\mathcal{E}[\sigma]\) (as \(\alpha\) is not free in \(\sigma\)), 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 \(\rho_1\) is indistinguishable from the behavior with implementation \(N_2\) with representation type \(\rho_2\).

7.7.6 Step-indexed logical relations

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

\[\tau ::= \ldots | \mu \alpha. \tau\]

A naive definition would be

\[\mathcal{V}[(\mu \alpha. \tau)]_\eta = \mathcal{V}[(\alpha \mapsto \mu \alpha. \tau)]_\eta\]

But this is ill-founded, because \([\alpha \mapsto \mu \alpha. \tau]_\tau\) is usually larger than \(\tau\).

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 $n \in \mathbb{N}$ that relates values of type $\tau$ up to $n$ reduction steps.

- $\mathcal{V}_k[B]_\eta = \{(tt, tt), (ff, ff)\}$
- $\mathcal{V}_k[\tau \rightarrow \sigma]_\eta = \{(V_1, V_2) | \forall j < k, \forall (W_1, W_2) \in \mathcal{V}_j[\tau]_\eta, (V_1 W_1, V_2 W_2) \in \mathcal{E}_j[\sigma]_\eta\}$
- $\mathcal{V}_k[\forall \alpha. \tau]_\eta = \{(V_1, V_2) | \forall \rho_1, \rho_2, R \in \mathcal{R}^k(\rho_1, \rho_2), \forall j < k, (V_1 \rho_1, V_2 \rho_2) \in \mathcal{V}_j[\tau]_{\eta, \alpha \mapsto (\rho_1, \rho_2, R)}\}$
- $\mathcal{V}_k[\mu \alpha. \tau]_\eta = \mathcal{V}_{k-1}[\exists \alpha \mapsto \mu \alpha. \tau]_{\eta, k}$
- $\mathcal{E}_k[\tau]_\eta = \{(M_1, M_2) | \forall j < k, M_1 \downarrow_j V_1 \implies \exists V_2, M_2 \downarrow_j V_2 \land (V_1, V_2) \in \mathcal{V}_{k-j}(\tau)_{\eta}\}$

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 $\rho_1$ and $\rho_2$ of length (at least) $j$.

Notice that the relation is asymmetric.

We define

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

and

$$\Delta; \Gamma \vdash M_1 \sim M_2 : \tau \equiv \begin{cases} \Delta; \Gamma \vdash M_1 \preceq M_2 : \tau \\ \Delta; \Gamma \vdash M_2 \preceq M_1 : \tau \\
\end{cases}$$

Notations and proofs get a bit involved.
A tour of scala: Implicit parameters. Part of scala documentation.


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


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


