Primitive Recursion
for
Higher Order Abstract Syntax

Joëlle Despeyroux[1], Frank Pfenning, Carsten Schürmann

August 30, 1996

CMU-CS-96-172

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Higher-order abstract syntax is a central representation technique in logical frameworks which maps variables of the object language into variables in the meta-language. It leads to concise encodings, but is incompatible with functions defined by primitive recursion or proofs by induction.

In this paper we propose an extension of the simply-typed lambda-calculus with iteration and case constructs which preserves the adequacy of higher-order abstract syntax encodings. The well-known paradoxes are avoided through the use of a modal operator which obeys the laws of S4. In the resulting calculus many functions over higher-order representations can be expressed elegantly. Our central technical result, namely that our calculus is conservative over the simply-typed lambda-calculus, is proved by a rather complex argument using logical relations.

We view our system as an important first step towards allowing the methodology of LF to be employed effectively in systems based on induction principles such as ALF, Coq, or Nuprl, leading to a synthesis of currently incompatible paradigms.

[1]INRIA, F-06902 Sophia-Antipolis Cedex, joelle.despeyroux@sophia.inria.fr

This work was sponsored NSF Grant CCR-9303383.

Keywords: modal lambda calculus, higher order abstract syntax, primitive recursion

Contents

1 Introduction

Higher-order abstract syntax is a central representation technique in many logical frameworks, that is, meta-languages designed for the formalization of deductive systems. The basic idea is to represent variables of the object language by variables in the meta-language. Consequently, object language constructs which bind variables must be represented by meta-language constructs which bind the corresponding variables.

This deceptively simple idea, which goes back to Church [Chu40] and Martin-Löf's system of arities [NPS90], has far-reaching consequences for the methodology of logical frameworks. On one hand, encodings of logical systems using this idea are often extremely concise and elegant, since common concepts and operations such as variable binding, variable renaming, capture-avoiding substitution, or parametric and hypothetical judgments are directly supported by the framework and do not need to be encoded separately in each application. On the other hand, higher-order representations are no longer inductive in the usual sense, which means that standard techniques for reasoning by induction do not apply.

Various attempts have been made to preserve the advantages of higher-order abstract syntax in a setting with strong induction principles [DH94, DFH95], but none of these is entirely satisfactory from a practical or theoretical point of view.

In this paper we take a first step towards reconciling higher-order abstract syntax with induction by proposing a system of *primitive recursive functionals* that permits iteration over subjects of functional type. In order to avoid the well-known paradoxes which arise in this setting (see Section 3), we decompose the primitive recursive function space $A \Rightarrow B$ into a modal operator and a parametric function space $(\Box A) \to B$. The inspiration comes from linear logic which arises from a similar decomposition of the intuitionistic function space $A \supset B$ into a modal operator and a linear function space $(!A) \multimap B$.

The resulting system allows, for example, iteration over the structure of expressions from the untyped λ-calculus when represented using higher-order abstract syntax. It is general enough to permit iteration over objects of *any* simple type, constructed over *any* simply typed signature and thereby encompasses Gödel's system T [Göd90]. Moreover, it is conservative over the simply-typed λ-calculus which means that the compositional adequacy of encodings in higher-order abstract syntax is preserved. We view our calculus as an important first step towards a system which allows the methodology of logical frameworks such as LF [HHP93] to be incorporated into systems such as Coq [PM93] or ALF [Mag95].

The remainder of this paper is organized as follows: Section 2 reviews the idea of higher order abstract syntax and introduces the simply typed λ-calculus ($\lambda^{\to}$) which we extend to a modal λ-calculus in Section 3. Section 4 then presents the iteration and Section 5 definition by cases. In Section 6 we start with the technical discussion and introduce some auxiliary concepts and derive some basic results. Section 7 shows the proof of the canoncial form theorem which is the essential for the proof of type preservation (Section 8) and our central result, namely that our system is conservative over $\lambda^{\to}$ (Section 9). Finally, Section 10 assesses the results, compares some related work, and outlines future work.

2 Higher-Order Abstract Syntax

Higher-order abstract syntax exploits the full expressive power of a typed λ-calculus for the representation of an object language, where λ-abstraction provides the mechanism to represent binding. In this paper, we restrict ourselves to a simply typed meta-language, although we recognize that an extension allowing dependent types and polymorphism is important future work (see Section 10). Our formulation of the simply-typed meta-language is standard.

$$
\begin{array}{lrcl}
\text{Pure types:} & B & ::= & a \mid B_1 \to B_2 \\
\text{Objects:} & M & ::= & x \mid c \mid \lambda x\!:\!A.\, M \mid M_1\ M_2 \\
\\
\text{Context:} & \Psi & ::= & \cdot \mid \Psi, x : B \\
\text{Signature:} & \Sigma & ::= & \cdot \mid \Sigma, a : \mathrm{type} \mid \Sigma, c : B
\end{array}
$$

We use a for type constants, c for object constants and x for variables. We assume that constants and variables are declared at most once in a signature and context, respectively. As usual, we apply tacit renaming of bound variables to maintain this assumption, and to guarantee capture-avoiding substitution. Before we proceed with the presentation of the typing rules we introduce define the union $\Psi_1 \cup \Psi_2$ of two contexts Ψ_1 and Ψ_2.

Definition 2.1 (Context Union)
Rules:

$$\Psi \cup \cdot = \Psi \qquad (\mathsf{CuBase})$$

$$\Psi_1 \cup (\Psi_2, x : A) = (\Psi_1 \cup \Psi_2), x : A \qquad (\mathsf{CuInd})$$

and the lookup of the type of a variable x in Ψ as $\Psi(x)$ as:

Definition 2.2 (Context Access) $\Psi(x) = A$ *iff there are* Ψ_1, Ψ_2 *s.t.* $\Psi = (\Psi_1, x : A) \cup \Psi_2$

It might sound awkward to define these notion in such depth of detail, but reasoning with variables requires a rigorous treatment. For our typing and evaluation judgments we also fix a signature Σ so we do not have to carry it around.

Definition 2.3 (Typing judgment) $\Psi \vdash M : B$ *is defined by:*

$$\frac{\Psi(x) = B}{\Psi \vdash x : B}\,\mathsf{StpVar} \qquad \frac{\Sigma(c) = B}{\Psi \vdash c : B}\,\mathsf{StpConst}$$

$$\frac{\Psi, x : B_1 \vdash M : B_2}{\Psi \vdash \lambda x\!:\!B_1.\, M : B_1 \to B_2}\,\mathsf{StpLam} \qquad \frac{\Psi \vdash M_1 : B_1 \to B_2 \qquad \Psi \vdash M_2 : B_1}{\Psi \vdash M_1\ M_2 : B_2}\,\mathsf{StpApp}$$

As running examples throughout the paper we use the representation of natural numbers and untyped λ-expressions.

Example 2.4 (Natural numbers)

$$
\begin{array}{rclll}
 & & & \mathrm{nat} & : \mathrm{type} \\
\ulcorner 0 \urcorner & = & \mathrm{z} & \mathrm{z} & : \mathrm{nat} \\
\ulcorner n+1 \urcorner & = & \mathrm{s}\ulcorner n \urcorner & \mathrm{s} & : \mathrm{nat} \to \mathrm{nat}
\end{array}
$$

Untyped λ-expressions illustrate the idea of higher-order abstract syntax: object language variables are represented by meta-language variables.

Example 2.5 (Untyped λ-expressions)

$$\text{Expressions}: \quad e \quad ::= \quad x \mid \mathbf{lam}\ x.e \mid e1@e2$$

$$\begin{array}{rcll} & & & \text{exp} : \text{type} \\ \ulcorner \mathbf{lam}\ x.e \urcorner & = & \text{lam}\ (\lambda x{:}\text{exp}.\ulcorner e \urcorner) & \text{lam} : (\text{exp} \to \text{exp}) \to \text{exp} \\ \ulcorner e_1 @ e_2 \urcorner & = & \text{app}\ \ulcorner e_1 \urcorner\ \ulcorner e_2 \urcorner & \text{app} : \text{exp} \to (\text{exp} \to \text{exp}) \\ \ulcorner x \urcorner & = & x & \end{array}$$

Not every well-typed object of the meta-language directly represents an expression of the object language. For example, we can see that $\ulcorner e \urcorner$ will never contain a β-redex. Moreover, the argument to lam which has type exp $\to$ exp will always be a λ-abstraction. Thus the image of the translation in this representation methodology is always a β-normal and η-long form. Following [HHP93], we call these forms *canonical* as defined by the following two judgments.

Definition 2.6 (Atomic and canonical forms)

1. *$\Psi \vdash V \downarrow B$ (V is atomic of type B in Ψ)*
2. *$\Psi \vdash V \Uparrow B$ (V is canonical of type B in Ψ)*

are defined by:

$$\frac{\Psi(x) = B}{\Psi \vdash x \downarrow B}\,\mathsf{AtVar} \qquad \frac{\Sigma(c) = B}{\Psi \vdash c \downarrow B}\,\mathsf{AtConst} \qquad \frac{\Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash V_2 \Uparrow B_2}{\Psi \vdash V_1\ V_2 \downarrow B_1}\,\mathsf{AtApp}$$

$$\frac{\Psi \vdash V \downarrow a}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt} \qquad \frac{\Psi, x : B_1 \vdash V \Uparrow B_2}{\Psi \vdash \lambda x{:}B_1.\,V \Uparrow B_1 \to B_2}\,\mathsf{CanLam}$$

Canonical forms play the role of "observable values" in a functional language: they are in one-to-one correspondence with the expressions we are trying to represent. For Example 2.5 (untyped λ-expressions) this is expressed by the following property, which is proved by simple inductions.

Example 2.7 (Compositional adequacy for untyped λ-expressions)

1. Let e be an expression with free variables among $x_1, \ldots, x_n$.
   Then $x_1 : \text{exp}, \ldots, x_n : \text{exp} \vdash \ulcorner e \urcorner \Uparrow \text{exp}$.
2. Let $x_1 : \text{exp}, \ldots, x_n : \text{exp} \vdash M \Uparrow \text{exp}$.
   Then $M = \ulcorner e \urcorner$ for an expression e with free variables among $x_1, \ldots, x_n$.
3. $\ulcorner \cdot \urcorner$ is a bijection between expressions and canonical forms where $\ulcorner [e'/x]e \urcorner = [\ulcorner e' \urcorner / x] \ulcorner e \urcorner$.

Since every object in $\lambda^{\to}$ has a unique $\beta\eta$-equivalent canonical form, the meaning of every well-typed object is unambiguously given by its canonical form. Our operational semantics (see Definitions 3.3 and 4.29) computes this canonical form and therefore the meaning of every well-typed object. That this property is preserved under an extension of the language by primitive recursion for higher-order abstract syntax may be considered the main technical result of this paper.

3 Modal λ-Calculus

The constructors for objects of type exp from Example 2.5 are $\mathrm{lam} : (\mathrm{exp} \to \mathrm{exp}) \to \mathrm{exp}$ and $\mathrm{app} : \mathrm{exp} \to (\mathrm{exp} \to \mathrm{exp})$. These cannot be the constructors of an *inductive* type exp, since we have a negative occurrence of exp in the argument type of lam. This is not just a formal observation, but has practical consequences: we cannot formulate a consistent induction principle for expressions in this representation. Furthermore, if we increase the computational power of the meta-language by adding definition by cases or an iterator, then not every well-typed object of type exp has a canonical form. For example,

$$\cdot \vdash \mathrm{lam}\ (\lambda E{:}\mathrm{exp}.\, \mathrm{case}\ E\ \mathrm{of}\ \mathrm{app}\ E_1\ E_2 \Rightarrow \mathrm{app}\ E_2\ E_1 \mid \mathrm{lam}\ E' \Rightarrow \mathrm{lam}\ E') : \mathrm{exp}$$

but the given object does not represent any untyped λ-expression, nor could it be converted to one. The difficulty with a **case** or iteration construct is that there are many new functions of type $\mathrm{exp} \to \mathrm{exp}$ which cannot be converted to a function in $\lambda^{\to}$. This becomes a problem when such functions are arguments to constructors, since then the extension is no longer conservative even over expressions of base type (as illustrated in the example above).

Thus we must cleanly separate the *parametric function space* $\mathrm{exp} \to \mathrm{exp}$ whose elements are convertible to the form $\lambda x{:}\mathrm{exp}.\, E$ where E is built only from the constructors app, lam, and the variable x, from the *primitive recursive function space* $\mathrm{exp} \Rightarrow \mathrm{exp}$ which is intended to encompass functions defined through case distinction and iteration. This separation can be achieved by using a modal operator: $\mathrm{exp} \to \mathrm{exp}$ will continue to contain only the parametric functions, while $\mathrm{exp} \Rightarrow \mathrm{exp} = (\Box\mathrm{exp}) \to \mathrm{exp}$ contains the primitive recursive functions.

Intuitively we interpret $\Box B$ as the type of *closed* objects of type B. We can iterate or distinguish cases over closed objects, since all constructors are statically known and can be provided for. This is not the case if an object may contain some unknown free variables. The system is non-trivial since we may also abstract over objects of type $\Box A$, but fortunately it is well understood and corresponds (via an extension of the Curry-Howard isomorphism) to the intuitionistic variant of S_4 [DP96].

In Section 4 we introduce schemas for defining functions by iteration and case distinction which require the subject to be of type $\Box B$. We can recover the ordinary scheme of primitive recursion for type nat if we also add pairs to the language. Pairs (with type $A_1 \times A_2$) are also necessary for the simultaneous definition of mutually recursive functions. Just as the modal type $\Box A$, pairs are lazy and values of these types are not observable—ultimately we are only interested in canonical forms of pure type.

The formulation of the modal λ-calculus below is copied from [DP96] and goes back to [PW95]. The language of types includes the pure types from the simply-typed λ-calculus in Section 2.

$$\begin{array}{llll}
\text{Types:} & A & ::= & a \mid A_1 \to A_2 \mid \Box A \mid A_1 \times A_2 \\
\text{Objects:} & M & ::= & c \mid x \mid \lambda x{:}A.\, M \mid M_1\ M_2 \\
 & & & \mid \mathrm{box}\ M \mid \mathrm{let\ box}\ x = M_1\ \mathrm{in}\ M_2 \mid \langle M_1, M_2\rangle \mid \mathrm{fst}\ M \mid \mathrm{snd}\ M \\
\text{Contexts:} & \Gamma & ::= & \cdot \mid \Gamma, x : A
\end{array}$$

For the sake of brevity we usually suppress the fixed signature Σ. However, it is important that signatures Σ and contexts denoted by Ψ will continue to contain only pure types, while contexts Γ and Δ may contain arbitrary types. We also continue to use B to range over pure types, while A ranges over arbitrary types. Definitions 2.1 and 2.2 extend in a trivial way to Γ (instead of Ψ), by

$$\frac{\Gamma(x) = A}{\Delta;\Gamma \vdash x : A}\,\textsf{TpVarReg} \qquad \frac{\Delta(x) = A}{\Delta;\Gamma \vdash x : A}\,\textsf{TpVarMod} \qquad \frac{\Sigma(c) = B}{\Delta;\Gamma \vdash c : B}\,\textsf{TpConst}$$

$$\frac{\Delta;\Gamma, x : A_1 \vdash M : A_2}{\Delta;\Gamma \vdash \lambda x{:}A_1.\, M : A_1 \to A_2}\,\textsf{TpLam} \qquad \frac{\Delta;\Gamma \vdash M_1 : A_2 \to A_1 \qquad \Delta;\Gamma \vdash M_2 : A_2}{\Delta;\Gamma \vdash M_1\ M_2 : A_1}\,\textsf{TpApp}$$

$$\frac{\Delta;\Gamma \vdash M_1 : A_1 \qquad \Delta;\Gamma \vdash M_2 : A_2}{\Delta;\Gamma \vdash \langle M_1, M_2\rangle : A_1 \times A_2}\,\textsf{TpPair}$$

$$\frac{\Delta;\Gamma \vdash M : A_1 \times A_2}{\Delta;\Gamma \vdash \mathrm{fst}\ M : A_1}\,\textsf{TpFst} \qquad \frac{\Delta;\Gamma \vdash M : A_1 \times A_2}{\Delta;\Gamma \vdash \mathrm{snd}\ M : A_2}\,\textsf{TpSnd}$$

$$\frac{\Delta;\cdot \vdash M : A}{\Delta;\Gamma \vdash \mathrm{box}\ M : \Box A}\,\textsf{TpBox} \qquad \frac{\Delta;\Gamma \vdash M_1 : \Box A_1 \qquad \Delta, x : A_1;\Gamma \vdash M_2 : A_2}{\Delta;\Gamma \vdash \mathrm{let\ box}\ x = M_1\ \mathrm{in}\ M_2 : A_2}\,\textsf{TpLet}$$

Figure 1: Typing judgment $\Delta;\Gamma \vdash M : A$

replacing all B's in the definition by A's. The typing judgment $\Delta;\Gamma \vdash M : A$ uses two contexts: Δ, whose variables range over closed objects, and Γ, whose variables range over arbitrary objects.

Definition 3.1 (Typing judgment) *$\Delta;\Gamma \vdash M : A$ is defined in Figure 1.*

As examples, we show some basic laws of the (intuitionistic) modal logic S_4.

Example 3.2 (Laws of S_4)

$$\begin{array}{rcl}
\mathrm{funlift} & : & \Box(A_1 \to A_2) \to \Box A_1 \to \Box A_2 \\
 & = & \lambda f{:}\Box(A_1 \to A_2).\, \lambda x{:}\Box A_1.\, \mathrm{let\ box}\ f' = f\ \mathrm{in\ let\ box}\ x' = x\ \mathrm{in\ box}\ (f'\ x') \\
\mathrm{unbox} & : & \Box A \to A \\
 & = & \lambda x{:}\Box A.\, \mathrm{let\ box}\ x' = x\ \mathrm{in}\ x' \\
\mathrm{boxbox} & : & \Box A \to \Box\Box A \\
 & = & \lambda x{:}\Box A.\, \mathrm{let\ box}\ x' = x\ \mathrm{in\ box}\ (\mathrm{box}\ x')
\end{array}$$

The rules for evaluation must be constructed in such a way that full canonical forms are computed for objects of pure type, that is, we must evaluate under certain λ-abstractions. Objects of type $\Box A$ or $A_1 \times A_2$ on the other hand are not observable and may be computed lazily. We therefore use two mutually recursive judgments for evaluation and conversion to canonical form, written $\Psi \vdash M \hookrightarrow V : A$ and $\Psi \vdash M \Uparrow V : B$, respectively. The latter is restricted to pure types, since only objects of pure type possess canonical forms. Since we evaluate under some λ-abstractions, free variables of pure type declared in Ψ may occur in M and V during evaluation.

Definition 3.3 (Evaluation judgments) *$\Psi \vdash M \hookrightarrow V : A$ and $\Psi \vdash M \Uparrow V : B$ are defined in Figure 2.*

Note that the rules $\textsf{EvApp}$ and $\textsf{EvAtomic}$ are mutually exclusive, since the evaluation of M_1 in an application $M_1\, M_2$ either yields an atomic term (with a constant or parameter at the head) or a λ-abstraction. Since constants and parameters must have pure type, the type of the argument M_2 in $\textsf{EvAtomic}$ must also be pure.

$$\frac{\Psi \vdash M \hookrightarrow V : a}{\Psi \vdash M \Uparrow V : a}\ \mathsf{EcAtomic} \qquad \frac{\Psi, x : B_1 \vdash M\ x \Uparrow V : B_2}{\Psi \vdash M \Uparrow \lambda x{:}B_1.\, V : B_1 \rightarrow B_2}\ \mathsf{EcArrow}$$

$$\frac{\Psi(x) = A}{\Psi \vdash x \hookrightarrow x : A}\ \mathsf{EvVar} \qquad \frac{\Sigma(c) = B}{\Psi \vdash c \hookrightarrow c : B}\ \mathsf{EvConst}$$

$$\frac{\cdot; \Psi, x : A_1 \vdash M : A_2}{\Psi \vdash \lambda x{:}A_1.\, M \hookrightarrow \lambda x{:}A_1.\, M : A_1 \rightarrow A_2}\ \mathsf{EvLam}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow \lambda x{:}A_2.\, M_1' : A_2 \rightarrow A_1 \qquad \Psi \vdash M_2 \hookrightarrow V_2 : A_2 \qquad \Psi \vdash [V_2/x](M_1') \hookrightarrow V : A_1}{\Psi \vdash M_1\ M_2 \hookrightarrow V : A_1}\ \mathsf{EvApp}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow V_1 : B_2 \rightarrow B_1 \qquad \Psi \vdash V_1 \downarrow B_2 \rightarrow B_1 \qquad \Psi \vdash M_2 \Uparrow V_2 : B_2}{\Psi \vdash M_1\ M_2 \hookrightarrow V_1\ V_2 : B_1}\ \mathsf{EvAtomic}$$

$$\frac{\cdot; \Psi \vdash M_1 : A_1 \qquad \cdot; \Psi \vdash M_2 : A_2}{\Psi \vdash \langle M_1, M_2 \rangle \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2}\ \mathsf{EvPair}$$

$$\frac{\Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2 \qquad \Psi \vdash M_1 \hookrightarrow V : A_1}{\Psi \vdash \mathrm{fst}\ M \hookrightarrow V : A_1}\ \mathsf{EvFst}$$

$$\frac{\Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2 \qquad \Psi \vdash M_2 \hookrightarrow V : A_2}{\Psi \vdash \mathrm{snd}\ M \hookrightarrow V : A_2}\ \mathsf{EvSnd}$$

$$\frac{\cdot; \cdot \vdash M : A}{\Psi \vdash \mathrm{box}\ M \hookrightarrow \mathrm{box}\ M : \Box A}\ \mathsf{EvBox}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow \mathrm{box}\ M_1' : \Box A \qquad \Psi \vdash [M_1'/x](M_2) \hookrightarrow V : A_2}{\Psi \vdash \mathrm{let\ box}\ x = M_1\ \mathrm{in}\ M_2 \hookrightarrow V : A_2}\ \mathsf{EvLet}$$

Figure 2: Evaluation judgments $\Psi \vdash M \hookrightarrow V : A$ and $\Psi \vdash M \Uparrow V : B$

4 Iteration

The modal operator $\Box$ introduced in Section 3 allows us to restrict iteration and case distinction to subjects of type $\Box B$, where B is a pure type. The technical realization of this idea in its full generality is rather complex. We therefore begin by describing the behavior of functions defined by iteration informally, incrementally developing their formal definition within our system. In the informal presentation we elide the box constructor, but we should convince ourselves that the subject of the iteration or case is indeed assumed to be closed.

Example 4.1 (Addition) The usual type of addition is $\mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}$. This is no longer a valid type for addition, since it must iterate over either its first or second argument and would therefore not be parametric in that argument. Among the possible types for addition, we will be interested particularly in $\Box\mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}$ and $\Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat}$.

$$
\begin{array}{lcl}
\mathrm{plus}\ \mathrm{z}\ n & = & n \\
\mathrm{plus}\ (\mathrm{s}\ m)\ n & = & \mathrm{s}\ (\mathrm{plus}\ m\ n)
\end{array}
$$

Note that this definition cannot be assigned type $\mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}$ or $\Box\mathrm{nat} \to \mathrm{nat} \to \Box\mathrm{nat}$.

In our system we view iteration as replacing constructors of a canonical term by functions of appropriate type, which is also the idea behind *catamorphisms* [FS96]. In the case of natural numbers, we replace $\mathrm{z} : \mathrm{nat}$ by a term $M_z : A$ and $\mathrm{s} : \mathrm{nat} \to \mathrm{nat}$ by a function $M_s : A \to A$. Thus iteration over natural numbers replaces type nat by A. We use the notation $a \mapsto A$ for a *type replacement* and $c \mapsto M$ for a *term replacement*. Iteration in its simplest form is written as "it $\langle a \mapsto A\rangle\ M\ \langle\Omega\rangle$" where M is the subject of the iteration, and Ω is a list containing term replacements for all constructors of type a. The formal typing rules for replacements are given later in this section; first some examples.

Example 4.2 (Addition via iteration) Addition from Example 4.1 can be formulated in a number of ways with an explicit iteration operator. The simplest one:

$$
\begin{array}{lcl}
\mathrm{plus}' & : & \Box\mathrm{nat} \to \mathrm{nat} \to \mathrm{nat} \\
 & = & \lambda m{:}\Box\mathrm{nat}.\, \lambda n{:}\mathrm{nat}.\, \mathrm{it}\ \langle \mathrm{nat} \mapsto \mathrm{nat}\rangle\ m\ \langle \mathrm{z} \mapsto n \mid \mathrm{s} \mapsto \mathrm{s}\rangle
\end{array}
$$

Later examples require addition with a result guaranteed to be closed. Its definition is only slightly more complicated.

$$
\begin{array}{lcl}
\mathrm{plus} & : & \Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat} \\
 & = & \lambda m{:}\Box\mathrm{nat}.\, \lambda n{:}\Box\mathrm{nat}.\, \mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat}\rangle\ m \\
 & & \quad \langle\, \mathrm{z} \mapsto n \\
 & & \quad \mid \mathrm{s} \mapsto (\lambda r{:}\Box\mathrm{nat}.\, \mathrm{let\ box}\ r' = r\ \mathrm{in\ box}\ (\mathrm{s}\ r'))\rangle
\end{array}
$$

If the data type is higher-order, iteration over closed objects must traverse terms with free variables. We model this in the informal presentation by introducing new parameters (written as $\nu x.M$) and extending the function definition dynamically to encompass the new parameters (written as "where $f(x) = M$").

Example 4.3 (Counting variable occurrences) Below is a function which counts the number of occurrences of bound variables in an untyped λ-expression in the representation of Example 2.5. It can be assigned type $\Box\mathrm{exp} \to \Box\mathrm{nat}$.

$$\begin{array}{lcl} \mathrm{cntvar}\ (\mathrm{app}\ e_1\ e_2) & = & \mathrm{plus}\ (\mathrm{cntvar}\ e_1)\ (\mathrm{cntvar}\ e_2) \\ \mathrm{cntvar}\ (\mathrm{lam}\ e) & = & \nu x.\, \mathrm{cntvar}\ (e\ x)\ \mathrm{where}\ \mathrm{cntvar}\ x = (\mathrm{s}\ \mathrm{z}) \end{array}$$

It may look like the recursive call in the example above is not well-typed since $(e\ x)$ is not closed as required, but contains a free parameter x. Making sense of this apparent contradiction is the principal difficulty in designing an iteration construct for higher-order abstract syntax. As before, we model iteration via replacements. Here, $\mathrm{exp} \mapsto \Box\mathrm{nat}$ and so $\mathrm{lam} \mapsto M_1$ and $\mathrm{app} \mapsto M_2$ where $M_1 : (\Box\mathrm{nat} \to \Box\mathrm{nat}) \to \Box\mathrm{nat}$ and $M_2 : \Box\mathrm{nat} \to (\Box\mathrm{nat} \to \Box\mathrm{nat})$. The types of replacement terms M_1 and M_2 arise from the types of the constructors $\mathrm{lam} : (\mathrm{exp} \to \mathrm{exp}) \to \mathrm{exp}$ and $\mathrm{app} : \mathrm{exp} \to (\mathrm{exp} \to \mathrm{exp})$ by applying the type replacement $\mathrm{exp} \mapsto \Box\mathrm{nat}$. We write

$$\begin{array}{lcl} \mathrm{cntvar} & : & \Box\mathrm{exp} \to \Box\mathrm{nat} \\ & = & \lambda x{:}\Box\mathrm{exp}.\, \mathrm{it}\ \langle \mathrm{exp} \mapsto \Box\mathrm{nat}\rangle\ x \\ & & \quad \langle\ \mathrm{app} \mapsto \mathrm{plus} \\ & & \quad \mid \mathrm{lam} \mapsto \lambda f{:}\Box\mathrm{nat} \to \Box\mathrm{nat}.\, f\ (\mathrm{box}\ (\mathrm{s}\ \mathrm{z}))\rangle \end{array}$$

For example, after β-reduction and replacement the term

$$\mathrm{cntvar}\ (\mathrm{box}\ (\mathrm{lam}\ (\lambda x{:}\mathrm{exp}.\, \mathrm{app}\ x\ x)))$$

reduces to

$$(\lambda f{:}\Box\mathrm{nat} \to \Box\mathrm{nat}.\, f\ (\mathrm{box}\ (\mathrm{s}\ \mathrm{z})))\ (\lambda n{:}\Box\mathrm{nat}.\, \mathrm{plus}\ n\ n)$$

which can in turn be β-reduced to plus (box (s z)) (box (s z)) and finally to the expected answer box (s (s z)).

Note that our operational semantics (see Definition 4.29) goes through different intermediate steps than the sequence above, but leads to the same result. Note also how replacement re-types (and possibly renames) bound variables (from $x : \mathrm{exp}$ to $n : \Box\mathrm{nat}$) in the canonical form to guarantee type preservation.

Example 4.4 (Counting abstractions) The function below counts the number of occurrences of λ-abstractions in an expression. It also has type $\Box\mathrm{exp} \to \Box\mathrm{nat}$.

$$\begin{array}{lcl} \mathrm{cntlam}\ (\mathrm{app}\ e_1\ e_2) & = & \mathrm{plus}\ (\mathrm{cntlam}\ e_1)\ (\mathrm{cntlam}\ e_2) \\ \mathrm{cntlam}\ (\mathrm{lam}\ e) & = & \mathrm{s}\ (\nu x.\, \mathrm{cntlam}\ (e\ x)\ \mathrm{where}\ \mathrm{cntlam}\ x = \mathrm{z}) \end{array}$$

Its representation as an iteration follows the same ideas as above.

$$\begin{array}{lcl} \mathrm{cntlam} & : & \Box\mathrm{exp} \to \Box\mathrm{nat} \\ & = & \lambda x{:}\Box\mathrm{exp}.\, \mathrm{it}\ \langle \mathrm{exp} \mapsto \Box\mathrm{nat}\rangle\ x \\ & & \quad \langle\ \mathrm{app} \mapsto \lambda n_1{:}\Box\mathrm{nat}.\, \lambda n_2{:}\Box\mathrm{nat}.\, \mathrm{plus}\ n_1\ n_2 \\ & & \quad \mid \mathrm{lam} \mapsto \lambda f{:}\Box\mathrm{nat} \to \Box\mathrm{nat}.\, \mathrm{let}\ \mathrm{box}\ m = f\ (\mathrm{box}\ \mathrm{z})\ \mathrm{in}\ \mathrm{box}\ (\mathrm{s}\ m)\rangle \end{array}$$

Example 4.5 (First order logic) First order formulas and terms are represented as canonical objects of type i over the signature which includes the following declarations.

$$\begin{array}{llcl} \text{Terms:} & t & & \\ \text{Formulas:} & F & ::= & \forall x.\, F \mid F_1 \supset F_2 \mid t_1 = t_2 \end{array}$$

$$\begin{array}{rcl|ll} & & & \mathrm{i} & : \mathrm{type} \\ & & & \mathrm{o} & : \mathrm{type} \\ \ulcorner \forall x.\, F \urcorner & = & \mathrm{forall}\ (\lambda x{:}\mathrm{i}.\, \ulcorner F \urcorner) & \mathrm{forall} & : (\mathrm{i} \to \mathrm{o}) \to \mathrm{o} \\ \ulcorner F_1 \supset F_2 \urcorner & = & \mathrm{impl}\ \ulcorner F_1 \urcorner\, \ulcorner F_2 \urcorner & \mathrm{impl} & : \mathrm{o} \to \mathrm{o} \to \mathrm{o} \\ \ulcorner t_1 = t_2 \urcorner & = & \mathrm{eq}\ \ulcorner t_1 \urcorner\, \ulcorner t_2 \urcorner & \mathrm{eq} & : \mathrm{i} \to \mathrm{i} \to \mathrm{o} \end{array}$$

To count the number of equality tests, we can specify cnteq with type $\mathrm{i} \to \Box\mathrm{o} \to \Box\mathrm{nat}$ as follows. We require an argument term t in order to instantiate the universal quantifier (since we did not assume any constants of type i).

$$\begin{array}{lcl} \mathrm{cnteq}\ t\ (\mathrm{forall}\ F) & = & \mathrm{cnteq}\ t\ (F\ t) \\ \mathrm{cnteq}\ t\ (\mathrm{impl}\ F_1\ F_2) & = & \mathrm{plus}\ (\mathrm{cnteq}\ t\ F_1)\ (\mathrm{cnteq}\ t\ F_2) \\ \mathrm{cnteq}\ t\ (\mathrm{eq}\ t_1\ t_2) & = & \mathrm{box}\ (\mathrm{s}\ \mathrm{z}) \end{array}$$

A representation of cnteq in the modal λ-calculus has the form:

$$\begin{array}{lcl} \mathrm{cnteq} & : & \mathrm{i} \to \Box\mathrm{o} \to \Box\mathrm{nat} \\ & = & \lambda t{:}\mathrm{i}.\, \lambda F{:}\Box\mathrm{o}.\, \mathrm{it}\ \langle \mathrm{o} \mapsto \Box\mathrm{nat} \rangle\ F \\ & & \quad \langle\, \mathrm{forall} \mapsto \lambda f{:}\mathrm{i} \to \Box\mathrm{nat}.\, (f\ t) \\ & & \quad \mid \mathrm{impl} \mapsto \mathrm{plus} \\ & & \quad \mid \mathrm{eq} \mapsto \lambda t_1{:}\mathrm{i}.\, \lambda t_2{:}\mathrm{i}.\, \mathrm{box}\ (\mathrm{s}\ \mathrm{z}) \rangle \end{array}$$

Example 4.6 (Booleans) Boolean values can be represented as objects of type bool over the signature which includes the following declaration:

$$\text{Boolean Values:} \quad b \quad ::= \quad \top \mid \bot$$

$$\begin{array}{rcl|ll} & & & \mathrm{bool} & : \mathrm{type} \\ \ulcorner \top \urcorner & = & \mathrm{true} & \mathrm{true} & : \mathrm{bool} \\ \ulcorner \bot \urcorner & = & \mathrm{false} & \mathrm{false} & : \mathrm{bool} \end{array}$$

Informally we can represent the Boolean operation and as follows. We must require all argument and all result types are boxed, because the result of and will be used as subject for another case distinction.

$$\begin{array}{lcl} \mathrm{and}\ \mathrm{true}\ B_2 & = & B_2 \\ \mathrm{and}\ \mathrm{false}\ B_2 & = & \mathrm{false} \end{array}$$

A formal representation of and is then as follows:

$$\begin{array}{lcl} \mathrm{and} & : & \Box\mathrm{bool} \to \Box\mathrm{bool} \to \Box\mathrm{bool} \\ & = & \lambda B_1{:}\Box\mathrm{bool}.\, \lambda B_2{:}\Box\mathrm{bool}. \\ & & \quad \mathrm{it}\ \langle \mathrm{bool} \mapsto \Box\mathrm{bool} \rangle\ B_1 \\ & & \quad \langle\, \mathrm{true} \mapsto B_2 \\ & & \quad \mid \mathrm{false} \mapsto \mathrm{box}\ \mathrm{false} \rangle \end{array}$$

Example 4.7 (Constant test) Below we define a function which returns true if a given functional object of type exp $\to$ exp (see Example 2.5) is constant with respect to the first argument.

$$\begin{array}{lcl}
\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,(\mathrm{lam}\ (\lambda y\mathbin{:}\mathrm{exp}.\,E\ x\ y)) & = & \nu y.\,\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,(E\ x\ y) \\
& & \qquad \text{where const } \lambda x\mathbin{:}\mathrm{exp}.\,y = \mathrm{true} \\
\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,(\mathrm{app}\ (E_1\ x)\ (E_2\ x)) & = & \mathrm{and}\ (\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,(E_1\ x))\ (\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,(E_2\ x)) \\
\mathrm{const}\ \lambda x\mathbin{:}\mathrm{exp}.\,x & = & \mathrm{false}
\end{array}$$

The representation of const has type $\Box(\mathrm{exp} \to \mathrm{exp}) \to \Box\mathrm{bool}$.

$$\begin{array}{rcl}
\mathrm{const} & : & \Box(\mathrm{exp} \to \mathrm{exp}) \to \Box\mathrm{bool} \\
& = & \lambda F\mathbin{:}\Box(\mathrm{exp} \to \mathrm{exp}).\ \ \mathrm{it}\ \langle \mathrm{exp} \mapsto \Box\mathrm{bool}\rangle\ F \\
& & \qquad\qquad\qquad\qquad \langle\, \mathrm{lam} \mapsto \lambda E\mathbin{:}\Box\mathrm{bool} \to \Box\mathrm{bool}.\,(E\ (\mathrm{box\ true})) \\
& & \qquad\qquad\qquad\qquad \mid \mathrm{app} \mapsto \mathrm{and}\rangle\ (\mathrm{box\ false})
\end{array}$$

Note how the last case in the informal definition is represented by applying the result of iteration (which will be of type $\Box\mathrm{bool} \to \Box\mathrm{bool}$) to box false.

Example 4.8 (Translation to de Bruijn representation) Untyped λ-expressions in de Bruijn form are represented as canonical objects of type db over the signature which includes the natural numbers and the following declarations.

$$\text{DeBruijn expressions:} \quad d \quad ::= \quad n \mid \mathbf{lam}\ d \mid d_1 @ d_2$$

$$\begin{array}{rcl@{\qquad}l}
 & & & \mathrm{db}\ :\mathrm{type} \\
\ulcorner n \urcorner & = & \mathrm{var}\ \ulcorner n \urcorner & \mathrm{var} : \mathrm{nat} \to \mathrm{db} \\
\ulcorner \mathbf{lam}\ d \urcorner & = & \mathrm{lm}\ \ulcorner d \urcorner & \mathrm{lm}\ : \mathrm{db} \to \mathrm{db} \\
\ulcorner d_1 @ d_2 \urcorner & = & \mathrm{ap}\ \ulcorner d_1 \urcorner\ \ulcorner d_2 \urcorner & \mathrm{ap}\ : \mathrm{db} \to \mathrm{db} \to \mathrm{db}
\end{array}$$

A translation from the higher-order representation to de Bruijn form has type $\Box\mathrm{exp} \to \mathrm{db}$ and is represented formally in terms of an auxiliary function trans of type $\Box\mathrm{exp} \to \Box\mathrm{nat} \to \mathrm{db}$:

$$\begin{array}{lcl}
\mathrm{trans}\ (\mathrm{lam}\ e)\ n & = & \mathrm{lm}\ (\nu x.\,\mathrm{trans}\ (e\ x)\ (\mathrm{s}\ n)) \\
& & \qquad \text{where } (\mathrm{trans}\ x\ m) = \mathrm{var}\ (\mathrm{minus}\ m\ n)) \\
\mathrm{trans}\ (\mathrm{app}\ e_1\ e_2)\ n & = & \mathrm{ap}\ (\mathrm{trans}\ e_1\ n)\ (\mathrm{trans}\ e_2\ n) \\
\mathrm{dbtrans}\ e & = & \mathrm{trans}\ e\ \mathrm{z}
\end{array}$$

At the top level (when translating a closed term) we can apply this to any natural number to obtain a function of type $\Box\mathrm{exp} \to \mathrm{db}$. Assuming functions minus (whose definition we discuss in the next section) and unbox (see Example 3.2), this is implemented by the following iteration.

$$\begin{array}{rcl}
\mathrm{trans} & : & \Box\mathrm{exp} \to \Box\mathrm{nat} \to \mathrm{db} \\
& = & \lambda x\mathbin{:}\Box\mathrm{exp}.\,\mathrm{it}\ \langle \mathrm{exp} \mapsto \Box\mathrm{nat} \to \mathrm{db}\rangle\ x \\
& & \quad \langle\, \mathrm{lam} \mapsto \lambda f\mathbin{:}(\Box\mathrm{nat} \to \mathrm{db}) \to (\Box\mathrm{nat} \to \mathrm{db}). \\
& & \qquad\qquad \lambda n\mathbin{:}\Box\mathrm{nat}.\,\mathrm{lm}\ (f\ (\lambda m\mathbin{:}\Box\mathrm{nat}.\,\mathrm{var}\ (\mathrm{unbox}\ (\mathrm{minus}\ m\ n))) \\
& & \qquad\qquad\qquad\qquad (\mathrm{let\ box}\ n' = n\ \mathrm{in\ box}\ (\mathrm{s}\ n'))) \\
& & \quad \mid \mathrm{app} \mapsto \lambda f_1\mathbin{:}\Box\mathrm{nat} \to \mathrm{db}.\,\lambda f_2\mathbin{:}\Box\mathrm{nat} \to \mathrm{db}. \\
& & \qquad\qquad \lambda n\mathbin{:}\Box\mathrm{nat}.\,\mathrm{ap}\ (f_1\ n)\ (f_2\ n)\rangle \\
\mathrm{dbtrans} & : & \Box\mathrm{exp} \to \mathrm{db} \\
& = & \lambda x\mathbin{:}\Box\mathrm{exp}.\,\mathrm{trans}\ x\ (\mathrm{box\ z})
\end{array}$$

A number of other functions can be defined elegantly in this representation. For example, pairs prove appear to be necessary for defining parallel β-reduction (which is convenient in the proof of the Church-Rosser theorem).

Example 4.9 (Parallel reduction) Parallel reduction is here defined over expressions (from Example 2.5). We state the function first informally:

$$\begin{array}{lcl}
\mathrm{par}\ (\mathrm{app}\ e_1\ e_2) & = & \mathrm{par}'\ e_1\ (\mathrm{par}\ e_2)\\
\mathrm{par}\ (\mathrm{lam}\ e_1) & = & \mathrm{lam}\ (\lambda x\!:\!\mathrm{exp}.\ \mathrm{par}\ (e_1\ x))\\
 & & \qquad\qquad \text{where } \mathrm{par}\ x = x \text{ and } \mathrm{par}'\ x\ e_3 = \mathrm{app}\ x\ e_3\\
\mathrm{par}'\ (\mathrm{app}\ e_1\ e_2)\ e_2' & = & \mathrm{app}\ (\mathrm{par}'\ e_1\ (\mathrm{par}\ e_2))\ e_2'\\
\mathrm{par}'\ (\mathrm{lam}\ e_1)\ e_2' & = & \nu x.\ \mathrm{par}\ (e_1\ x)\\
 & & \qquad\qquad \text{where } \mathrm{par}\ x\ =\ e_2' \text{ and } \mathrm{par}'\ x\ e_3 = \mathrm{app}\ e_2'\ e_3
\end{array}$$

Parallel reduction can be represented in our system — pairs are essential to do so. The type of par is $\Box\mathrm{exp} \to \mathrm{exp}$. The (unnamed) intermediate function defined by iteration has type $\Box\mathrm{exp} \to \mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp})$.

$$\begin{array}{lcl}
\mathrm{par} & : & \Box\mathrm{exp} \to \mathrm{exp}\\
 & = & \lambda e\!:\!\Box\mathrm{exp}.\\
 & & \mathrm{fst}(\ \mathrm{it}\ \langle \mathrm{exp} \mapsto \mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp})\rangle\ e\\
 & & \quad \langle\ \mathrm{app} \mapsto\ \ \lambda e_1\!:\!\mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp}).\ \lambda e_2\!:\!\mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp}).\\
 & & \qquad\qquad\qquad \langle\ (\mathrm{snd}\ e_1)\ (\mathrm{fst}\ e_2),\\
 & & \qquad\qquad\qquad\ \ \lambda e_2'\!:\!\mathrm{exp}.\ \mathrm{app}\ ((\mathrm{snd}\ e_1)\ (\mathrm{fst}\ e_2))\ e_2'\rangle\\
 & & \quad \mid \mathrm{lam} \mapsto\ \ \lambda e_1\!:\!(\mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp})) \to (\mathrm{exp} \times (\mathrm{exp} \to \mathrm{exp})).\\
 & & \qquad\qquad\qquad \langle \mathrm{lam}\ (\lambda x\!:\!\mathrm{exp}.\ \mathrm{fst}\ (e_1\ \langle x, \lambda e_3\!:\!\mathrm{exp}.\ \mathrm{app}\ x\ e_3\rangle)),\\
 & & \qquad\qquad\qquad\ \ \lambda e_2'\!:\!\mathrm{exp}.\ \mathrm{fst}\ (e_1\ \langle e_2', \lambda e_3\!:\!\mathrm{exp}.\ \mathrm{app}\ e_2'\ e_3\rangle)\rangle\rangle)
\end{array}$$

The following example illustrates two concepts: mutually inductive types and iteration over the form of a (parametric!) function (which we already saw in Example 4.7).

Example 4.10 (Substitution in normal forms) Substitution is already directly definable by application, but one may also ask if there is a structural definition in the style of [Mil91]. Normal forms of the untyped λ-calculus are represented by the type nf with an auxiliary definition for atomic forms of type at.

$$\begin{array}{llcl}
\text{Normal forms :} & N & ::= & P \mid \mathbf{lam}\ x.N\\
\text{Atomic forms:} & P & ::= & x \mid P@N
\end{array}$$

The representation function $\ulcorner.\urcorner$ is now overloaded, but it should be clear how to resolve it from the context.

$$\begin{array}{rcll}
 & & & \mathrm{nf}\ \ \ :\mathrm{type}\\
 & & & \mathrm{at}\ \ \ :\mathrm{type}\\
\ulcorner P\urcorner & = & \mathrm{atnf}\,\ulcorner P\urcorner & \mathrm{atnf} : \mathrm{at} \to \mathrm{nf}\\
\ulcorner \mathbf{lam}\ x.N\urcorner & = & \mathrm{l}\ (\lambda x\!:\!\mathrm{at}.\ulcorner N\urcorner) & \mathrm{l}\ \ \ \ \ : (\mathrm{at} \to \mathrm{nf}) \to \mathrm{nf}\\
\ulcorner P@N\urcorner & = & \mathrm{a}\,\ulcorner P\urcorner\,\ulcorner N\urcorner & \mathrm{a}\ \ \ \ : \mathrm{at} \to \mathrm{nf} \to \mathrm{at}\\
\ulcorner x\urcorner & = & x &
\end{array}$$

Substitution of atomic objects for variables is defined by two mutually recursive functions, one with type subnf : $\Box(\mathrm{at} \to \mathrm{nf}) \to \mathrm{at} \to \mathrm{nf}$ and subat : $\Box(\mathrm{at} \to \mathrm{at}) \to \mathrm{at} \to \mathrm{at}$.

$$\begin{array}{lcl}
\mathrm{subnf}\ (\lambda x\!:\!\mathrm{at}.\, \mathrm{l}\ (\lambda y\!:\!\mathrm{at}.\, N\ x\ y))\ Q & = & \mathrm{l}\ (\lambda y\!:\!\mathrm{at}.\, \mathrm{subnf}\ (\lambda x\!:\!\mathrm{at}.\, (N\ x\ y))\ Q \\
 & & \quad \text{where subat } (\lambda x\!:\!\mathrm{at}.\, y)\ Q = y) \\
\mathrm{subnf}\ (\lambda x\!:\!\mathrm{at}.\, \mathrm{atnf}\ (P\ x))\ Q & = & \mathrm{atnf}\ (\mathrm{subat}\ (\lambda x\!:\!\mathrm{at}.\, P\ x)\ Q) \\
\mathrm{subat}\ (\lambda x\!:\!\mathrm{at}.\, \mathrm{a}\ (P\ x)\ (N\ x))\ Q & = & \mathrm{a}\ (\mathrm{subat}\ (\lambda x\!:\!\mathrm{at}.\, P\ x)\ Q)\ (\mathrm{subnf}\ (\lambda x\!:\!\mathrm{at}.\, N\ x)\ Q) \\
\mathrm{subat}\ (\lambda x\!:\!\mathrm{at}.\, x)\ Q & = & Q
\end{array}$$

The last case arises since the parameter x must be considered as a new constructor in the body of the abstraction. The functions above are realized in our calculus by a simultaneous replacement of objects of type nf and at. In other words, the type replacement must account for all mutually recursive types, and the term replacement for all constructors of those types.

$$\begin{array}{lcl}
\mathrm{subnf} & : & \Box(\mathrm{at} \to \mathrm{nf}) \to \mathrm{at} \to \mathrm{nf} \\
 & = & \lambda N\!:\!\Box(\mathrm{at} \to \mathrm{nf}).\, \lambda Q\!:\!\mathrm{at}.\, \mathrm{it}\ \langle \mathrm{nf} \mapsto \mathrm{nf} \mid \mathrm{at} \mapsto \mathrm{at} \rangle\ N \\
 & & \qquad \langle\ \mathrm{l} \mapsto \lambda F\!:\!\mathrm{at} \to \mathrm{nf}.\, \mathrm{l}\ (\lambda y\!:\!\mathrm{at}.\, (F\ y)) \\
 & & \qquad \mid \mathrm{atnf} \mapsto \lambda P\!:\!\mathrm{at}.\, \mathrm{atnf}\ P \\
 & & \qquad \mid \mathrm{a} \mapsto \lambda P\!:\!\mathrm{at}.\, \lambda N\!:\!\mathrm{nf}.\, \mathrm{a}\ P\ N \rangle \\
 & & \qquad Q
\end{array}$$

Via η-contraction we can see that substitution amounts to a structural identity function.

Example 4.11 (Further mathematical operations) Below we define the multiplication and the exponentiation function which we can informally define as follows:

$$\begin{array}{lcl}
\mathrm{mult}\ \mathrm{z}\ N & = & \mathrm{z} \\
\mathrm{mult}\ (\mathrm{s}\ M)\ N & = & \mathrm{plus}\ (\mathrm{mult}\ M\ N)\ N
\end{array}$$

$$\begin{array}{lcl}
\mathrm{ex}\ M\ \mathrm{z} & = & \mathrm{s}\ \mathrm{z} \\
\mathrm{ex}\ M\ (\mathrm{s}\ N) & = & \mathrm{mult}\ (\mathrm{ex}\ M\ N)\ M
\end{array}$$

The representation of mult and ex has type $\Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat}$.

$$\begin{array}{lcl}
\mathrm{mult} & : & \Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat} \\
 & = & \lambda M\!:\!\Box\mathrm{nat}.\, \lambda N\!:\!\Box\mathrm{nat}.\ \ \mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat} \rangle\ M \\
 & & \qquad \langle\ \mathrm{z} \mapsto \mathrm{box}\ \mathrm{z} \\
 & & \qquad \mid \mathrm{s} \mapsto \lambda M'\!:\!\Box\mathrm{nat}.\, (\mathrm{plus}\ M'\ N) \rangle
\end{array}$$

$$\begin{array}{lcl}
\mathrm{ex} & : & \Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat} \\
 & = & \lambda M\!:\!\Box\mathrm{nat}.\, \lambda N\!:\!\Box\mathrm{nat}.\ \ \mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat} \rangle\ N \\
 & & \qquad \langle\ \mathrm{z} \mapsto \mathrm{box}\ (\mathrm{s}\ \mathrm{z}) \\
 & & \qquad \mid \mathrm{s} \mapsto \lambda N'\!:\!\Box\mathrm{nat}.\, (\mathrm{mult}\ M\ N') \rangle
\end{array}$$

Example 4.12 (Ackermann's function) Below we define the function which we can informally define as follows:

$$\begin{array}{lcl}
\mathrm{A}\ \mathrm{z} & = & \lambda x\!:\!\mathrm{nat}.\, (\mathrm{s}\ x) \\
\mathrm{A}\ (\mathrm{s}\ n) & = & \lambda x\!:\!\mathrm{nat}.\, (\mathrm{A}^x\ n)\ x
\end{array}$$

where $(\mathrm{A}^x\ n)$ stands for $\underbrace{(\mathrm{A}...(\mathrm{A}}_{x-\text{times}}\ n))$. The representation of A has type $\Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat}$.

$$
\begin{array}{lll}
\mathrm{A} & : & \Box\mathrm{nat} \to \Box\mathrm{nat} \to \Box\mathrm{nat} \\
 & = & \lambda m{:}\Box\mathrm{exp}.\ \mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat} \to \mathrm{nat}\rangle\ m \\
 & & \qquad \langle\, \mathrm{z} \mapsto \lambda x{:}\Box\mathrm{nat}.\,\mathrm{let\ box}\ x' = x\ \mathrm{in\ box}\ (\mathrm{s}\ x') \\
 & & \qquad \mid \mathrm{s} \mapsto \lambda f{:}\Box\mathrm{nat} \to \Box\mathrm{nat}.\ \lambda x{:}\Box\mathrm{nat}.\,\mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat}\rangle\ x\ \langle \mathrm{z} \mapsto x, \mathrm{s} \mapsto f\rangle\rangle
\end{array}
$$

The following example shows a scheme how to represent primitive recursion over natural numbers using pairs.

Example 4.13 (Primitive recursion over natural numbers) Below we define a general primitive recursive scheme over natural numbers. Let A be the result type of the primitive recursion. For every $N_z : A$ and $N_s : \Box\mathrm{nat} \to A \to A$ we define informally the primitive recursion scheme:

$$
\begin{array}{lll}
\mathrm{pr}\ \mathrm{z} & = & N_z \\
\mathrm{pr}\ (\mathrm{s}\ m')\ N & = & N_s\ m'\ (\mathrm{pr}\ m')
\end{array}
$$

The representation of pr must make use of pairs. The reason for this is that the natural number must be passed as an argument to N_s. Using a standard iteration scheme would not be enough, because this information is discarded. Pairs allow to recover the structure of this natural number:

$$
\begin{array}{lll}
\mathrm{pr} & : & \Box\mathrm{nat} \to A \\
 & = & \lambda m{:}\Box\mathrm{nat}.\,\mathrm{snd} \\
 & & \quad (\mathrm{it}\ \langle \mathrm{nat} \mapsto \Box\mathrm{nat} \times A\rangle\ m \\
 & & \quad \langle\, \mathrm{z} \mapsto \langle \mathrm{box\ z}, N_z\rangle \\
 & & \quad \mid \mathrm{s} \mapsto \lambda p{:}\Box\mathrm{nat} \times A.\,\langle \mathrm{box}\ (\mathrm{let\ box}\ m' = (\mathrm{fst}\ p)\ \mathrm{in\ s}\ m'), N_s\ (\mathrm{fst}\ p)\ (\mathrm{snd}\ p)\rangle\rangle)
\end{array}
$$

We begin now with the formal discussion and description of the full language. Due to the possibility of mutual recursion among types, the type replacements must be lists (see Example 4.10).

$$\text{Type replacement:} \quad \omega \quad ::= \quad \cdot \mid (\omega \mid a \mapsto A)$$

The types being replaced form a type domain, i.e. a list of type constants.

$$\text{Type domain:} \quad \alpha \quad ::= \quad \cdot \mid \alpha, a$$

Which types must be replaced by an iteration depends on which types are mutually recursive according to the constructors in the signature Σ and possibly the type of the iteration subject itself. If we iterate over a function, the parameter of a function must be treated like a constructor for its type, since it can appear in that role in the body of a function.

The types which must be replaced by an iteration form a type domain. The type replacement must be defined in such a way that every type in this domain is replaced by some other type. This leads to the introduction of *well-formed* type replacements $\vdash \omega : \alpha$.

Definition 4.14 (Well formed type replacements)
Rules:

$$\frac{}{\vdash \cdot : \cdot}\,\mathsf{WrBase} \qquad \frac{\vdash \omega : \alpha}{\vdash (\omega \mid a \mapsto A) : (\alpha, a)}\,\mathsf{WrInd}$$

We address now the question of mutual dependency between atomic types by defining the notion of *type subordination* which summarizes all dependencies between atomic types by separately considering its *static* part $\lhd_\Sigma$ which derives from the dependencies induced by the constructor types from Σ and its *dynamic* part $\lhd_B$ which accounts for dependencies induced from the argument types of B. We say that type a_1 subordinates type a_2 if objects of the later type can be constructed from objects of the former type.

But what does it mean, to build up objects of some pure type B? In the easier case where iteration is performed over objects of atomic type a objects can only be constructed from constructors with target type a, applied to some objects of the argument types. In the more complicated case, they can be also formed from variables introduced by λ-abstractions. These so called "parameters" or "pseudo constructors" must be of target type a. We denote the *target type* of a pure type B by $\tau(B)$.

Definition 4.15 (Target types)

$$\begin{aligned} \tau(a) &:= a \\ \tau(B_1 \to B_2) &:= \tau(B_2) \end{aligned}$$

Let B be the type of a (pseudo) constructor and M be an object of this type B. From which other objects M can be constructed can be directly extracted from the type B, namely all objects of the argument types of B — regardless if they occur positively or negatively. For a given pure type B we define the type domain $\mathrm{Source}(B)$ as

Definition 4.16 (Source types)

$$\begin{aligned} \mathit{Source}(a) &:= \cdot \\ \mathit{Source}(B_1 \to B_2) &:= (\mathit{Source}(B_1), \tau(B_1)) \cup \mathit{Source}(B_2) \end{aligned}$$

Note, that the resulting type domain can only contain atomic types. For example 4.10 it is easily verified that the constructor type of a yields: $\mathrm{Source}(\mathrm{at} \to \mathrm{nf} \to \mathrm{at}) = (\mathrm{at}, \mathrm{nf})$.

To obtain a set of all types on which an atomic type a may depend, we must select a subset of the signature Σ containing all constant declaration with target type a. This set is called a set of constructor types for a and denoted by $\mathcal{S}(\Sigma; a)$:

Definition 4.17 (Constructor types)

$$\begin{aligned} \mathcal{S}(\cdot; a) &= \cdot \\ \mathcal{S}(\Sigma', x : B; a) &= \begin{cases} \mathcal{S}(\Sigma'; a), x : B & \mathit{if}\ \tau(B) = a \\ \mathcal{S}(\Sigma'; a) & \mathit{otherwise} \end{cases} \end{aligned}$$

In a more general setting we need the set of all constructor declarations of a mutual inductive type. Type domains have been introduced to represent the set of all participating atomic types for mutual inductive datatypes. The definition of the set of constructors $\mathcal{S}^*(\Sigma; \alpha)$ follows easily:

Definition 4.18 (Constructor types over type domains)

$$\begin{aligned} \mathcal{S}^*(\Sigma; \cdot) &= \cdot \\ \mathcal{S}^*(\Sigma; \alpha, a) &= \mathcal{S}^*(\Sigma; \alpha) \cup \mathcal{S}(\Sigma; a) \end{aligned}$$

The intuition behind this construction is to define the subordination relation which reflects the dependencies of atomic types in between each other. Target types and source types help us to capture the dependencies for one pure type: the target type *depends* on each source type. These dependencies are captured by the immediate subordination relation:

Definition 4.19 (Immediate subordination relation) *Let B a type.*

$$a <_B \tau(B) \textit{ iff } a \in \textit{Source}(B)$$

If we take the union of all immediate subordination relations which are induced by a (sub)signature Σ, we obtain the static subordination relation. It is called static because the signature is fixed.

Definition 4.20 (Static subordination relation) *Let Σ be a signature.*

$$a_1 \lhd_\Sigma a_2 \textit{ iff } \Sigma = \Sigma', c : B \textit{ and either } a_1 <_B a_2 \textit{ or } a_1 \lhd_{\Sigma'} a_2$$

If we consider Example 4.10 again we see that the static subordination relation of the signature is the union of all immediate subordination relations:

- $\mathrm{a} : \mathrm{at} \to \mathrm{nf} \to \mathrm{at}$:
$$\mathrm{at} <_{\mathrm{at}\to\mathrm{nf}\to\mathrm{at}} \mathrm{at} \quad \text{and} \quad \mathrm{nf} <_{\mathrm{at}\to\mathrm{nf}\to\mathrm{at}} \mathrm{at}$$
- $\mathrm{atnf} : \mathrm{at} \to \mathrm{nf}$:
$$\mathrm{at} <_{\mathrm{at}\to\mathrm{nf}} \mathrm{nf}$$
- $\mathrm{l} : (\mathrm{at} \to \mathrm{nf}) \to \mathrm{nf}$:
$$\mathrm{at} <_{\mathrm{at}\to\mathrm{nf}\to\mathrm{nf}} \mathrm{nf} \quad \text{and} \quad \mathrm{nf} <_{\mathrm{at}\to\mathrm{nf}\to\mathrm{nf}} \mathrm{nf}$$

Putting it all together we obtain the static subordination relation:

$$\mathrm{at} \lhd_\Sigma \mathrm{at}, \mathrm{nf} \lhd_\Sigma \mathrm{at}, \mathrm{at} \lhd_\Sigma \mathrm{nf}, \mathrm{nf} \lhd_\Sigma \mathrm{nf}$$

Note that the static subordination relation need not to be transitive or reflexive.

Constructor types are not the only source of subordination. As briefly mentioned above, another source are types introduced by the type of the iteration subject. Fortunately, it is always closed which guarantees that no free variables can be encountered while traversing its structure except internal variables defined in the body of the subject. Iteration over functional objects can introduce new dependencies into the subordination graph as the following example shows.

Example 4.21 (Higher-order logic) First order logic can be easily extended to higher order logic by introducing a reification function from formulas to terms. To count the number of equality tests, we extend the subject of iteration defined in Example 4.5 by a new abstraction over the reification function r which has type $\mathrm{o} \to \mathrm{i}$. The introduction of a reification function makes terms and formulas depend mutually on each other. We therefore must distinguish between cnteqi of type $\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{i}) \to \Box\mathrm{nat}$ which counts occurrences of equality tests in terms and cnteqo of type $\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \Box\mathrm{nat}$ which counts them in formulas.

$$
\begin{array}{lcl}
\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(\mathrm{forall}\ (\lambda x\!:\!\mathrm{i}.\,F\ r\ x)) & = & \nu x.\,(\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(F\ r\ x)) \\
 & & \qquad \text{where cnteqi } \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,x = \mathrm{z} \\
\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(\mathrm{impl}\ (F_1\ r)\ (F_2\ r)) & = & \mathrm{plus}\ (\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(F_1\ r))\ (\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(F_2\ r)) \\
\mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(\mathrm{eq}\ (t_1\ r)\ (t_2\ r)) & = & \mathrm{s}\ (\mathrm{plus}\ (\mathrm{cnteqi}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(t_1\ r))\ (\mathrm{cnteqi}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(t_2\ r))) \\
\mathrm{cnteqi}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(r\ (F\ r)) & = & \mathrm{cnteqo}\ \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\,(F\ r)
\end{array}
$$

The representation of cnteqo in the modal λ-calculus has the form:

$$
\begin{array}{lcl}
\mathrm{cnteqo} & : & \Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \Box\mathrm{nat} \\
 & = & \lambda F\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}).\,\mathrm{it}\ \langle \mathrm{o} \mapsto \Box\mathrm{nat}, \mathrm{i} \mapsto \Box\mathrm{nat}\rangle\ F \\
 & & \quad \langle\, \mathrm{forall} \mapsto \lambda f\!:\!\Box\mathrm{nat} \to \Box\mathrm{nat}.\,(f\ (\mathrm{box}\ \mathrm{z})) \\
 & & \quad \mid \mathrm{impl} \mapsto \mathrm{plus} \\
 & & \quad \mid \mathrm{eq} \mapsto \lambda m\!:\!\Box\mathrm{nat}.\,\lambda n\!:\!\Box\mathrm{nat}.\,\mathrm{let\ box}\ r = \mathrm{plus}\ m\ n\ \mathrm{in\ box}\ (\mathrm{s}\ r)\rangle
\end{array}
$$

It is clear that the type of the iteration subject must be taken into consideration when defining the general subordination relation. We proceed now by characterizing all those dependencies which arise from the type B of the iteration subject which will lead to the notion of *dynamic* subordination. From the example above we can see that variables occuring in the closed subject of iteration can be interpreted as constructors if we look at the object from a purely syntactical point of view. We call those variables *pseudo constructors* and correspondingly their types as *pseudo constructor types*.

A closer look reveals that not all abstractions in a closed term introduce relevant pseudo constructors: Relevant with respect to the subordination relation are only the *top-level* pseudo constructors which are introduced as variables of the argument types of B. All others are of some subtype of either a constructor type from the signature or a pseudo constructor type. Pseudo constructor types of B are hence inductively defined as follows.

Definition 4.22 (Set of pseudo constructor types)

$$
\begin{array}{rcl}
PCT(a) & = & \{\} \\[1ex]
PCT(B_1 \to B_2) & = & \{B_1\} \cup PCT(B_2)
\end{array}
$$

In the next step we define the dynamic subordination relation which can be directly determined from the set of pseudo constructor types. We follow the same idea as in the static case: every pseudo constructor type in $\mathrm{PCT}(B)$ induces a new set of dependencies. Taking all these sets together we finally arrive at the dynamic subordination relation:

Definition 4.23 (Dynamic subordination relation) *Let B be a pure type.*

$$a_1 \lhd_B a_2 :\Leftrightarrow B = B_1 \to B_2 \textit{ and either } a_1 <_{B_1} a_2 \textit{ or } a_1 \lhd_{B_2} a_2$$

Consider type $B = \Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o})$ from the previous example. The dynamic subordination relation is then characterized by $\mathrm{o} \lhd_B \mathrm{i}$ and $\mathrm{i} \lhd_B \mathrm{i}$. $\mathrm{o} \lhd_B \mathrm{i}$ says now that we have a pseudo constructor which behaves as some kind of embedding function from formulas into individuals. It should be immediately evident, that the presence of such an embedding function turns the first order logic from example 4.5 into a higher order logic. Static and dynamic subordination represent local dependencies between atomic types. To obtain the global subordination relation, the union of both must be closed under transitivity.

Definition 4.24 (Global subordination relation) *Let B be a pure type.*

$$\blacktriangleleft_{\Sigma;B} \;:\Leftrightarrow\; (\; \lhd_\Sigma \;\cup\; \lhd_B \;)^*$$

Note, that that the global subordination relation is not necessarily reflexive. On the other hand if the subordination relation is reflexive, i.e. for the atomic type $\tau(B)$, $\tau(B) \blacktriangleleft_{\Sigma;B} \tau(B)$ holds, then $\tau(B)$ is *recursive* or *inductive* with respect to B.

The notion of inductive type and subordination are very closely related. In fact, the subordination relation is defined with the purpose to extend the notion of inductive types. Note that static type subordination is built into calculi where inductive types are defined explicitly (such as the Calculus of Inductive Constructions [PM93]); here it must be recovered from the signature since we impose no ordering constraints except that a type must be declared before it is used. Our choice to recover the type subordination relation from the signature allows us to perform iteration over any functional type, without fixing the possibilities in advance.

As we have seen in example 4.21, the dynamic subordination relation implies that terms and formulas depend on each other. Hence, static subordination constitutes only part of the subordination relation. If we would follow the paradigm used in Coq we must calculate internally a syntactical definition of the new inductive type, where pseudo constructors are defined as real constructors. This has to be done on the fly because as we will see later in the typing rules, the type of the subject of iteration B must be inferred first. It is indeed possible to proceed this way, and it is also possible to show the equivalence of both formulations (which we are not going to do here). All type constants which are mutually dependent with $\tau(B)$, written $\mathcal{I}(\Sigma; B)$, form an inductive datatype.

Definition 4.25 (Inductive type) *Let B be a type and Σ a signature:*

$$\mathcal{I}(\Sigma; B) := \{a | \tau(B) \blacktriangleleft_{\Sigma;B} a \textit{ and } a \blacktriangleleft_{\Sigma;B} \tau(B)\}$$

Revisiting example 4.21 extending first order logic to higher order logic we can calculate the inductive type $\mathcal{I}(\Sigma; (\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) = \{\mathrm{o}, \mathrm{i}\}$. The set of constructors has then the following form:

$$\mathcal{S}^*(\Sigma; \mathrm{o}, \mathrm{i}) = \mathrm{forall} : (\mathrm{i} \to \mathrm{o}) \to \mathrm{o}, \mathrm{impl} : \mathrm{o} \to \mathrm{o} \to \mathrm{o}, \mathrm{eq} : \mathrm{i} \to \mathrm{i} \to \mathrm{o}$$

Let us now address the question of how the type of an iteration is formed: If the subject of iteration has type B, the iterator object has type $\langle\omega\rangle(B)$, where $\langle\omega\rangle(B)$ is defined inductively by replacing each type constant according to ω, leaving types outside the domain fixed. The replacement application might traverse over type constants not defined in ω. This becomes immediately evident when we consider Example 4.8: nat is traversed, but not defined in ω. Also in Example 4.5: i is not defined in ω. But since objects of such strictly subordinated types do not participate in the process of iteration, their types remain unchanged.

Definition 4.26 (Type replacement)

$$\begin{array}{rcl} \langle\omega\rangle(a) & := & \begin{cases} A & \textit{if } \omega(a) = A \\ a & \textit{otherwise} \end{cases} \\ \langle\omega\rangle(B_1 \to B_2) & := & \langle\omega\rangle(B_1) \to \langle\omega\rangle(B_2) \end{array}$$

A similar replacement is applied at the level of terms: the result of an iteration is an object which resembles the (canonical) subject of the iteration in structure, but object constants are replaced by other objects carrying the intended computational meaning of the different cases. Even though the subject of iteration is closed at the beginning of the replacement process, we need to deal with embedded λ-abstractions due to higher-order abstract syntax. But since such functions are parametric we can simply replace variables x of type B by new variables x' of type $\langle\omega\rangle(B)$.

$$\text{Term replacement:} \quad \Omega \quad ::= \quad \cdot \mid (\Omega \mid c \mapsto M) \mid (\Omega \mid x \mapsto x')$$

Initially the domain of a term replacement is a signature containing all constructors whose target type is in $\mathcal{I}(\Sigma; B)$. We refer to this signature as $\mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$. The form of iteration follows now quite naturally: We extend the notion of objects by

$$M ::= \ldots \mid \text{it } \langle\omega\rangle\ M\ \langle\Omega\rangle$$

and extend the typing rules for iteration. To do so we must introduce a new typing judgment for term replacements Ω: $\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma)$. Ω is well-typed if it replaces every constant of some signature Σ with some object of correct type.

Definition 4.27 (Typing judgment for iteration) *extending Definition 3.1:*

$$\frac{\Delta; \Gamma \vdash M : \Box B \qquad \vdash \omega : \alpha \qquad \Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')}{\Delta; \Gamma \vdash it\ \langle\omega\rangle\ M\ \langle\Omega\rangle : \langle\omega\rangle(B)}\ \mathsf{TpIt}$$

$$\textit{where } \alpha = \mathcal{I}(\Sigma; B) \textit{ and } \Sigma' = \mathcal{S}^*(\Sigma; \alpha)$$

$$\frac{}{\Delta; \Gamma \vdash \cdot : \langle\omega\rangle(\cdot)}\ \mathsf{TrBase} \qquad \frac{\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma) \qquad \Delta; \Gamma \vdash M : \langle\omega\rangle(B')}{\Delta; \Gamma \vdash (\Omega \mid c \mapsto M) : \langle\omega\rangle(\Sigma, c : B')}\ \mathsf{TrInd}$$

It should now be clear, how to proceed when developing a function which involves iteration, as the one used to translate λ-expressions into deBruijn representation in Example 4.8. In a first step it is necessary to define the type of the function, that is to make explicit which arguments must be boxed and which not. This is mainly determined by the subject of the iteration. On the basis of this type the type replacement ω need to be specified. In Example 4.8 we defined iteration over an object of type $\Box\text{exp}$. During the traversal of the term, exp was mapped into a function of type $\Box\text{nat} \to \text{db}$ because the relative de Bruijn index changes during the traversal. This is already enough to fix the type replacement: $\omega = \text{exp} \mapsto \Box\text{nat} \to \text{db}$.

In a second step the set of constructors which must be replaced is determined by the inductive datatype $\mathcal{I}(\Sigma; \text{exp}) = \text{exp}$. By definition it follows that

$$\mathcal{S}^*(\Sigma; \text{exp}) = \text{lam} : (\text{exp} \to \text{exp}) \to \text{exp}, \text{app} : \text{exp} \to (\text{exp} \to \text{exp})$$

The operational character of iteration makes it now necessary to define the term replacement, mapping lam and app to objects M_1 and M_2, respectively. The typing rule TrBase and TrInd determine their types, but there is also a very intuitive way to do so: lam expects one parameter, which we assume to be transformed to a parameter of new type. Its type results from replacing every occurrence exp in the type of the parameter by the new type $\Box\mathrm{nat} \to \mathrm{db}$ — which is exactly expressed by the type replacement. Hence M_1 must have the type

$$((\Box\mathrm{nat} \to \mathrm{db}) \to (\Box\mathrm{nat} \to \mathrm{db})) \to (\Box\mathrm{nat} \to \mathrm{db})$$

and similarly the replacement for app must have type

$$(\Box\mathrm{nat} \to \mathrm{db}) \to (\Box\mathrm{nat} \to \mathrm{db}) \to (\Box\mathrm{nat} \to \mathrm{db}).$$

The iteration itself has hence the type $(\Box\mathrm{nat} \to \mathrm{db})$.

Applying a term replacement must be restricted to canonical forms in order to preserve types. Fortunately, our type system guarantees that the subject of an iteration can be converted to canonical form. Applying a replacement then transforms a canonical form V of type B into a well-typed object $\langle\omega;\Omega\rangle(V)$ of type $\langle\omega\rangle(B)$. We call this operation *elimination*. It is defined along the structure of V.

Definition 4.28 (Elimination)

$$\begin{array}{rcll}
\langle\omega;\Omega\rangle(c) & = & \begin{cases} M & \mathit{if}\ \Omega(c) = M \\ c & \mathit{otherwise} \end{cases} & (\mathsf{ElConst}) \\
\langle\omega;\Omega\rangle(x) & = & \Omega(x) & (\mathsf{ElVar}) \\
\langle\omega;\Omega\rangle(\lambda x\!:\!B.\,V) & = & \lambda x'\!:\!\langle\omega\rangle(B).\,\langle\omega;\Omega \mid x \mapsto x'\rangle(V) & (\mathsf{ElLam}) \\
\langle\omega;\Omega\rangle(V_1\ V_2) & = & \langle\omega;\Omega\rangle(V_1)\ \langle\omega;\Omega\rangle(V_2) & (\mathsf{ElApp})
\end{array}$$

Constructors and variables must be mapped to some objects defined in the term replacement Ω. As encountered above, not all types occuring in the subject type of the iteration object live in the inductive datatype. This property implies that elimination might encounter constructors which are not defined in the term replacement. In this case we do not replace the constants, as already indicated by the type replacement which leaves those atomic types unchanged. When eliminating a λ-abstraction $\lambda x\!:\!B.\,V$, ElLam applies: x, introduced by the λ-abstraction is a pseudo constructor which will be renamed to x'. The term replacement must hence be extended by $x \mapsto x'$. The elimination result must then be abstracted over the newly introduced variable x' of type $\langle\omega\rangle(B)$.

The term resulting from elimination might, of course, contain redices and must itself be evaluated to obtain a final value. Thus we obtain the following evaluation rule for iteration.

Definition 4.29 (Evaluation judgment) *extending Definition 3.3:*

$$\frac{\Psi \vdash M \hookrightarrow \mathit{box}\, M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \langle\omega;\Omega\rangle(V') \hookrightarrow V : \langle\omega\rangle(B)}{\Psi \vdash \mathit{it}\ \langle\omega\rangle\ M\ \langle\Omega\rangle \hookrightarrow V : \langle\omega\rangle(B)}\ \mathsf{EvIt}$$

The reader is invited to convince himself that this operational semantics yields the expected results on the examples of this section.

Our calculus also contains a **case** construct whose subject may be of type $\Box B$ for arbitrary pure B. It allows us to distinguish cases based on the intensional structure of the subject. For example, we can test if a given (parametric!) function is the identity or not. We discuss the case construct in the next section.

5 Case

Iteration is a powerful mechanism which replaces a general recursion scheme in our system. We have seen that we can recover primitive recursion of natural numbers since our calculus contains pairs. A quite natural question to ask is if iteration can be used to mimic definition by cases. We currently have no proof of this, but we strongly suspect that it is not. We start with some easy examples.

Example 5.1 (Comparison) To check if a natural number is greater then 0 we would like to write informally

$$\begin{array}{rcl} \mathrm{gt0}\ m & = & \mathrm{case}\ m\ \mathrm{of}\ \mathrm{z} \Rightarrow \mathrm{false} \\ & & \mid (\mathrm{s}\ m') \Rightarrow \mathrm{true} \end{array}$$

In our system we view case distinction as a selection of a branch, triggered by the head constant of the case subject. We replace the z : nat by an object $M_z : A$ and s : nat $\to$ nat by $M_s : \Box\mathrm{nat} \to A$. Note that the argument type of M_s is $\Box$nat and not nat as one might suspect at a first sight. This is because we know that the case subject is closed and hence its arguments must be closed, too. It seems that our construction does not work for functional case subjects. To solve this obvious contradiction is one of the main difficulties of the design of a suitable case operator. The case construct in its simplest form is written as "case $\langle A\rangle$ M $\langle\Xi\rangle$" where M (of type $\Box a$) is the subject of case, and Ξ is a list containing matches for all constructors of type a.

Example 5.2 (Comparison with case) The greater-than function from example 5.1 can be formulated as follows:

$$\begin{array}{rcl} \mathrm{gt0} & : & \Box\mathrm{nat} \to \mathrm{bool} \\ & = & \lambda m\!:\!\Box\mathrm{nat}.\,\mathrm{case}\ \langle\mathrm{bool}\rangle\ m\ \langle \mathrm{z} \Rightarrow \mathrm{false} \mid \mathrm{s} \Rightarrow \lambda m'\!:\!\Box A.\,\mathrm{true}\rangle \end{array}$$

Boolean connectives serve as further simple examples which show the use of the case constructor in our system. We have already seen a representation of conjunction in example 4.6.

Example 5.3 (Boolean operators) Informally we can represent not and or as follows. We must require all argument and all result types are boxed, because the result of a boolean operation might be used for another case distinction — as it is commonly the case.

$$\begin{array}{rl} \mathrm{not}\ B = & \mathrm{case}\ B\ \mathrm{of} \\ & \langle\, \mathrm{true} \Rightarrow \mathrm{false} \\ & \mid \mathrm{false} \Rightarrow \mathrm{true}\rangle \end{array} \qquad \begin{array}{rl} \mathrm{or}\ B_1\ B_2 = & \mathrm{case}\ B_1\ \mathrm{of} \\ & \langle\, \mathrm{true} \Rightarrow \mathrm{true} \\ & \mid \mathrm{false} \Rightarrow B_2\rangle \end{array}$$

The formal representation of the Boolean operations is as follows:

$$\begin{array}{rcl} \mathrm{not} & : & \Box\mathrm{bool} \to \Box\mathrm{bool} \\ & = & \lambda B\!:\!\Box\mathrm{bool}. \\ & & \quad \mathrm{case}\ \langle\Box\mathrm{bool}\rangle\ B \\ & & \quad \langle\, \mathrm{true} \Rightarrow \mathrm{box\ false} \\ & & \quad \mid \mathrm{false} \Rightarrow \mathrm{box\ true}\rangle \end{array} \qquad \begin{array}{rcl} \mathrm{or} & : & \Box\mathrm{bool} \to \Box\mathrm{bool} \to \Box\mathrm{bool} \\ & = & \lambda B_1\!:\!\Box\mathrm{bool}.\,\lambda B_2\!:\!\Box\mathrm{bool}. \\ & & \quad \mathrm{case}\ \langle\Box\mathrm{bool}\rangle\ B_1 \\ & & \quad \langle\, \mathrm{true} \Rightarrow \mathrm{box\ true} \\ & & \quad \mid \mathrm{false} \Rightarrow B_2\rangle \end{array}$$

Many more examples are representable in our system. We start with presenting subtraction (which we already assumed to be representable in Example 4.8) where we will need a combination of iteration and case distinction.

Example 5.4 (Subtraction) Among others the type of subtraction could be $\Box\text{nat} \to \Box\text{nat} \to \Box\text{nat}$. It is informally defined as follows.

$$\begin{array}{lcl} \text{minus } m \text{ z} & = & m \\ \text{minus } m \text{ (s } n') & = & \text{case } m \text{ of z} \Rightarrow \text{z} \\ & & \mid (\text{s } m') \Rightarrow (\text{minus } m' \; n') \end{array}$$

Both arguments of minus must be closed, because we use case distinction over the first argument and iteration over the second.

$$\begin{array}{lcl} \text{minus} & : & \Box\text{nat} \to \Box\text{nat} \to \Box\text{nat} \\ & = & \lambda x{:}\Box\text{nat}.\, \lambda y{:}\Box\text{nat}.\, \text{it}\, \langle \text{nat} \mapsto (\Box\text{nat} \to \Box\text{nat})\rangle\; y \\ & & \quad \langle\, \text{z} \mapsto \lambda m{:}\Box\text{nat}.\, m \\ & & \quad \mid \text{s} \mapsto \lambda n{:}(\Box\text{nat} \to \Box\text{nat}). \\ & & \qquad\quad \lambda m{:}\Box\text{nat}.\, \text{case}\, \langle \Box\text{nat}\rangle\; m \\ & & \qquad\qquad \langle\, \text{z} \Rightarrow \text{box z} \\ & & \qquad\qquad \mid \text{s} \Rightarrow \lambda m'{:}\Box\text{nat}.\, (n\; m')\rangle\rangle\; x \end{array}$$

Case is not restricted to range over atomic types only. Using case over functional types we show in the next example how an identity function test can be implemented.

Example 5.5 (Identity test) Below is a function which decides if a parametric function mapping exp to exp is the identity function or not. The function has type $\Box(\text{exp} \to \text{exp}) \to \text{bool}$.

$$\begin{array}{lcl} \text{id-test } E & = & \text{case } E \text{ of } \lambda x{:}\text{exp}.\, (\text{app } (E_1\; x)\; (E_2\; x)) \Rightarrow \text{false} \\ & & \mid \lambda x{:}\text{exp}.\, (\text{lam } \lambda y{:}\text{exp}.\, E\; x\; y) \Rightarrow \text{false} \\ & & \mid \lambda x{:}\text{exp}.\, x \Rightarrow \text{true} \end{array}$$

Following the same idea as above we match in the first case F with app : exp $\to$ (exp $\to$ exp) and return M_a. Instead of just boxing the arguments of app we must be more careful because those arguments might contain the free variable which was introduced by the case subject. In fact every argument of app must be closed under this variable: The objects to be expected by M_a have hence the form $\lambda x{:}\text{exp}.\, E_1$ and $\lambda x{:}\text{exp}.\, E_2$, both of type exp $\to$ exp. Since x was the only free variable which might occur in E_1 or E_2 we also know that both λ-expressions are closed. Hence they can be boxed. The type of M_a is therefore $\Box(\text{exp} \to \text{exp}) \to \Box(\text{exp} \to \text{exp}) \to \Box\text{bool}$.

A very similar argument can be applied to determine the type of M_l, which is the match for lam : (exp $\to$ exp) $\to$ exp. x can occur free in the body E of the λ-expression, hence M_l will be passed the boxed object $\lambda x{:}\text{exp}.\, E$ which gives M_l the type $\Box(\text{exp} \to (\text{exp} \to \text{exp})) \to \Box\text{bool}$.

It should not be forgotten that there is also a third case to be considered. x can occur as a pseudo constructor in the body of the case subject. Here again as in the iterator case, the matching objects for pseudo constructors are not given immediately, but instead the result of case is a function *expecting* the match object M_x for the pseudo constructor: M_x must be of type $\Box$bool. Let us return to the previous example:

Example 5.5 (Identity test using case) The identity test function is hence represented as follows.

$$\begin{array}{lcl} \text{id-test} & : & \Box(\text{exp} \to \text{exp}) \to \Box\text{bool} \\ & = & \lambda F{:}\Box(\text{exp} \to \text{exp}).\, \text{case}\, \langle \text{bool}\rangle\; F \\ & & \quad \langle\, \text{app} \Rightarrow \lambda E_1{:}\Box(\text{exp} \to \text{exp}).\, \lambda E_2{:}\Box(\text{exp} \to \text{exp}).\, \text{box false} \\ & & \quad \mid \text{lam} \Rightarrow \lambda E{:}\Box(\text{exp} \to \text{exp} \to \text{exp}).\, \text{box false}\rangle\; (\text{box true}) \end{array}$$

In the following we develop two functions to test if an expression from Example 2.5 is a β-redex or if it is an η-redex. It shows how more than one case construction can be nested, and demonstrates again how case distinction proceeds over functional objects. To remind the reader β- and η-reduction are defined as follows.

$$\begin{array}{lll} \beta\text{-reduction:} & (\lambda x\!:\!\mathrm{exp}.\, E_1)\ E_2 & \rightsquigarrow\ [E_2/x](E_1) \\ \eta\text{-reduction:} & \lambda x\!:\!\mathrm{exp}.\,(E\ x) & \rightsquigarrow\ E \quad \text{where } x \text{ does not occur free in } E \end{array}$$

$\lambda x\!:\!\mathrm{exp}.\, E_1\ E_2$ is called a β redex, $\lambda x\!:\!\mathrm{exp}.\,(E\ x)$ is called a η-redex if x does not occur free in E. These examples can be modified to implement the reduction.

Example 5.6 (β-redex test) The β-redex test function has type $\Box\mathrm{exp} \to \Box\mathrm{bool}$ and can informally be defined as follows.

$$\begin{array}{lll} \text{beta-test } F & = & \text{case } F \text{ of } (\text{lam } E) \Rightarrow \text{false} \\ & & \mid (\text{app } E_1\ E_2) \Rightarrow\ \text{case } E_1 \text{ of } (\text{lam } E') \Rightarrow \text{true} \\ & & \qquad\qquad\qquad\qquad \mid (\text{app } E_1'\ E_2') \Rightarrow \text{false} \end{array}$$

Its representation in our calculus is hence:

$$\begin{array}{lll} \text{beta-test} & : & \Box\mathrm{exp} \to \Box\mathrm{bool} \\ & = & \lambda F\!:\!\Box\mathrm{exp}.\ \text{case } \langle \Box\mathrm{bool}\rangle\ F \\ & & \quad \langle\, \text{lam} \Rightarrow \lambda E\!:\!\Box(\mathrm{exp}\to\mathrm{exp}).\ \text{box false} \\ & & \quad \mid \text{app} \Rightarrow \lambda E_1\!:\!\Box\mathrm{exp}.\,\lambda E_2\!:\!\Box\mathrm{exp}. \\ & & \qquad\quad \text{case } \langle\Box\mathrm{bool}\rangle\ E_1 \\ & & \qquad\quad \langle\, \text{lam} \Rightarrow \lambda E'\!:\!\Box(\mathrm{exp}\to\mathrm{exp}).\ \text{box true} \\ & & \qquad\quad \mid \text{app} \Rightarrow \lambda E_1'\!:\!\Box\mathrm{exp}.\,\lambda E_2'\!:\!\Box\mathrm{exp}.\ \text{box false}\rangle\rangle \end{array}$$

Example 5.7 (η-redex test) The function to decide if a given expression is a η-redex is more difficult to define. Clearly, it will have type $\Box\mathrm{exp} \to \Box\mathrm{bool}$. The main difficulties arise because the decision cannot simply be made by considering the structure of the expression, but we must ensure the side condition for η-redices. Fortunately we already defined the functions const and id-test which come handy for the informal definition of the η-redex test.

$$\begin{array}{lll} \text{eta-test } F & = & \text{case } F \text{ of} \\ & & (\text{lam } E) \Rightarrow\ \text{case } E \text{ of } \lambda x\!:\!\mathrm{exp}.\,(\text{lam } \lambda y\!:\!\mathrm{exp}.\, E'\ x\ y) \Rightarrow \text{false} \\ & & \qquad\qquad\quad \mid \lambda x\!:\!\mathrm{exp}.\,(\text{app } (E_1'\ x)\ (E_2'\ x)) \Rightarrow \\ & & \qquad\qquad\qquad (\text{and } (\text{const } E_1')\ (\text{id-test } E_2')) \\ & & \qquad\qquad\quad \mid \lambda x\!:\!\mathrm{exp}.\, x \Rightarrow \text{false} \\ & & \mid (\text{app } E_1\ E_2) \Rightarrow \text{false} \end{array}$$

Its representation in our calculus is hence:

$$\begin{array}{lll} \text{eta-test} & : & \Box\mathrm{exp} \to \Box\mathrm{bool} \\ & = & \lambda F\!:\!\Box\mathrm{exp}. \\ & & \quad \text{case } \langle\Box\mathrm{bool}\rangle\ F \\ & & \quad \langle\, \text{lam} \Rightarrow \lambda E\!:\!\Box(\mathrm{exp}\to\mathrm{exp}). \\ & & \qquad \text{case } \langle\Box\mathrm{bool}\rangle\ E \\ & & \qquad \langle\, \text{lam} \Rightarrow \lambda E'\!:\!\Box(\mathrm{exp}\to\mathrm{exp}\to\mathrm{exp}).\ \text{box false} \\ & & \qquad \mid \text{app} \Rightarrow \lambda E_1'\!:\!\Box(\mathrm{exp}\to\mathrm{exp}).\,\lambda E_2'\!:\!\Box(\mathrm{exp}\to\mathrm{exp}). \\ & & \qquad\qquad (\text{and } (\text{const } E_1')\ (\text{id-test } E_2'))\rangle\ (\text{box false}) \\ & & \quad \mid \text{app} \Rightarrow \lambda E_1\!:\!\Box\mathrm{exp}.\,\lambda E_2\!:\!\Box\mathrm{exp}.\ (\text{box false})\rangle \end{array}$$

We begin now with the formal discussion of the case construct: Differently from iteration which traverses the entire structure of the subject, case only recurses down to the head constructor of the subject leaving possible arguments aside. The subject for selection is always of the form $\lambda x_1 : B_1 \ldots \lambda x_m : B_m .\, c\, M_1..M_n$ with a head constructor c of type B'. Operationally speaking, during the process of selection, the head constructor is replaced by an object M representing the computational content of the applying case. At a first glance one might suspect that M's type is $B'_1 \to .. \to B'_n \to A$ where the B'_i's are the argument types of c and A is the result type of the case. This is not powerful enough. Since case distinction requires its subject to be closed, no further case distinction could be performed over any of the objects $M_1..M_n$. To solve this dilemma we close each argument M_i by abstracting over each variable which might possibly occur free in it. It should be clear that all those variables can be determined because each M_i is a subobject of the case subject. This allows us to finally close the newly constructed object with a box. To make this more formal we define a generalized λ-abstraction which we call *abstraction closure*: $\lambda\{\Psi\}.\, M$ stands for a closed object where M is wrapped in λ-abstractions defined by Ψ.

Definition 5.8 (Abstraction closure)

$$\begin{aligned} \lambda\{\cdot\}.\, M &:= M \\ \lambda\{\Psi', x : B\}.\, M &:= \lambda\{\Psi'\}.\, (\lambda x{:}B.\, M) \end{aligned}$$

and its type is defined as $\Pi\{\Psi\}.\, A$:

Definition 5.9 (Abstraction closure type)

$$\begin{aligned} \Pi\{\cdot\}.\, A &:= A \\ \Pi\{\Psi, x : A'\}.\, A &:= \Pi\{\Psi\}.\, (A' \to A) \end{aligned}$$

Returning to our discussion we can now write box $(\lambda\{\Psi\}.\, M_i)$ for the abstracted and closed versions of M_i where Ψ is a context accounting for all free variables possibly occuring in M_i. It follows that this argument closing operation determines the type of M which we discuss next.

In Example 5.5 we encountered the problem to assign types to the arguments of the object M_a and M_l which represent the computational meaning of the cases app and lam, respectively. We generalize now this approach which leads to the notion of *case types*. As pointed out above, the general form of the canonical case subject is $\lambda\{\Psi\}.\, c\, M_1..M_n$ with a head constructor c of type B'. The type of the case subject is hence $\Pi\{\Psi\}.\, a$ for some atomic type a. The set of different constructors c constitutes all constructors with the same target type a. All pseudo constructors defined in Ψ having target type a could also occur in the position of c. Consider now some (pseudo) constructor c of type B' where $\tau(B') = a$. Selecting a case for this constructor c means to select an object M_c — carrying the intended computational meaning for this case. M_c must be function, which expects as parameters box $\lambda\{\Psi\}.\, M_1 \ldots$ box $\lambda\{\Psi\}.\, M_m$ as has be motivated above. M_c's type can hence be derived from the type of the case subject $B = \Pi\{\Psi\}.\, a$, the result type of case A and the type of the constructor $c : B'$. We abbreviate the type by writing $\mathcal{C}\ (\Pi\{\Psi\}.\, a, A, B')$.

Definition 5.10 (Case types)

$$\begin{aligned} \mathcal{C}\ ((\Pi\{\Psi\}.\, a), A, a') &:= \begin{cases} A & \textit{if } a = a' \\ a' & \textit{otherwise} \end{cases} \\ \mathcal{C}\ ((\Pi\{\Psi\}.\, a), A, (B_1 \to B_2)) &:= \Box(\Pi\{\Psi\}.\, B_1) \to \mathcal{C}\ ((\Pi\{\Psi\}.\, a), A, B_2) \end{aligned}$$

Note, that in the presentation so far $\tau(B) = \tau(B')$ holds. Hence the *otherwise* case in the definition above does not apply. This changes if we direct our attention to the type of the case object itself since pseudo constructors must be replaced by variables of case types. That is if the case subject is of type $B = B_1 \to \ldots \to B_m \to a$ then the type of the case construct is $\mathcal{C}\ (B, A, B_1) \to \ldots \to \mathcal{C}\ (B, A, B_m) \to A$. It is obvious that there might be pseudo constructor types B_i which target type is different from a. In those cases, the atomic case types $\mathcal{C}\ (B, A, B')$ must be well-defined which makes the *otherwise* clause necessary. In fact those pseudo constructors can never occur in head position. Even though we wish to omit them from the definition of case types in general — as the following example shows — we do not pursue this idea here, but leave it to future research.

Example 5.11 (Equality formula test for higher order logic) Consider a function which returns true if the a higher-order formula is of the form $t_1 = t_2$, else false. We call this function eq-test. The type of this function should be $\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \Box\mathrm{bool}$. Informally we would write:

$$\begin{array}{rcl} \text{eq-test } F & = & \text{case } F \text{ of} \\ & & \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\, \mathrm{forall}\ \lambda x\!:\!\mathrm{o}.\, F'\ r\ x \Rightarrow \mathrm{false} \\ & & \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\, \mathrm{impl}\ (F_1'\ r)\ (F_2'\ r) \Rightarrow \mathrm{false} \\ & & \lambda r\!:\!\mathrm{o} \to \mathrm{i}.\, \mathrm{eq}\ (t_1'\ r)\ (t_2'\ r) \Rightarrow \mathrm{true} \end{array}$$

The straightforward representation of this function in our system would be

$$\begin{array}{rcl} \text{eq-test} & : & \Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to (\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \mathrm{i}) \to \Box\mathrm{bool} \\ & = & \lambda F\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}). \\ & & \quad \text{case } \langle \Box\mathrm{bool} \rangle\ F \\ & & \quad \langle\, \mathrm{forall} \Rightarrow \lambda F'\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{i} \to \mathrm{o}).\, \mathrm{box\ false} \\ & & \quad \mid \mathrm{impl} \Rightarrow \lambda F_1'\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}).\, \lambda F_2'\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}).\, \mathrm{box\ false} \\ & & \quad \mid \mathrm{eq} \Rightarrow \lambda t_1'\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{i}).\, \lambda t_2'\!:\!\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{i}).\, \mathrm{box\ true} \rangle \end{array}$$

The type of this representation of eq-test has an unexpected form, we would expect that its type is $\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \Box\mathrm{bool}$. The reader can convince himself, that the type of eq-test is correct, and that the second argument type $\Box((\mathrm{o} \to \mathrm{i}) \to \mathrm{o}) \to \mathrm{i}$ represents the branch for the pseudo constructor. The pseudo constructor itself happens to be insignificant because it cannot occur in head position but our system currently does not check this special case for simplicity of the development below and the proofs in section 7. Thus a dummy argument must be supplied.

The type of case construction case $\langle A \rangle\ M\ \Xi$ is called *complete case type* $\mathcal{C}^*\ (B, A, B)$ where B is the type of M. $\mathcal{C}^*\ (B, A, B')$ is defined for some pure type B' as follows.

Definition 5.12 (Complete case type)

$$\begin{array}{rcl} \mathcal{C}^*\ (B, A, a) & := & \mathcal{C}\ (B, A, a) \\[1em] \mathcal{C}^*\ (B, A, (B_1 \to B_2)) & := & \mathcal{C}\ (B, A, B_1) \to \mathcal{C}^*\ (B, A, B_2) \end{array}$$

Case distinction is now defined similarly to iteration. The result of the selection process — i.e. executing the case construct — is an object which resembles the (canonical) subject of the case in structure, but the head constant is replaced by some *matched* object carrying the intended operational meaning of the selected branch. Even though the subject of iteration is closed before the selection process, we need to deal with embedded λ-abstractions introducing pseudo constructors. We can simply replace variables x of type B' by new variables x' of type $\mathcal{C}\ (B, A, B')$, where B is the type of the case subject.

Definition 5.13 (Match)

$$\textit{Match:}\quad \Xi \quad ::= \quad \cdot \mid (\Xi \mid c \Rightarrow M) \mid (\Xi \mid x \Rightarrow x')$$

Initially the domain of a match is a signature containing all constructors whose target type is equal to $\tau(B)$. This signature is $\mathcal{S}(\Sigma; \tau(B))$. The form of case follows naturally: We extend the notion of objects by

$$M ::= \ldots \mid \text{case}\ \langle A\rangle\ M\ \langle\Xi\rangle$$

and extend the typing rules for case. To do so we must introduce a new typing judgment for matches Ξ: $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma)$. Ξ is well-typed if it provides an object of case type for every constant in some signature Σ.

Definition 5.14 (Typing judgment for case) *extending definition 3.1*

$$\frac{\Delta; \Gamma \vdash M : \Box B \qquad \Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')}{\Delta; \Gamma \vdash \textit{case}\ \langle A\rangle\ M\ \langle\Xi\rangle : \mathcal{C}^*\ (B, A, B)}\ \mathsf{TpCase}$$

$$\textit{where}\ \Sigma' = \mathcal{S}(\Sigma; \tau(B))$$

$$\frac{}{\Delta; \Gamma \vdash \cdot : \langle B \Rightarrow A\rangle(\cdot)}\ \mathsf{TmBase} \qquad \frac{\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma) \qquad \Delta; \Gamma \vdash M : \mathcal{C}\ (B, A, B')}{\Delta; \Gamma \vdash (\Xi \mid c \Rightarrow M) : \langle B \Rightarrow A\rangle(\Sigma, c : B')}\ \mathsf{TmInd}$$

To summarize definition by cases we return to Example 5.5. As in the iteration case, it is necessary to first define the type of the function, because the subject of case must be closed. For our example we expect a function as input, which must be passed as subject to case. The result is bool, which makes it necessary to box it, otherwise it cannot be used as input for other Boolean operations. Hence id-test has type $\Box(\text{exp} \to \text{exp}) \to \Box\text{bool}$.

The second step is to examine the type $B = \Box(\text{exp} \to \text{exp})$ and the signature Σ for possible constructors and pseudo constructors of this type. For id-test we see that only lam, app, and x, the newly introduced pseudo constructor are candidates for the head position. After determining the types of the match objects M_l, M_a, and M_x, the objects can be defined.

Having done this, a match must be constructed, representing only the constructors from the signature Σ and the according case objects ($\Xi = \text{lam} \Rightarrow M_l, \text{app} \Rightarrow M_a$). The case construct is then a function which must be applied to the object M_x.

The operational semantics of case is defined by one rule using an auxiliary function which defines the process of *selection*. The subject of case must be closed. Therefore we can define selection along its canonical form. The selection process then transforms this canonical object V of type B into $\{B \Rightarrow A; \Xi; \cdot\}(V)$ of type $\mathcal{C}^*\ (B, A, B)$.

Definition 5.15 (Selection)

$$
\begin{aligned}
\{B \Rightarrow A; \Xi; \Psi\}(c) &= \Xi(c) && (\mathsf{SeConst})\\
\{B \Rightarrow A; \Xi; \Psi\}(x) &= \Xi(x) && (\mathsf{SeVar})\\
\{B \Rightarrow A; \Xi; \Psi\}(\lambda x : B'.\, V) &= \lambda u : \mathcal{C}\ (B, A, B').\, \{B \Rightarrow A; \Xi, x \Rightarrow u; (\Psi, x : B')\}(V) && (\mathsf{SeLam})\\
\{B \Rightarrow A; \Xi; \Psi\}(V_1\, V_2) &= \{B \Rightarrow A; \Xi; \Psi\}(V_1)\ (\mathit{box}\ \lambda\{\Psi\}.\, V_2) && (\mathsf{SeApp})
\end{aligned}
$$

Looking at an arbitrary canonical form of a case subject of type B we recognize that it has always the form $\lambda\{\Psi\}.\, c\, M_1..M_n$. Performing selection means, to first traverse all λ-abstractions, and introducing new variables for the functionality associated with each pseudo-constructor. This is done by rule SeLam. While traversing the body of the canonical form, each argument M_i must be closed under Ψ and boxed which is expressed by rule SeApp. Eventually the head constructor c is reached. In the case that c is a constructor, SeConst replaces the constant by the object which is defined in the match Ξ. If c is a pseudo constructor, then it was a variable name, which counterpart can also be found in the match Ξ by SeVar.

The selection process is triggered by an additional evaluation rule, which defines the operational semantics of case.

Definition 5.16 (Evaluation judgment) *extending definition 3.3:*

$$
\frac{\Psi \vdash M \hookrightarrow \mathit{box}\, M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}{\Psi \vdash \mathit{case}\ \langle A \rangle\ M\ \langle \Xi \rangle \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}\ \mathsf{EvCase}
$$

The reader can now convince himself that the operational semantics yields the expected results on the examples of this section. This concludes the presentation of core system of the modal λ-calculus. In the next sections, we address the meta-theoretical properties of our system.

6 Preliminary results

In the remainder of the paper we seek to prove that the modal λ-calculus we were presenting in the previous sections is a conservative extension over the simply typed λ-calculus from Section 2. This theorem shows that every object of pure type in our system evaluates to an object of canonical form in the simply typed λ-calculus.

A milestone on the way towards this proof is the canonical form theorem which we present in the next section. It guarantees, that every object typable by a pure type possesses a canonical form. As a corollary of the canoncial form theorem we obtain a type preservation result which says that types are preserved under evaluation.

In the remainder of this paper we will need some more basic technical notions and properties which we are presenting in this section. Due to the basic character, a lot of the forthcoming lemmas of this section are very clear and their proofs do not require more than easy inductive arguments. If appropriate we omit the proof.

6.1 Context

In section 2 we defined a context Ψ as a list of variables of pure type which we generalized to a context Γ representing variables of arbitrary type in section 3. In the following sections it will be necessary to reason about contexts. The argument will involve extensions of contexts. Since this is a very standard and basic definition we introduce it at this point and show some simple properties which we need later.

Definition 6.1 (Context extension) $\Gamma' \geq \Gamma :\Leftrightarrow$ *Context* Γ' *extends context* Γ.
Rules:

$$\frac{}{\Gamma \geq \Gamma}\,\mathsf{CeBase} \qquad \frac{\Gamma' \geq \Gamma}{\Gamma', x : A \geq \Gamma}\,\mathsf{CeInd}$$

Note that defining a context extension for Γ subsumes the notion of context extension for pure contexts Ψ. We show now four properties which are a direct consequence of this definition. First, it is clear that every context extends the empty context. The proof is done by induction.

Lemma 6.2 (Every context extends the empty context) $\Gamma \geq \cdot$

Proof: See appendix B. □

Second, context extension is transitive. This can be easily shown by induction over the structure of the second Γ'.

Lemma 6.3 (Transitivity of context extension) *If* $\Gamma'' \geq \Gamma'$ *and* $\Gamma' \geq \Gamma$ *then* $\Gamma'' \geq \Gamma$

Proof: omitted. □

And third if a context $\bar{\Gamma}$ extends a union of two contexts, then $\bar{\Gamma}$ itself can be written as a union of one of contexts and an extension of the other. This less obvious property is proven by induction.

Lemma 6.4 (Context form) *If* $\bar{\Gamma}'' \geq \Gamma \cup \bar{\Gamma}$ *then* $\bar{\Gamma}'' = \Gamma \cup \bar{\Gamma}'$ *and* $\bar{\Gamma}' \geq \bar{\Gamma}$

Proof: See appendix B. □

As fourth and last property of context extension we show that if one context extends another, the extension remains valid under union with some other context.

Lemma 6.5 (Context union) *If* $\bar{\Gamma}' \geq \bar{\Gamma}$ *then* $\Gamma \cup \bar{\Gamma}' \geq \Gamma \cup \bar{\Gamma}$

Proof: omitted. □

Without proof we just state that the union with the empty context yields the same context.

Lemma 6.6 (Empty context on the left) $\cdot \cup \Psi = \Psi$

Proof: omitted. □

Recall how the access to a context was defined: We write $\Gamma(x) = A$ iff we can write Γ as $\Gamma_1, x \in A \cup \Gamma_2$. To be painstakingly precise we must require that if Γ itself was a union of two contexts, then one of the contexts must be written as such a union — we omit the boring proof.

Lemma 6.7 (Context union access) *If* $(\Gamma \cup \Gamma')(x) = A$ *then* $\Gamma = \Gamma_1, x : A \cup \Gamma_2$ *or* $\Gamma' = \Gamma'_1, x : A \cup \Gamma'_2$

Proof: omitted. □

Closely related to contexts are substitutions which we introduce after discussing the typing relation of our system.

6.2 Typing

The basic property which is needed in the proof of lemma 7.17 in the next section is the admissibility of weakening for the typing relation. In other words: An extension of the typing context cannot invalidate typing derivations. The following lemma has two parts, the first part discusses weakening in the modal context, the second weakening in the regular context. We omit the easy proof by induction.

Lemma 6.8 (Weakening on typing relation)

1. *If* $\Delta; \Gamma \vdash M : A$ *and* $\Delta' \geq \Delta$ *then* $\Delta'; \Gamma \vdash M : A$
2. *If* $\Delta; \Gamma \vdash M : A$ *and* $\Gamma' \geq \Gamma$ *then* $\Delta; \Gamma' \vdash M : A$

Proof: omitted. □

Besides weakening we need another important lemma — the substitution lemma which we present in two different flavors. The first substitution lemma guarantees, if we replace a variable from the modal context by a closed object of required type, the result will still be typable.

Lemma 6.9 (Modal substitution lemma)

If $\Delta, y : A_1; \Gamma \vdash M : A_2$ *and* $\Delta; \cdot \vdash M' : A_1$ *then* $\Delta; \Gamma \vdash [M'/y](M) : A_2$

Proof: omitted. □

The second substitution lemma is very similar to the first: It says that if a variable from the regular context is replaced by an arbitrary object of correct type, then the result will be still typable.

Lemma 6.10 (Regular substitution lemma)

If $\Delta; \Gamma, y : A_1 \vdash M : A_2$ *and* $\Delta; \Gamma \vdash M' : A_1$ *then* $\Delta; \Gamma \vdash [M'/y](M) : A_2$

Proof: omitted. □

These three lemmas form the basic properties of the typing relation which we need for the formal discussion of our system. In the next subsection we introduce the notion of substitution and show some related basic properties.

6.3 Substitution

In Section 3 we described the distinction between the parametric function space $A_1 \to A_2$ and the primitive recursive function space $A_1 \Rightarrow A_2$ which made a refinement of context Ψ from Section 2 necessary: We introduced the *modal* context Δ, whose variables range over closed objects and the *arbitrary* context Γ to replace Ψ — the context representing variables of arbitrary type in the simply typed λ-calculus. Contexts and substitutions are closely related. A substitution is defined as follows.

$$\text{Substitution:} \quad \varrho \quad ::= \quad \cdot \mid \varrho, M/x$$

Due to the presence of two contexts we carefully distinguish between a modal substitution θ which substitutes closed objects for variables defined in a context Δ and ϱ which substitutes arbitrary objects for variables defined in a context Γ. We write $\theta; \varrho$ for such a pair of (necessaily disjoint) substitutions. Being disjoint means, that θ and ϱ do not have any variable names in common. This is guaranteed, because the context $\Delta; \Gamma$ cannot declare the same variable name twice.

In our system substitutions are only applied to well-typed objects. Moreover a substitution must substitute something for every free variable in the object. We make this intuition about well-typed substitutions now more precise by introducing a typing judgment $\Delta'; \Gamma' \vdash (\theta; \varrho) : (\Delta; \Gamma)$ for substitutions. $\theta; \varrho$ can be applied to objects which are well-typed in context $\Delta; \Gamma$. The range of the substitution $\theta; \varrho$ are objects which might depend on free variables from $\Delta'; \Gamma'$.

Definition 6.11 (Typing of substitution judgment)
Rules:

$$\frac{}{\Delta'; \Gamma' \vdash (\cdot; \cdot) \ : \ (\cdot; \cdot)} \ \mathsf{TSBase}$$

$$\frac{\Delta'; \cdot \vdash M : A \qquad \Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)}{\Delta'; \Gamma' \vdash (\theta, M/x; \varrho) \ : \ (\Delta, x : A; \Gamma)} \ \mathsf{TSMod}$$

$$\frac{\Delta'; \Gamma' \vdash M : A \qquad \Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)}{\Delta'; \Gamma' \vdash (\theta; \varrho, M/x) \ : \ (\Delta; \Gamma, x : A)} \ \mathsf{TSReg}$$

Note that substitutions satisfy a modal restriction (rule TSMod) which mirrors the restriction on typing box . Throughout this paper we can apply a substitution $(\theta; \varrho)$ satisfying $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ to an object M, a term replacement Ω, or a match Ξ only if

$$\Delta; \Gamma \vdash M : A \qquad \Delta; \Gamma \vdash \Omega : \langle \omega \rangle (B') \qquad \Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle (B')$$

holds, respectively. The result of the application will then be $[\theta; \varrho](M)$, $[\theta; \varrho](\Omega)$, or $[\theta; \varrho](\Xi)$ which have the following properties, respectively:

$$\Delta'; \Gamma' \vdash [\theta; \varrho](M) : A \qquad \Delta'; \Gamma' \vdash [\theta; \varrho](\Omega) : \langle \omega \rangle (B') \qquad \Delta'; \Gamma' \vdash [\theta; \varrho](\Xi) : \langle B \Rightarrow A \rangle (B')$$

We address now the definition of substitution application by starting to define how to access a substitution ϱ. The definition follows similarly to the access of contexts as presentented in Section 2. First a union operation $\varrho_1 \cup \varrho_2$ is needed.

Definition 6.12 (Substitution Union)
Rules:

$$\varrho \cup \cdot = \varrho \qquad (\mathsf{SbuBase})$$

$$\varrho_1 \cup (\varrho_2, M/x) = (\varrho_1 \cup \varrho_2), M/x \qquad (\mathsf{SbuInd})$$

To look up a value of a variable in a substitution we write $\varrho(x)$ which definition follows closely the definition of the lookup function for types in a context (Definition 2.2).

Definition 6.13 (Substitution access)
Rules:

$$(\varrho, M/y)(x) = \begin{cases} M & \textit{if } x = y \\ \varrho(x) & \textit{otherwise} \end{cases} \qquad (\mathsf{SbaInd})$$

To look up the value in a substitution is the first step towards the application of a substitution $\theta; \varrho$ to an object which we define below. Recall that it is necessary to include both substitutions in this definition because of the distinction between modal and non-modal variables.

Definition 6.14 (Substitution application:) *Let $\theta; \varrho$ a substitution.*
Rules:

$$[\theta;\varrho](x) = \begin{cases} M & \textit{if } \theta(x) = M \\ M & \textit{if } \varrho(x) = M \end{cases} \qquad (\mathsf{SBVar})$$

$$[\theta;\varrho](c) = c \qquad (\mathsf{SBConst})$$

$$[\theta;\varrho](\lambda x:A.\, M) = \lambda x:A.\, [\theta;\varrho, x/x](M) \qquad (\mathsf{SBLam})$$

$$[\theta;\varrho](M_1\ M_2) = [\theta;\varrho](M_1)\ [\theta;\varrho](M_2) \qquad (\mathsf{SBApp})$$

$$[\theta;\varrho](\langle M_1, M_2\rangle) = \langle [\theta;\varrho](M_1), [\theta;\varrho](M_2)\rangle \qquad (\mathsf{SBPair})$$

$$[\theta;\varrho](\mathit{fst}\ M) = \mathit{fst}\ [\theta;\varrho](M) \qquad (\mathsf{SBFst})$$

$$[\theta;\varrho](\mathit{snd}\ M) = \mathit{snd}\ [\theta;\varrho](M) \qquad (\mathsf{SBSnd})$$

$$[\theta;\varrho](\mathit{box}\ M) = \mathit{box}\ [\theta;\cdot](M) \qquad (\mathsf{SBBox})$$

$$[\theta;\varrho](\mathit{let\ box}\ x = M_1\ \mathit{in}\ M_2) = \mathit{let\ box}\ x = [\theta;\varrho](M_1)\ \mathit{in}\ [\theta, x/x;\varrho](M_2) \qquad (\mathsf{SBLet})$$

$$[\theta;\varrho](\mathit{case}\ \langle A\rangle\ M\ \langle\Xi\rangle) = \mathit{case}\ \langle A\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Xi)\rangle \qquad (\mathsf{SBCase})$$

$$[\theta;\varrho](\mathit{it}\ \langle\omega\rangle\ M\ \langle\Omega\rangle) = \mathit{it}\ \langle\omega\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Omega)\rangle \qquad (\mathsf{SBIt})$$

Substitution on replacements Ω is defined as:
Rules:

$$[\theta;\varrho](\cdot) = \cdot \qquad (\mathsf{SBOmegaEmpty})$$

$$[\theta;\varrho](\Omega | c \mapsto M) = [\theta;\varrho](\Omega)\ |\ (c \mapsto [\theta;\varrho](M)) \qquad (\mathsf{SBOmega})$$

Substitution on matches Ξ is defined as:
Rules:

$$[\theta;\varrho](\cdot) = \cdot \qquad (\mathsf{SBXiEmpty})$$

$$[\theta;\varrho](\Xi\ |\ c \Rightarrow M) = [\theta;\varrho](\Xi)\ |\ (c \Rightarrow [\theta;\varrho](M)) \qquad (\mathsf{SBXi})$$

Note that the case SBVar is well-defined because every variable name which the substitution process might encounter is defined in either θ or ϱ. The rules SBLam and SBLet look peculiar because of the extension of the substitution $\theta;\varrho$ by x/x. The insight behind this construction is that we require a substitution to be defined on all free variables of a term. x may occur free in M in the case of SBLam and it may also occur free in M_2 in rule SBLet. Writing x/x implicitly stands for introducing a new variable name x and replacing it for x. We use this notational trick throughout this report.

A crucial rule in our system is SBBox. Since a boxed term is closed it can only contain variables representing closed objects and not variables representing arbitrary objects. This is easily verified by inversion of the TpBox rule because we assume the subject of substitution always to be well-typed. This means, that ϱ will not be used during the substitution process. Hence we need not consider it when traversing over a box.

Our imposed requirement that substitutions must substitute all free variables in an object has further effects. It makes it also necessary to explicitly introduce the notion of identity substitutions because the empty substitution plays the role of an identity substitution only when applied to closed terms. We hence define identity with respect to a context.

Definition 6.15 (Identity substitution:) *Let Γ be a context.*
Rules:

$$\mathrm{id}_{\cdot} = \cdot \qquad (\mathsf{IdEmpty})$$

$$\mathrm{id}_{\Gamma,x{:}A} = \mathrm{id}_{\Gamma}, x/x \qquad (\mathsf{IdNonEmpty})$$

It is not difficult to show that the identity substitution $\mathrm{id}_{\Delta}; \mathrm{id}_{\Gamma}$ is well-typed and hence satisfies

Lemma 6.16 (Well-typedness of identity substitution)

$$\Delta; \Gamma \vdash (\mathrm{id}_{\Delta}; \mathrm{id}_{\Gamma}) \ : \ (\Delta; \Gamma)$$

Proof: omitted. □

Two different notions of substitution have been used in the presentation of our system so far. For the evaluation rules EvApp and EvLet, we used a substitution $[M_1/x](M_2)$ to express that all occurrences of x must be replaced in M_2 by M_1. On the other hand we introduced the notion $(\theta; \varrho)$ in the most recent discussion. It is necessary to examine how those both notions of substititution fit together.

Lemma 6.17 (Property of substitutions)

1. $[M'/x]([\theta; \varrho, x/x](M)) = [\theta; \varrho, M'/x](M)$
2. $[M'/x]([\theta, x/x; \varrho](M)) = [\theta, M'/x; \varrho](M)$

Proof: omitted. □

Weakening is also admissible for the typing judgment of substitutions. Without proof we just state the result:

Lemma 6.18 (Weakening on typing substitution relation)

1. *If* $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *and* $\Delta'' \geq \Delta'$ *then* $\Delta''; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$
2. *If* $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *and* $\Gamma'' \geq \Gamma'$ *then* $\Delta'; \Gamma'' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$

Proof: omitted. □

By induction we can prove that restricting a well-typed substitution $(\theta; \varrho)$ to $(\theta; \cdot)$ is also well-typed. If the domain of $(\theta; \varrho)$ is $(\Delta; \Gamma)$, the domain of the restricted substitution is clearly $(\Delta; \cdot)$. Less clear is that we can also restrict the context $\Delta'; \Gamma'$ of the original well-typed substitution to $\Delta'; \cdot$ because ϱ cannot depend on any variables from Γ'.

Lemma 6.19 (Modal substitution restriction)

If $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *then* $\Delta'; \cdot \vdash (\theta; \cdot) \ : \ (\Delta; \cdot)$

Proof: See appendix B. □

Another important observation is that if a variable name defined in the context $\Delta; \Gamma$ is encountered while substituting, we can determine the type and the new context of the substituted object. In the case the encountered variable name is defined in the modal context, we can restrict the second component of the context to $\cdot$.

Lemma 6.20 (Properties of typing relation for substitutions)

1. *If* $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *then* $\theta(x) = M$ *and* $\Delta'; \cdot \vdash M : A$
2. *If* $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *then* $\varrho(x) = M$ *and* $\Delta'; \Gamma' \vdash M : A$

Proof: See appendix B. □

These preparatory results lead us to a general substitution lemma for the typing relation. If an object M is well-typed in a context for which a substitution is well-defined then its application to M yields an object of the same type as M in the context of the substitution. Since objects are also defined in terms of term replacements and matches we must extend the result to both construction as we show be induction over the typing relations.

Lemma 6.21 (Substitution lemma for typing relation)
Let $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$, *then the following holds:*

1. *If* $\Delta; \Gamma \vdash M : A$ *then* $\Delta'; \Gamma' \vdash [\theta; \varrho](M) : A$
2. *If* $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma')$ *then* $\Delta'; \Gamma' \vdash [\theta; \varrho](\Xi) : \langle B \Rightarrow A \rangle(\Sigma')$
3. *If* $\Delta; \Gamma \vdash \Omega : \langle \omega \rangle(\Sigma')$ *then* $\Delta'; \Gamma' \vdash [\theta; \varrho](\Omega) : \langle \omega \rangle(\Sigma')$

Proof: See appendix B. □

As corollary we can apply the substitution for typing relation to the identity substitution as inferred by Lemma 6.16.

Corollary 6.22 (Well-typedness of application of identity substitution)

If $\Delta; \Gamma \vdash M : A$ *then* $\Delta; \Gamma \vdash [\mathrm{id}_\Delta; \mathrm{id}_\Gamma](M) : A$

Proof: omitted. □

It follows now by a short inductive argument that the identity substitution behaves as expected. Applied to an object M it returns M.

Lemma 6.23 (Identity substitution) *Let* Δ, Γ *contexts.*

If $\Delta; \Gamma \vdash M : A$ *then* $[\mathrm{id}_\Delta; \mathrm{id}_\Gamma](M) = M$

Proof: omitted. □

6.4 Atomic and canonical forms

For atomic and canonical form judgments there is also a weakening result. The proof goes by mutual induction over the derivation of atomic and canonical forms. It is straightforward, and we omit it here.

Lemma 6.24 (Weakening for atomic/canonical forms)

1. *If* $\Psi \vdash V \downarrow A$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash V \downarrow A$
2. *If* $\Psi \vdash V \Uparrow A$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash V \Uparrow A$

Proof: omitted. □

A slightly more complicated property of atomic and canonical forms is that the type of an object can be directly inferred from the judgment. With an easy proof by mutual induction over the derivation of the canonical and the atomic form judgment we prove the following lemma:

Lemma 6.25 (Type preservation of atomic and canonical types)

1. *If* $\Psi \vdash V \downarrow B$ *then* $\cdot; \Psi \vdash V : B$
2. *If* $\Psi \vdash V \Uparrow B$ *then* $\cdot; \Psi \vdash V : B$

Proof: See appendix B. □

6.5 Evaluation

For the evaluation judgments there exists also a weakening result. The proof goes by induction over the evaluation derivation. It is so easy that we omit it here.

Lemma 6.26 (Weakening for atomic/canonical forms)

If $\Psi \vdash M \hookrightarrow V : A$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash M \hookrightarrow V : A$

Proof: omitted. □

6.6 Subordination of types

In this subsection we discuss basic properties which are related to the subordination of types. Recall that the subordination relation accounts for all dependencies which are introduced by the signature or by the subject type of iteration or case.

The first property is needed in some of the lemmas preceeding the canonical form theorem. We require that the target types are always of atomic type. The proof is easy because intuitively the target type is defined to be the last atomic type occurring in an arbitrarily formed pure type. Therefore we do not feel too guilty in omitting the proof.

Lemma 6.27 (Properties of target types)

$\tau(B) = a$

for some atomic type a.

Proof: omitted. □

We will also need that a target type of some type B does not change regardless if B is embedded in some abstraction closure. The proof goes by induction over the context defining the abstraction closure.

Lemma 6.28 (Goal types and abstraction closure types)

$\tau(\Pi\{\Psi\}.\, B) = \tau(B)$

Proof: See appendix B. □

Before we start with the discussion of the subordination we further refine the notion of subordination. So far $a_1 \blacktriangleleft_{\Sigma;B} a_2$ expresses that objects of type a_1 can be used to construct objects of type a_2. In Section 4 we have seen that the subordination relation is not reflexive. The simplest examples are the Booleans from Example 4.6. bool is not recursive, hence it doesn't hold that bool $\blacktriangleleft_{\Sigma;B}$ bool. A closer look reveals, that the subordination relation for bool is empty. But it is definitely not the case that bool is. To account for this observation we extend the notion of subordination relation. If $a \trianglelefteq_{\Sigma;B}$ bool holds, then objects of type a can occur as objects or subobjects of objects of type bool. We call this the *weak* subordination relation.

Definition 6.29 (Weak subordination relation) *Let B be a pure type.*

$$a_1 \trianglelefteq_{\Sigma;B} a_2 :\Leftrightarrow a_1 \blacktriangleleft_{\Sigma;B} a_2 \text{ or } a_1 = a_2$$

In the remainder of this section we characterize and discuss a few major properties of type subordination which will prove very useful when we tackle the proof of the canonical form theorem.

First we want to point out that there is a close relationship between source types of a constructor type, and the subordination relation: This relationship can be characterized by the following observation: Every source type of the constructor type is trivially subordinated by the goal type of the constructor. A second important property is a transitivity property of the weak subordination relation: If a type a_1 is subordinated by a type a_2 and a_2 is weakly subordinated by a type a_3, then a_1 is automatically subordinated by a_3. Note that this formulation of the lemma is slightly more stronger than just assuming a_2 to be subordinated by a_3. In this case the result follows trivially from the definition 4.24 of subordination.

Lemma 6.30 (Properties of subordination) *Let $c : C \in \Sigma$, B a pure type.*

1. *If $a \in Source(C)$ then $a \blacktriangleleft_B \tau(C)$*
2. *If $a_1 \blacktriangleleft_B a_2$ and $a_2 \trianglelefteq_B a_3$ then $a_1 \blacktriangleleft_B a_3$*

Proof: See appendix B. □

Variables introduced by a context can be interpreted as a set of pseudo constructors or parameters as we pointed out in Section 4. Since pseudo constructors are variable names which are represented in a context, we must extend the notion subordination to contexts: For all pseudo constructors types B' if the target type $\tau(B')$ subordinates a, all source types of B' must also subordinate a.

Definition 6.31 (Subordination on contexts) *Let a be an atomic type.*

$$\cdot \blacktriangleleft_B a$$

$$\Psi, x : B' \blacktriangleleft_B a \quad \Leftrightarrow \quad \Psi \blacktriangleleft_B a \textit{ and if } \tau(B') \trianglelefteq_B a \textit{ then forall } \; y \in \textit{Source}(B') : y \blacktriangleleft_B a$$

It will become clear during the proof of the canonical form theorem, how context subordination is used. Here is what we need for the proof: If a variable x of type B' is defined in a context Ψ and $\Psi \blacktriangleleft_B \tau(B)$, then all source types of type B' are automatically subordinated by the goal type of of B. The proof is an easy induction over the context Ψ.

Lemma 6.32 (Properties of context subordination) *Let B be a pure type.*

If $\Psi = (\Psi_1, x : B') \cup \Psi_2$ and $\Psi \blacktriangleleft_B \tau(B)$ then $\tau(B') \trianglelefteq_B \tau(B)$ implies that forall $y \in \textit{Source}(B')$: $y \blacktriangleleft_B \tau(B)$

Proof: See appendix B. □

The next result has to do with pseudo constructor types. By easy induction it can be shown, that if B' is a pseudo constructor introduced by a type B, then all source types of B' must be also source types of B. This is clear, because every pseudo constructor type corresponds to a domain type of B and B must be a function type. Every source type of B' is hence a source type of B. We omit the proof.

Lemma 6.33 (Subset property of PCT)

Forall $B' \in PCT(B)$ the following holds:

$$\textit{Source}(B') \subseteq \textit{Source}(B)$$

Proof: omitted. □

Pseudo constructor types have also another property. This property has to do with dynamic type subordination: If a pseudo constructor type B' of a type B is given, and a is a type which is immediately subordinated by the target type of B', then we have $a \lhd_B \tau(B')$, because every source type of B' is also a source type of B. The proof is done by induction over type B.

Lemma 6.34 (Property of dynamic typing)

If $B' \in PCT(B)$ then $a <_{B'} \tau(B')$ implies $a \lhd_B \tau(B')$

Proof: See appendix B. □

The last three lemmas in this subsection answer an important question which will be raised in the proof of the canonical form theorem. Without going into details the property we need is as follows. Assume B, C are two pure types. We must show that if the target type of B is not subordinated by the target type of C then $\langle\omega\rangle(C) = C$. We will try to shed some light on this problem. While traversing a subject of type B iteration may encounter constants of type C whose target type does not occur in the inductive datatype $\mathcal{I}(\Sigma; B)$ (see Example 4.8, Example 4.5). This information can already be extracted from the subordination relation. If a constant c of type C is encountered during the traversal, $\tau(C) \blacktriangleleft_{\Sigma;B} \tau(B)$ must hold – as we prove below. If the reverse ($\tau(B) \blacktriangleleft_{\Sigma;B} \tau(C)$) also holds, $\tau(C)$ must an element in $\mathcal{I}(\Sigma; B)$ by definition. This might not always be the case. In Example 4.8 nat $\notin \mathcal{I}(\Sigma; \mathrm{db})$, in example 4.5 i $\notin \mathcal{I}(\Sigma; \mathrm{o})$. Constants with a target type not being an element of the inductive type remain untouched by the process of elimination. Is the result of the elimination process $\langle\omega; \Omega\rangle(M)$ still well-typed? We must require that ω applied to a constructor type C with target type outside of $\mathcal{I}(\Sigma; B)$ is mapped to C.

Thus, more formally, we must show that if $\tau(C) \notin \mathcal{I}(\Sigma; B)$ then $\langle\omega\rangle(C) = C$. We split this proof into two parts. The first part we present here, that is we show that if $\tau(B)$ is not subordinated by $\tau(C)$ the claim is fulfilled. The second part will be mainly discussed in the proof of the canonical form theorem but we show an auxiliary result in this section. The idea behind the second part of the proof is to show that the case "$\tau(C)$ is not subordinated by $\tau(B)$" cannot occur. For the first part of the proof we show two lemmas.

1. If $\tau(B) \not\blacktriangleleft_B \tau(C)$ then $\mathrm{Source}(C) \cap \mathcal{I}(\Sigma; B) = \emptyset$
2. If $\mathrm{Source}(C) \cap \mathcal{I}(\Sigma; B) = \emptyset$ and $\tau(C) \notin \mathcal{I}(\Sigma; B)$ then $\langle\omega\rangle(C) = C$

If the target type of B is not subordinated, then clearly none of its (atomic) source types is a member of the the inductive datatype. If there would actually be one atomic type, being an element of the inductive datatype and a source type of C, then this atomic source type would subordinate the target type of B and hence our assumption would be violated. We show this claim directly.

Lemma 6.35 (Independence)

If $\tau(B) \not\blacktriangleleft_B \tau(C)$ then Source$(C) \cap \mathcal{I}(\Sigma; B) = \emptyset$

Proof: See appendix B. □

The second property ensures that if there isn't any source type of C in the domain of a type replacement, the type replacement does not have any effect on C. We show this claim by induction over the constructor type.

Lemma 6.36 (Properties of Join) *Let $c : C \in \Sigma$, α arbitrary and $\vdash \omega : \alpha$*

If Source$(C) \cap \alpha = \emptyset$ and $\tau(C) \notin \alpha$ then $\langle\omega\rangle(C) = C$

Proof: See appendix B. □

The auxiliary lemma for the second part states, that if the target type of a constructor type C does not occur in the inductive datatype $\mathcal{I}(\Sigma; B)$ but the target type of C is either equal or subordinated by the target type of B, then it is impossible that $\tau(B)$ is subordinated by $\tau(C)$.

Lemma 6.37 (Properties of subordination)

If $\tau(C) \notin \mathcal{I}(\Sigma; B)$ *and* $\tau(C) \trianglelefteq_B \tau(B)$ *then* $\tau(B) \not\triangleleft_B \tau(C)$

Proof: See appendix B. □

This concludes the section of the basic preliminary results. In the next section we will address the problem of the existence of canonical forms for typed objects.

7 Canonical form theorem

The aim of this section is to prove the canonical form property of the modal λ-calculus. The main result will be that every object of pure type in a pure context possesses a canonical form. In our notation this property is expressed as: if $\cdot;\Psi \vdash M : B$ then $\Psi \vdash M \Uparrow V : B$. This result implies the conservative extension property of $\lambda^{\Box}$ which we will show in section 9. We prove this by Tait's method, often called an argument by logical relations or reducibility candidates. In such an argument we construct an interpretation of types as a relation between objects, in our case a unary relation. Assume we are trying to establish a property P of all well-typed objects of type B. In our case P holds if there is an object V s.t. $\Psi \vdash M \Uparrow V : B$. The proof using logical relation proceeds then in two steps. In the first step the object which should satisfy P must be proven to be a member in the logical relation. Finally we prove by induction that for every member in the logical relation the property P holds.

Before we go into details how the logical relation is defined, we derive some useful lemmas, which are necessary for the argument. Some of the following proofs rely on the fact, that canonical and atomic forms evaluate to themselves. This fact, even though it might seem trivial, requires a mutual inductive argument: To prove that atomic and canonical forms evaluate to themselves we require that the notion of evaluation implies somehow the notion of evaluation to a canonical form: If M evaluates to V and V happens to be canonical, then M evaluates canonically to V. These properties are expressed by the following

Lemma 7.1 (Self evaluation)

1. *If* $\Psi \vdash M \hookrightarrow V : B$ *and* $\Psi \vdash V \Uparrow B$ *then* $\Psi \vdash M \Uparrow V : B$
2. *If* $\Psi \vdash V \Uparrow B$ *then* $\Psi \vdash V \hookrightarrow V : B$
3. *If* $\Psi \vdash V \downarrow B$ *then* $\Psi \vdash V \hookrightarrow V : B$

Proof: See appendix C. □

Another result which seems intuitively clear but must be proven is the following: by definition objects evaluate to other objects under the judgment $\Psi \vdash M \Uparrow V : B$. Since this is the judgment for canonical evaluation, we expect V to be canonical. That this holds is expressed by the next lemma. Contrary to the intuition, the proof is not straightforward since the notion of conversion to canonical forms depend on the evaluation judgment. On the other hand, it is also not very sensible to try to prove that for every object M, $\Psi \vdash M \hookrightarrow V : A$ implies that V is a canonical form. For example, consider the signature from example 2.5: It is easy to see that

$$\cdot \vdash \lambda x\!:\!\mathrm{exp}.\,(\lambda y\!:\!\mathrm{exp}.\,y)\ \mathrm{z} \hookrightarrow \lambda x\!:\!\mathrm{exp}.\,(\lambda y\!:\!\mathrm{exp}.\,y)\ \mathrm{z} : \mathrm{exp} \to \mathrm{exp}$$

but it is also clear, that $\lambda x\!:\!\mathrm{exp}.\,(\lambda y\!:\!\mathrm{exp}.\,y)\ \mathrm{z}$ is not canonical because the body of the λ-expression can be β-reduced. However, it holds when restricted to atomic types:

Lemma 7.2 (Property of evaluation results)

1. *If* $\Psi \vdash M \Uparrow V : B$ *then* $\Psi \vdash V \Uparrow B$

2. If $\Psi \vdash M \hookrightarrow V : a$ *then* $\Psi \vdash V \downarrow a$

Proof: See appendix C. $\square$

Consider example 2.5 again. The constant app is not a canonical form either. The reason is that canonical forms are actually objects in η-long β-normal form. app can be easily transformed into such a form: $\lambda x\!:\!\text{exp}.\, \lambda y\!:\!\text{exp}.\, \text{app}\; x\; y$. According to lemma 7.2 (2) the result of an evaluation is canonical only if it is an object of atomic type. Nothing is said about functions. Fortunately we can prove that every object M evaluating to an atomic form V necessarily evaluates to a canonical form V':

Lemma 7.3 (Evaluation to atomic forms implies evaluation to canonical forms)

If $\Psi \vdash M \hookrightarrow V : B$ *and* $\Psi \vdash V \downarrow B$ *then* $\Psi \vdash M \Uparrow V' : B$ *for a* V'

Proof: See appendix C. $\square$

These three lemmas are necessary for some of the proofs below. We address now the definition of the logical relation. Recall that the logical relation is a set of objects satisfying a certain property specified by a type A and a context Ψ. For our system we introduce two logical relations. The logical relations of objects evaluating to some other object, and the logical relation of values. For the first we write $\Psi \vdash M \in [\![A]\!]$ to express that object M satisfies the relation $[\![A]\!]$ in context Ψ. Similary for the logical relation of values. We write $\Psi \vdash V \in |A|$ meaning that value V satisfies the relation $|A|$ in context Ψ. Both relations are defined by structural induction over A.

Definition 7.4 (Logical relation)

$\Psi \vdash M \in [\![A]\!] :\Leftrightarrow \cdot;\Psi \vdash M : A$ *and* $\Psi \vdash M \hookrightarrow V : A$ *and* $\Psi \vdash V \in |A|$

$\Psi \vdash V \in |A| :\Leftrightarrow$

Case: $A = a$ *and* $\Psi \vdash V \Uparrow a$

Case: $A = A_1 \to A_2$ *and either*

Case: $V = \lambda x\!:\!A_1.\, M$ *and for all* $\Psi' \geq \Psi$*:* $\Psi' \vdash V' \in |A_1| \Rightarrow \Psi' \vdash [V'/x](M) \in [\![A_2]\!]$

or

Case: $\Psi \vdash V \downarrow A_1 \to A_2$ *and for all* $\Psi' \geq \Psi$*:* $\Psi' \vdash V' \Uparrow A_1 \Rightarrow \Psi' \vdash V\; V' \in |A_2|$

Case: $A = A_1 \times A_2$ *and* $V = \langle M_1, M_2\rangle$ *and* $\Psi \vdash M_1 \in [\![A_1]\!]$ *and* $\Psi \vdash M_2 \in [\![A_2]\!]$

Case: $A = \Box A'$*:* $V = \mathit{box}\; M$ *and* $\cdot \vdash M \in [\![A']\!]$

Note that the first logical relation requires its elements to be well-typed. This is necessary because our argument requires that all objects in the relation are well-typed as we will see in Lemma 7.47. $\Psi \vdash M \in [\![A]\!]$ must imply that M has that type A in $\cdot;\Psi$. In lemma 7.19 we show that this property propagates to the logical relation of values.

Objects were defined in terms of term replacements and matches (see Section 4, Section 5). Later on in this section we will need to show that every object defined in a term replacement or match is a member of a logical relation. To make our presentation of this circumstance cleaner and easier to understand we will introduce the notion of logical relations for term replacements. A term replacement is an element of the logical relation defined by a signature Σ and a context representing pseudo constructors $\hat{\Psi}$, if every object associated with a (pseudo) constructor satisfies the logical relation defined by the type which results from applying the type replacement ω — defined by the iterator object — to the (pseudo) constructor type.

Definition 7.5 (Logical relation for term replacements) $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!] :\Leftrightarrow$

Case: *If* $\hat{\Psi} = \cdot$ *and* $\Sigma = \cdot$ *then* $\Omega = \cdot$

Case: *If* $\hat{\Psi} = \cdot$ *and* $\Sigma = \Sigma', c : B$ *then* $\Omega = \Omega' \mid c \mapsto M$ *and* $\Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ *and* $\Psi + \tilde{\Psi} \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma'; \cdot)]\!]$

Case: *If* $\hat{\Psi} = \hat{\Psi}', x : B$ *then* $\Omega = \Omega' \mid x \mapsto u$ *and* $\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ *and* $\Psi + \tilde{\Psi} \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi}')]\!]$

The context defined for the logical relation of term replacements is split into two parts $\Psi; \tilde{\Psi}$. Ψ represents the context of variables which might occur free in the objects associated with constructors (note: *not* pseudo constructors), and $\tilde{\Psi}$ stands for the context of newly defined variables which rename the original pseudo constructors. We must keep both contexts separately, because to prove lemma 7.46 and lemma 7.43 we define a substitution, which acts as the identity on all variables defined in Ψ, but not necessarily on $\tilde{\Psi}$. Not distinguishing between both contexts of variables would mean to discard this information — which we will need.

We have seen that every object in the logical relation $[\![A]\!]$ is well-typed. This property propagates to term replacements. With an easy inductive proof we can show that

Lemma 7.6 (Type preservation for term replacements) *If* $\Psi + \cdot \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \cdot)]\!]$ *then* $\cdot; \Psi \vdash \Omega : \langle\omega\rangle(\Sigma)$

Proof: See appendix C. □

Similarly we define the logical relation for matches. A match is an element of the logical relation defined by a signature Σ and a context representing pseudo constructors $\hat{\Psi}$, if every object associated with a (pseudo) constructor satisfies the logical relation defined by the case type of the (pseudo) constructor type. Because of the same reasons as for the term replacement we define the logical relation using two contexts: $\Psi; \tilde{\Psi}$. Here is the definition.

Definition 7.7 (Logical relation for matches) $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!] :\Leftrightarrow$

Case: *If* $\hat{\Psi} = \cdot$ *and* $\Sigma = \cdot$ *then* $\Xi = \cdot$

Case: *If* $\hat{\Psi} = \cdot$ *and* $\Sigma = \Sigma', c : B'$ *then* $\Xi = \Xi' \mid c \Rightarrow M$ *and* $\Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$ *and* $\Psi + \tilde{\Psi} \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$

Case: *If* $\hat{\Psi} = \hat{\Psi}', x : B'$ *then* $\Xi = \Xi' \mid x \Rightarrow u$ *and* $\tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$ *and* $\Psi + \tilde{\Psi} \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi}')]\!]$

We have seen that every object in the logical relation $[\![A]\!]$ is well-typed and so is every term replacement. As we might expect, this property can also be shown for matches. By simple induction we obtain:

Lemma 7.8 (Type preservation for matches) *If* $\Psi + \cdot \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \cdot)]\!]$ *then* $\cdot; \Psi \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma)$

Proof: See appendix C. $\square$

We start now with the discussion of the logical relation. The next few lemmas show some useful properties implied by this definition, all necessary to eventually prove the canonical form theorem. The first lemma is a standard weakening lemma for logical relations:

Lemma 7.9 (Weakening for logical relations)

1. *If* $\Psi \vdash M \in [\![A]\!]$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash M \in [\![A]\!]$
2. *If* $\Psi \vdash V \in |A|$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash V \in |A|$

Proof: See appendix C. $\square$

We must extend this result to logical relations for term replacements. Recall that the logical relation for term replacements is defined using two separate contexts $\Psi; \tilde{\Psi}$. For our purposes it is enough to prove weakening as an extension of context $\tilde{\Psi}$.

Lemma 7.10 (Weakening for logical relations for replacement)

If $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *and* $\tilde{\Psi}' \geq \tilde{\Psi}$ *then* $\Psi + \tilde{\Psi}' \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$

Proof: See appendix C. $\square$

The logical relation for matches was defined analogously to the logical relation of term replacements. As expected the formulation of the weakening property is similar to the previous one.

Lemma 7.11 (Weakening for logical relations for matches)

If $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ *and* $\tilde{\Psi}' \geq \tilde{\Psi}$ *then* $\Psi + \tilde{\Psi}' \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$

Proof: See appendix C. $\square$

The logical relations of term replacements and matches play a very important role when we discuss the elimination process. Recall from Definition 4.28 that this process traverses the structure of the subject of iteration. Eventually constants or variables will be encountered. We will see in the proof of Lemma 7.43 that the term replacement Ω is then an element of the corresponding logical relation. The problem reduces to looking up the encountered constant or variable in Ω. The attentive reader has probably already recognized that three cases might occur.

- A constructor has been encountered which is defined by Ω.
- A constructor has been encountered which has not been defined by Ω.
- A pseudo constructor has been encountered which must be defined in Ω.

The third case speaks of "must be defined in Ω" because the subject of iteration was closed, every traversed λ-abstraction has extended Ω by an appropriate variable renaming. We discuss now each of those three cases by showing three lemmas.

If $c : B$ is the encountered constructor which happens to be defined in Σ, the domain of the logical relation for replacement, then $\langle\omega; \Omega\rangle(c)$ is of correct type and in the logical relation $[\![\langle\omega\rangle(B)]\!]$.

Lemma 7.12 (Access to logical relations for replacements I) *If* $\Sigma = \Sigma_1, c : B \cup \Sigma_2$ *and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ *and* $M = \langle\omega; \Omega\rangle(c)$

Proof: See appendix C. □

In the case that $c : B$ is not defined in this signature, then $\Omega(c)$ is undefined.

Lemma 7.13 (Access to logical relations for replacements II) *If* $\Sigma(c)$ *is undefined and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Omega(c)$ *is undefined*

Proof: See appendix C. □

In the case that the traversal of the iteration encountered a pseudo constructor $x : B$ defined in $\hat{\Psi}$, x is being renamed by the term replacement to a new variable name u, which happens to be an element of $[\![\langle\omega\rangle(B)]\!]$.

Lemma 7.14 (Access to logical relations for replacements III) *If* $\hat{\Psi} = \hat{\Psi}_1, x : B \cup \hat{\Psi}_2$ *and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ *and* $\tilde{\Psi} = \tilde{\Psi}_1, u : \langle\omega\rangle(B) \cup \tilde{\Psi}_2$ *and* $u = \langle\omega; \Omega\rangle(x)$

Proof: See appendix C. □

We have a very similar situation for matches. Recall from Definition 5.15 that the process of selection traverses the subject of case to find its head constructor. In the proof of Lemma 7.46 we will have that Ξ is in the logical relation of matches. Thus we must look up the head constructor in Ξ. Contrary to the term replacement only two cases can occur, because the case object must have been well-typed.

- A constructor is the head constructor which is defined in Ξ
- A pseudo constructor is the head constructor which is defined in Ξ.

We discuss now each of the cases by showing two lemmas. If $c : B'$ is the head constructor, it is accounted for in Ξ and $\{B \Rightarrow A; \Xi; \Psi\}(c)$ is of correct type and an element in the logical relation $[\![\mathcal{C}\ (B, A, B')]\!]$.

Lemma 7.15 (Access to logical relations for matches I) *If* $\Sigma = \Sigma_1, c : B' \cup \Sigma_2$ *and* $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$ *and* $M = \{B \Rightarrow A; \Xi; \Psi'\}(c)$ *for an arbitrary* Ψ'.

Proof: See appendix C. □

On the other hand, if $x : B'$ is the head constructor, it is also accounted for in Ξ and $\{B \Rightarrow A; \Xi; \Psi\}(x)$ is of correct type and an element in the logical relation $[\![\mathcal{C}\ (B, A, B')]\!]$.

Lemma 7.16 (Access to logical relations for matches II) *If $\hat{\Psi} = \hat{\Psi}_1, x : B' \cup \hat{\Psi}_2$ and $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ then $\tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$ and $\tilde{\Psi} = \tilde{\Psi}_1, u : \mathcal{C}\ (B, A, B') \cup \tilde{\Psi}_2$ and $u = \{B \Rightarrow A; \Xi; \Psi'\}(x)$ for an arbitrary Ψ'.*

Proof: See appendix C. □

The principal lemmas we need for the proof of the canonical form theorem are the following. Every well-typed object is an element of the logical relation defined by its type, and every element of a logical relation has a canonical form. More formally:

1. If $\cdot; \Psi \vdash M : A$ then $\Psi \vdash M \in [\![A]\!]$
2. If $\Psi \vdash M \in [\![B]\!]$ then $\Psi \vdash M \Uparrow V : B$

In this presentation we first show the second lemma. To prove it, we must generalize its formulation. The proof depends on the fact that atomic objects of pure types B are always in the logical relation of values $|B|$. By mutual induction we can then show the following lemma:

Lemma 7.17 (Logical relations and canonical forms)

1. *If $\Psi \vdash M \in [\![B]\!]$ then $\Psi \vdash M \Uparrow V : B$*
2. *If $\Psi \vdash V \downarrow B$ then $\Psi \vdash V \in |B|$*

Proof: See appendix C. □

In Section 3 we introduced arbitrary types opposed to pure types from Section 2. We show now by an easy inductive argument that if an object is atomic of some type A in a pure context, then A must be necessarily pure.

Lemma 7.18 (Types of atomic objects are pure)

If $\Psi \vdash V \downarrow A$ then A is pure.

Proof: See appendix C. □

This lemma is necessary for the proof of the well-typedness of objects in the logical relation of values, as briefly discussed above. We have seen that every object in $[\![A]\!]$ is of type A. We show now that every object in relation $|A|$ is also well-typed.

Lemma 7.19 (Well-typedness of logical relations)

If $\Psi \vdash V \in |A|$ *then* $\cdot; \Psi \vdash V : A$

Proof: See appendix C. $\square$

But this is not the only property objects satisfying relation $|A|$ enjoy. Based on the self evaluation lemma 7.1, it is now easy to show that every object in $|A|$ evaluates to itself. The proof goes by structural induction over type A.

Lemma 7.20 (Logical relations: Self evaluation of values)

If $\Psi \vdash V \in |A|$ *then* $\Psi \vdash V \hookrightarrow V : A$

Proof: See appendix C. $\square$

A direct consequence of these two lemmas is that every object in $|A|$ is also an object in $[\![A]\!]$. This is a result which we use quite often in proofs of the subsequent lemmas. We state this result in form of a lemma:

Lemma 7.21 (Logical relation subsumption)

If $\Psi \vdash V \in |A|$ *then* $\Psi \vdash V \in [\![A]\!]$

Proof: See appendix C. $\square$

Recall that all the lemmas which are presented here serve the purpose to prove the first of both lemmas necessary for the proof of the canonical form theorem. We shall now introduce more pieces to complete the mosaic. To prove the first lemma we need to show that every typable object of type A satisfies the relation $[\![A]\!]$. Consider a typing derivation ending with the typing rule TpApp. We see that the result object of the rule is an application $M_1\ M_2$. M_1 is a function, M_2 is the parameter object of suitable type. The next lemma shows that it is legitimate to establish a similar way of reasoning with logical relations. If M_1 satisfies the logical relation created by a function type $A_2 \to A_1$ and M_2 satisfies the logical relation created by the domain of the function type, namely $[\![A_2]\!]$, then $(M_1\ M_2)$ satisfies $[\![A_1]\!]$. The proof of this lemma is fairly straightforward and makes use of lemma 7.17 and lemma 7.2. This property will be very useful for the proof of the canonical form theorem.

Lemma 7.22 (Logical relation is closed under application)

If $\Psi \vdash M_1 \in [\![A_2 \to A_1]\!]$ *and* $\Psi \vdash M_2 \in [\![A_2]\!]$ *then* $\Psi \vdash M_1\ M_2 \in [\![A_1]\!]$

Proof: See appendix C. $\square$

Recall that the proof of the canonical form theorem is performed in two steps.

1. If $\Psi \vdash M \in [\![B]\!]$ then $\Psi \vdash M \Uparrow V : B$

2. If $\cdot; \Psi \vdash M : A$ then $\Psi \vdash M \in [\![A]\!]$

We already proved the first step by slightly generalizing the lemma. The second lemma cannot be proven without generalization either. The problem we encounter when we try to prove it directly is that the context Ψ may grow during the typing process (TpLam). It is also possible that the modal context which is empty in the current formulation of the lemma does not remain empty throughout the typing derivation (TpLet).

To successfully prove this lemma we must generalize its formulation by using substitutions. Given a typing derivation $\Delta; \Gamma \vdash M : A$ and a substitution for $\Delta; \Gamma$, where the objects introduced by the substitution might depend on free variables from a context Ψ, we can show that the substituted object $[\theta; \varrho](M)$ is indeed an object in $[\![A]\!]$. The desired result is a consequence of this generalized lemma using the identity substitution introduced by definition 6.15.

We are still far away from proving this generalized lemma, some key lemmas must still be developed and proven. In the remainder of this section, we address the following issues: First we introduce logical relations for contexts. Substitutions are elements of such a logical relation. Some more technical results are necessary, which will be discussed right after this definition. Second, we need auxiliary lemmas for the elimination and the selection judgment. These auxiliary lemmas obviously depend on type replacements and (complete) case types. Some additional reasoning is necessary to show that the treatment of constants during the elimination process does not destroy the type preservation property. Many lemmas which are needed to establish this claim have been already discussed in section 6. Finally we assemble all these pieces to obtain a generalized version of the second half of the canonical element theorem.

Let us start with the description of the basic ingredients. The first ingredient is the notion of logical relation for modal and arbitrary contexts. It follows from the previous discussion, that substitutions are necessary to generalize the formulation of the lemma in question. We are given a typing derivation $\mathcal{D} :: \cdot; \Psi \vdash M : A$. Some subderivation of $\mathcal{D}$ might be of the form $\Delta; \Gamma \vdash M' : A'$. Hence M' can contain free variables from Δ and from Γ. The logical relation which we define now contains all substitutions $\theta; \varrho$, with the following properties:

1. If $\theta(x) = M$ where $x : A$ is defined in the context Δ, then M must be a closed object of type A. For our purposes we also must require that M is actually a closed object satisfying $[\![A]\!]$.

2. If $\varrho(x) = M$ where $x : A$ is defined in the context Γ, then M must be an object satisfying $|A|$.

The formal definition of logical relation for modal/arbitrary contexts follows directly. We define three logical relations. The first two correspond to these two properties, the third is a combination of both.

Definition 7.23 (Logical relation for modal contexts) $\vdash \theta \in [\![\Delta]\!] :\Leftrightarrow$

Case: *If* $\Delta = \cdot$ *then* $\theta = \cdot$

Case: *If* $\Delta = \Delta', x : A$ *then* $\theta = \theta', M/x$ *and* $\cdot \vdash M \in [\![A]\!]$ *and* $\vdash \theta' \in [\![\Delta']\!]$

Differently from this definition, the second logical relation must account for the fact that objects occurring in the substitution may depend on free variables from a context Ψ. Hence the context Ψ must be involved in the definition.

Definition 7.24 (Logical relation for regular contexts) $\Psi \vdash \varrho \in |\Gamma| :\Leftrightarrow$

Case: *If* $\Gamma = \cdot$ *then* $\varrho = \cdot$

Case: *If* $\Gamma = \Gamma', x : A$ *then* $\varrho = \varrho', V/x$ *and* $\Psi \vdash V \in |A|$ *and* $\Psi \vdash \varrho' \in |\Gamma'|$

As described earlier in this paper, we prefer to see the context Δ and the context Γ as a combined context. In this sense, it is useful to define a combined logical relation, which contains both, the logical relation for Δ and the one for Γ. Here is the formal definition.

Definition 7.25 (Logical relation for combined contexts)

$$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma] \ \ \textit{iff} \ \ \vdash \theta \in [\![\Delta]\!] \ \textit{and} \ \Psi \vdash \varrho \in |\Gamma|$$

Weakening must be also proven for the logical relation for modal/regular context. We omit the easy proof by induction:

Lemma 7.26 (Weakening on logical relation on contexts)

1. *If* $\Psi \vdash \varrho \in |\Gamma|$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash \varrho \in |\Gamma|$
2. *If* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash \theta; \varrho \in [\Delta; \Gamma]$

Proof: omitted. □

The logical relation for contexts was defined to make a generalization of the second part of the canonical form theorem feasible. Looking at the typing rules in Section 3, Section 4, and Section 5, we observe that the new Δ in each premiss is always an extension of the Δ in the conclusion. But this doesn't hold for the context Γ. In the rule TpBox for example we see, that the context Γ is discarded. Let $\theta; \varrho$ be a substitution satisfying the logical relation $[\Delta; \Gamma]$ where $\Delta; \Gamma$ is the context of some typing derivation ending with rule TpBox. The question which arises immediately is whether $\theta; \varrho$ can be restricted in some way to be also an element of $[\Delta; \cdot]$? As side condition we must require Ψ to be empty — as will become clear when we discuss the case TpBox in lemma 7.47. The answer is yes: Choose the new substitution to be $\theta; \cdot$. The proof again is an easy induction.

Lemma 7.27 (Modal substitution restriction)

If $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\cdot \vdash \theta; \cdot \in [\Delta; \cdot]$

Proof: See appendix C. □

We address now the question if every substitution $\theta; \varrho$ which satisfies a logical relation $[\Delta; \Gamma]$ is well-formed with respect to Definition 6.11. The answer is yes, the proof is easy if we consider the modal part $\theta; \cdot$ and the regular part $\cdot; \varrho$ one by one. We show this property in four parts. First we show that $\theta; \cdot$ is well-formed in $\Delta; \cdot$:

Lemma 7.28 (Well-typedness of modal substitutions in logical relations)

If $\vdash \theta \in [\![\Delta]\!]$ *then* $\cdot; \cdot \vdash (\theta; \cdot) \ : \ (\Delta; \cdot)$

Proof: See appendix C. □

Then we show that $\cdot; \varrho$ is well-formed in $\cdot; \Gamma$. It should be clear that the formulation of this lemma must involve Ψ because the objects defined by ϱ might contain free variables from Ψ.

Lemma 7.29 (Well-typedness of regular substitutions in logical relations)

If $\Psi \vdash \varrho \in |\Gamma|$ *then* $\cdot; \Psi \vdash (\cdot; \varrho) \ : \ (\cdot; \Gamma)$

Proof: See appendix C. □

To assemble these both results, we must prove a combination lemma: If $\theta; \cdot$ is well-formed with respect to $\Delta; \cdot$ and $\cdot; \varrho$ is well-formed with respect to $\cdot; \Gamma$ in a context Ψ, then surely $\theta; \varrho$ should be well-formed with respect to $\Delta; \Gamma$ in context Ψ. This is formalized and proved by the following lemma.

Lemma 7.30 (Combination of two substitutions)

If $\cdot; \cdot \vdash (\theta; \cdot) \ : \ (\Delta; \cdot)$ *and* $\cdot; \Psi \vdash (\cdot; \varrho) \ : \ (\cdot; \Gamma)$ *then* $\cdot; \Psi \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$

Proof: See appendix C. □

The previous three lemmas are now being summarized to form the final result. Every substitution in a logical relation $[\Delta; \Gamma]$ is well-typed with respect to context $(\Delta; \Gamma)$.

Lemma 7.31 (Well-typedness of substitutions in logical relations:)

If $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\cdot; \Psi \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$

Proof: See appendix C. □

A rather technical but intuitively clear lemma is the following: If a substitution satisfying the logical relation $|\Delta; \Gamma|$ is given and $x : A$ is defined in the modal context Δ, then the lookup of x in θ will return an object M which fortunately happens to satisfy $[\![A]\!]$.

Lemma 7.32 (Properties of logical relation for modal contexts)

If $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\vdash \theta \in [\![\Delta]\!]$ *then* $\theta(x) = M$ *and* $\cdot \vdash M \in [\![A]\!]$

Proof: See appendix C. □

A similar result holds for $x : A$ being defined in the context Γ.

Lemma 7.33 (Properties of logical relation for regular contexts)

If $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Psi \vdash \varrho \in |\Gamma|$ *then* $\varrho(x) = M$ *and* $\Psi \vdash M \in |A|$

Proof: See appendix C. □

In the following discussion we would like to consider the substitution $\theta; \varrho$ as a unit to simplify the presentation of the proofs. Hence we rephrase the previous two statements by the following lemma.

Lemma 7.34 (Properties of logical relation for contexts)

1. *If* $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\theta(x) = M$ *and* $\cdot \vdash M \in [\![A]\!]$
2. *If* $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\varrho(x) = M$ *and* $\Psi \vdash M \in |A|$

Proof: See appendix C. □

For the same purpose we state the trivial fact of how to extend the regular part of such a substitution pair by some value V, element of the logical relation $|A|$.

Lemma 7.35 (Extending logical relations for contexts) *If* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *and* $\Psi \vdash V \in |A|$ *then* $\Psi \vdash \theta; \varrho, V/x \in [\Delta; \Gamma, x : A]$

Proof: See appendix C. □

After this excursion into the basics of well-typed substitutions and their membership to logical relations, we focus now again on the proof of the canonical form theorem. With logical relations for contexts we are can generalize the lemma leading eventually to the canonical form theorem:

If $\Delta; \Gamma \vdash M : A$ and $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ then $\Psi \vdash [\theta; \varrho](M) \in [\![A]\!]$

To obtain the result we are interested in we must use the identity substitution of context Ψ which supposedly is an instance of $[\cdot; \Psi]$ and apply lemma 7.34. It is necessary to prove that $\Psi \vdash \cdot; \mathrm{id}_\Psi \in [\cdot; \Psi]$.

Because of the definition of logical relation for context $[\cdot; \Psi]$ two lemmas are necessary to prove this lemma. First we show as an auxiliary result that for every Ψ, $\Psi \vdash \mathrm{id}_\Psi \in |\Psi|$. This is not trivial, because the proof relies on lemma 7.17.

Lemma 7.36 (Identity substitution for regular context)

For all Ψ *the following holds:* $\Psi \vdash \mathrm{id}_\Psi \in |\Psi|$

Proof: See appendix C. □

As a second step we bring this lemma into the desired form: $\Psi \vdash \cdot; \mathrm{id}_\Psi \in [\cdot; \Psi]$.

Lemma 7.37 (Identity substitution for context)

For all Ψ *the following holds:* $\Psi \vdash \cdot; \mathrm{id}_\Psi \in [\cdot; \Psi]$

Proof: See appendix C. $\square$

Slowly we are approaching the canonical form theorem. Recall that we are still trying to show that $\cdot; \Psi \vdash M : A$ implies that $\Psi \vdash M \in [\![A]\!]$. The logical relation for contexts, properties of the identity function have been the first steps towards this lemma. Two more challenging problems must be tackled before we can prove it: the role of elimination and selection. This stems from the conclusion of the lemma: An iterator or a case object can only then be in the logical relation $[\![A]\!]$ if they evaluate to a value. For the iterator this implies that the process of elimination must be "well-behaved", for case it means that the process of selection must generate an object which evaluates to a value.

The elimination process is evoked by the evaluation of an iterator by the rule EvIt. Recall that elimination traverses the structure of the canonical form of the subject of iteration, by looking up each of the (pseudo) constructors in the term replacement Ω. Under the assumption that Ω is an element of the logical relation of the term replacement ($\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$) we can now state the missing link to the canonical form theorem for the iterator. $\hat{\Psi}$ represents the set of pseudo constructors possibly occurring in the canonical object, $\tilde{\Psi}$ contains their images under the term replacement Ω, and Ψ is the context in which the iterator object is well-typed.

If $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$

1. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \langle\omega; \Omega\rangle(V) \in [\![\langle\omega\rangle(B')]\!]$

From our experience with other proofs in this paper it can be easily seen that we must generalize this property before we can prove it. Canonical forms depend mutually on atomic forms, hence as first approximation we generalize the statement:

If $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$

1. If $\hat{\Psi} \vdash V \downarrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \langle\omega; \Omega\rangle(V) \in [\![\langle\omega\rangle(B')]\!]$
2. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \langle\omega; \Omega\rangle(V) \in [\![\langle\omega\rangle(B')]\!]$

But even this generalization does not go far enough. The attentive reader might already suspect that the generalized formulation of the property is not adequate to provide a strong enough induction hypothesis to actually prove the lemma. The problem lies in the rule CanLam which introduces new pseudo constructors into the set $\hat{\Psi}$. To solve this problem we introduce a substitution $\cdot; \mathrm{id}_\Psi \cup \varrho$ which satisfies the logical relation $[\![\cdot; \Psi \cup \tilde{\Psi}]\!]$. Please note, that is substitution acts as the identity substitution on all variables from Ψ. This is a crucial property in the argument because it allows us to use the *strengthening* lemma which states that if a substitution acts as the identity function on all free variables of a term M, then its application to M yields M. We call it strengthening lemma because the domain of the substitution might contain further variables which are not mapped to themselves.

Lemma 7.38 (Strengthening lemma)

Let $\hat{\Delta}; \Gamma \cup \Gamma^* \cup \hat{\Gamma} \vdash (\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}) \ : \ (\hat{\Delta}; \Gamma \cup \tilde{\Gamma} \cup \hat{\Gamma})$

1. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : A$ *then* $M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$
2. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma')$ *then* $\Xi = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi)$
3. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Omega : \langle \omega \rangle(\Sigma')$ *then* $\Omega = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega)$

Proof: See appendix C. □

The trick to use the identity function in the formulation of the lemma leaves us to prove the following lemma for the iterator:

If $\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_{\Psi} \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle \omega \rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$

1. If $\hat{\Psi} \vdash V \downarrow B'$ then $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_{\Psi} \cup \varrho](\langle \omega; \Omega \rangle(V)) \in [\![\langle \omega \rangle(B')]\!]$
2. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_{\Psi} \cup \varrho](\langle \omega; \Omega \rangle(V)) \in [\![\langle \omega \rangle(B')]\!]$

This formulation of the lemma is very close to the version we will finally prove. But we cannot prove it directly. The reason for this has already been mentioned in Section 6. Not every constructor $c : C$ which is encountered during the traversal is an element of Σ': constructors whose target type $\tau(C)$ is not in $\mathcal{I}(\Sigma; B)$ are replaced by themselves according to the definition of $\mathsf{ElConst}$. Being not an element in $\mathcal{I}(\Sigma; B)$ can have two possible reasons:

1. $\tau(C) \not\blacktriangleleft_{\Sigma;B} \tau(B)$
2. $\tau(B) \not\blacktriangleleft_{\Sigma;B} \tau(C)$

In the first case we are done, as a consequence of lemma 6.35 and lemma 6.36 from section 6. All what remains to show is that the second case cannot occur. We prove this property as by generalizing the auxiliary lemma for iterator further. The idea is to show that if during the elimination process a canonical form V of type B' is encountered then $\tau(B') \blacktriangleleft_B \tau(B)$ holds.

To accomplish this we define three conditions which are formulated as preconditions and postconditions for the elimination process — which we will add into the formulation of the auxiliary lemma for iterator. We distinguish between two preconditions: One for the case that the encountered object is canonical and for the case that the object is atomic. If an atomic form is an application, then ElApp must be applied. To show that the precondition for the second branch holds, we must define a postcondition, which is valid after the iteration of the first premiss terminates. The first premiss of AtApp is always atomic hence it is enough to establish the post condition only for the atomic case.

Definition 7.39 (Atomic precondition for elimination) *Let B an arbitrary pure type:*

$$\mathit{Pre}{\downarrow_B}\ (\Psi, B') :\Leftrightarrow \tau(B') \trianglelefteq_B \tau(B) \ \mathit{and}\ \Psi \blacktriangleleft_B \tau(B)$$

The first part of this definition guarantees the weak subordination of $\tau(B')$ and $\tau(B)$. We cannot assume strong subordination, because elimination can be applied to non-inductive types. The second part of this precondition is necessary because the precondition must imply the postcondition, which finally might be used to imply the precondition for canonical form elimination. The precondition for canonical forms is stronger then the one for atomic forms. It states in addition that for every pseudo constructor, all source types are subordinated by $\tau(B)$, as long as the pseudo constructor can be used in the definition of the object.

Definition 7.40 (Canonical precondition for elimination) *Let B arbitrary pure type:*

$$Pre{\Uparrow_B}\ (\Psi, B') :\Leftrightarrow$$

$$Pre{\downarrow_B}\ (\Psi, B') \textit{ and forall } B'' \in PCT(B') :$$

$$\textit{if } \tau(B'') \trianglelefteq_B \tau(B) \textit{ then forall } y \in Source(B'') : y \blacktriangleleft_B \tau(B)$$

The postcondition for atomic form elimination provides that every source type of a type of the atomic object to be eliminated is actually subordinated by the goal type of B — which might be very different from B'.

Definition 7.41 (Atomic postcondition for elimination) *Let B arbitrary pure type:*

$$Post{\downarrow_B}\ (B') :\Leftrightarrow \textit{forall } y \in Source(B') : y \blacktriangleleft_B \tau(B)$$

The proof of the auxiliary lemma for iterator will be by induction over the atomic or canonical structure of the elimination subject. The following lemma shows that preconditions and postconditions imply each other in a suitable way as necessary to perform the inductive argument. For this purpose recall the definition of atomic and canonical forms from Definition 2.6. If we have a derivation ending in an application of AtVar and the atomic precondition holds then we must show that the postcondition holds. The same holds for a derivation ending with AtConst. In the case of application (AtApp) we encounter the following situation. Let $\mathcal{D}$ be a derivation ending in

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Psi \vdash V_1 \downarrow B_1 \to B_2\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Psi \vdash V_2 \Uparrow B_1\end{array}}{\Psi \vdash V_1\ V_2 \downarrow B_2}\ \textsf{AtApp}$$

By assumption we know that $\mathrm{Pre}{\downarrow_B}\ (\Psi, B_2)$ holds. The application of the induction hypothesis to $\mathcal{D}_1$ requires that $\mathrm{Pre}{\downarrow_B}\ (\Psi, B_1 \to B_2)$ holds (to be proven). Finally for this case, the application of the induction hypothesis to $\mathcal{D}_2$ requires that $\mathrm{Pre}{\Uparrow_B}\ (\Psi, B_1)$ holds. For this proof we fortunately can assume $\mathrm{Post}{\downarrow_B}\ (B_1 \to B_2)$.

A complete list of all necessary implications is summarized by the following lemma. The first statement is needed for AtVar, the second for AtConst. The third and the forth are necessary for AtApp. CanAt doesn't require any special considerations, since the canonical precondition is stronger then the atomic one. The fifth statement makes the case CanLam go through. And finally the last fact provides the necessary information to ensure that the initial precondition holds (see Lemma 7.47).

Lemma 7.42 (Preservation of Preconditions and Postconditions)

1. $Pre{\downarrow_B}\ (\Psi, B')$ *and* $\Psi(x) = B'$ *then* $Post{\downarrow_B}\ (B')$
2. $Pre{\downarrow_B}\ (\Psi, B')$ *and* $\Sigma(c) = B'$ *then* $Post{\downarrow_B}\ (B')$
3. $Pre{\downarrow_B}\ (\Psi, B_2)$ *implies* $Pre{\downarrow_B}\ (\Psi, B_1 \to B_2)$
4. $Pre{\downarrow_B}\ (\Psi, B_2)$ *and* $Post{\downarrow_B}\ (B_1 \to B_2)$ *implies* $Pre{\Uparrow_B}\ (\Psi, B_1)$ *and* $Post{\downarrow_B}\ (B_2)$

5. *Pre*$\Uparrow_B$ $(\Psi, B_1 \to B_2)$ *implies Pre*$\Uparrow_B$ $(\Psi, x : B_1, B_2)$

6. *For all pure types B: Pre*$\Uparrow_B$ $(\cdot, B)$

Proof: See appendix C. □

Now all ingredients for the formulation of the auxiliary lemma for iterator are prepared. By inserting preconditions and postconditions for atomic and canonical forms into the formulation we obtain the auxiliary lemma for iterator which, as expected, is proven by mutual induction over atomic and canonical forms. The proof relies on the results from lemma 7.42.

Lemma 7.43 (Auxiliary lemma for iterator)
If $\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ *and* $\Sigma' = \mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$

1. *If* $\hat{\Psi} \vdash V \downarrow B'$ *and Pre*$\downarrow_B$ $(\hat{\Psi}, B')$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(V)) \in [\![\langle\omega\rangle(B')]\!]$ *and Post*$\downarrow_B$ (B')

2. *If* $\hat{\Psi} \vdash V \Uparrow B'$ *and Pre*$\Uparrow_B$ $(\hat{\Psi}, B')$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(V)) \in [\![\langle\omega\rangle(B')]\!]$

Proof: See appendix C. □

Similarly to the development of the auxiliary lemma for iterator we can prove an auxiliary lemma for case, which shows that the selection process "behaves" well. The selection process seems to be much easier then the elimination process, because only the head constructor selects an object, its arguments must be closed and boxed. Thus, before discussing the auxiliary lemma for case we present a result which shows how to close and box arguments.

For this purpose we must show that a canonical object of type B is an element of the logical relation $[\![B]\!]$. For the same reasons as above, we must prove this result also for atomic objects. The possible introduction of pseudo constructors by CanLam makes it necessary that we introduce a substitution $\cdot; \varrho$, replacing all pseudo constructors in Ψ by some object. The substitution is assumed to satisfy $[\cdot; \Psi]$. We need this result for the proof of Lemma 7.45

Lemma 7.44 (Every canonical element is member of the logical relation)

1. *If* $\Psi \vdash V \downarrow B$ *and* $\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$ *then* $\Psi' \vdash [\cdot; \varrho](V) \in [\![B]\!]$

2. *If* $\Psi \vdash V \Uparrow B$ *and* $\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$ *then* $\Psi' \vdash [\cdot; \varrho](V) \in [\![B]\!]$

Proof: See appendix C. □

We now present the final preparatory lemma for the auxiliary lemma for case. Consider rule SeApp from definition 5.15. The selection judgment applied to an application of the form $V_1\ V_2$ selects some object for M_1 first and constructs a boxed abstraction closure over object V_2. The result of the selection process is then the application of M_1 to this newly constructed object. Note that V_2 can contain any pseudo constructor introduced by preceding λ-abstractions. We show now that we can close V_2 under all these variables by using abstraction closures.

Lemma 7.45 (Properties of transformation types)

If $\Psi \vdash V \Uparrow B$ *then* $\cdot \vdash \lambda\{\Psi\}.\, V \in [\![\Pi\{\Psi\}.\, B]\!]$

Proof: See appendix C. □

We can now use the previous four lemmas to prove the auxiliary lemma for case. Recall that the overall goal is a result which is very similar to the iterator. From the assumption that the match Ξ satisfies $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ we need to prove that the selection process generates an object which satisfies the logical relation $[\![\mathcal{C}^*\ (B, A, B)]\!]$. Recall from the formulation of the auxiliary lemma of the iterator that $\hat{\Psi}$ represents the set of pseudo constructors possibly occurring in the canonical object, $\tilde{\Psi}$ contains their images under the match Ξ, and Ψ is the context in which the case object is well-typed.

If $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}(\Sigma; \tau(B))$ then

1. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \{B \Rightarrow A; \Xi; \hat{\Psi}\}(V) \in [\![\mathcal{C}^*\ (B, A, B')]\!]$

Unfortunately, this formulation of the lemma cannot be proven without further generalization. The first generalization step has to do with the mutual dependency of atomic and canonical forms. We have to be more careful then in the iterator case because atomic objects are only of "case type" but not of "complete case type".

If $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}(\Sigma; \tau(B))$ then

1. If $\hat{\Psi} \vdash V \downarrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \{B \Rightarrow A; \Xi; \hat{\Psi}\}(V) \in [\![\mathcal{C}\ (B, A, B')]\!]$
2. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \tilde{\Psi} \vdash \{B \Rightarrow A; \Xi; \hat{\Psi}\}(V) \in [\![\mathcal{C}^*\ (B, A, B')]\!]$

Following the example of the iterator, we encounter exactly the same problem here. The rule CanLam extends the set of pseudo constructors which would prevent the induction hypothesis to apply in this case. To generalize this lemma more, we use the same trick with the substitution $\cdot; \mathrm{id}_\Psi \cup \varrho$ as in the iterator case.

If $\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ and $\Sigma' = \mathcal{S}(\Sigma; \tau(B))$

1. If $\hat{\Psi} \vdash V \downarrow B'$ then $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}\ (B, A, B')]\!]$
2. If $\hat{\Psi} \vdash V \Uparrow B'$ then $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}^*\ (B, A, B')]\!]$

This formulation is very close to the lemma as we prove it. But still the proof doesn't go through. The problem is the context $\hat{\Psi}'$ of pseudo constructors. We must be able to recover the information that $\Pi\{\hat{\Psi}\}.\, \tau(B) = B$. To make this more formal, we can assume this property to hold for the atomic case: $\hat{\Psi}$ essentially represents B. For the canonical case, we can assume that $\hat{\Psi}$ represents the initial set of domain types of type B. The remaining domain types are still represented by the type B': $\Pi\{\hat{\Psi}\}.\, B' = B$. Another piece of information is needed for the successful proof: $\tau(B) = \tau(B')$.

Lemma 7.46 (Auxiliary lemma for case)

If $\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ *and* $\Sigma' = \mathcal{S}(\Sigma; \tau(B))$

1. *If* $\hat{\Psi} \vdash V \downarrow B'$ *and* $\Pi\{\hat{\Psi}\}.\,\tau(B) = B$ *and* $\tau(B') = \tau(B)$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}\ (B, A, B')]\!]$
2. *If* $\hat{\Psi} \vdash V \Uparrow B'$ *and* $\Pi\{\hat{\Psi}\}.\,B' = B$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}^*\ (B, A, B')]\!]$

Proof: See appendix C. □

This concludes the presentation of the preparatory work. All the ingredients are prepared, we just have to put them together to obtain the generalized first lemma completing the canonical form theorem. Recall that the proof of the canonical form theorem is performed in two steps — as we discussed earlier:

1. If $\Psi \vdash M \in [\![B]\!]$ then $\Psi \vdash M \Uparrow V : B$
2. If $\cdot; \Psi \vdash M : A$ then $\Psi \vdash M \in [\![A]\!]$

The first property was already proven in lemma 7.17. We discussed that a generalization of the second step is necessary. This generalization led to the introduction of logical relations for contexts with substitutions as their elements. We now prove this generalized version, the centerpiece of this work. All results obtained so far are needed for the proof of this lemma. The proof is done by induction over the typing derivation.

Lemma 7.47 (Typing and logical relations)
Let $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$

1. *If* $\Delta; \Gamma \vdash M : A$ *then* $\Psi \vdash [\theta; \varrho](M) \in [\![A]\!]$
2. *If* $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$ *then* $\Psi + \cdot \vdash [\theta; \varrho](\Xi) \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$
3. *If* $\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$ *then* $\Psi + \cdot \vdash [\theta; \varrho](\Omega) \in [\![\langle\omega\rangle(\Sigma'; \cdot)]\!]$

Proof: See appendix C. □

As the main result of this section both lemmas, namely lemma 7.17 and lemma 7.47 can be summarized to the canonical form theorem. This theorem says, that every object M of pure type evaluates to a canonical form. In other words, no matter how complex the form of the object M is, it may contain λ-abstractions, applications, boxes, and lets, it will evaluate to a canonical form, only containing λ-abstractions and applications. Section 9 emphasizes this point again and shows the usefulness of this result.

Theorem 7.48 (Canonical form theorem)

If $\cdot; \Psi \vdash M : B$ *then* $\Psi \vdash M \Uparrow V : B$

Proof: See appendix C.

□

8 Type preservation theorem

The canonical form lemma and the corresponding theorem are two very powerful theorems. The type preservation property of the operational semantics of our system follows as a corollary if we have one more lemma, namely, that the value of an object is unique. This claim is intuitively immediate because the form of the object triggers the evaluation rule — which is uniquely defined. The operational semantics and evaluation to canonical forms are mutual dependent, hence the uniqueness lemma reads as follows.

Lemma 8.1 (Uniqueness of evaluation)

1. *If* $\Psi \vdash M \Uparrow V : A$ *and* $\Psi \vdash M \Uparrow V' : A$ *then* $V = V'$
2. *If* $\Psi \vdash M \hookrightarrow V : A$ *and* $\Psi \vdash M \hookrightarrow V' : A$ *then* $V = V'$

Proof: See appendix D. □

An easy corollary from Lemma 7.47 is now the type preservation theorem guarenteeing that types are preserved under our operational semantics.

Theorem 8.2 (Type preservation)

If $\cdot; \Psi \vdash M : A$ *and* $\Psi \vdash M \hookrightarrow V : A$ *then* $\cdot; \Psi \vdash V : A$

Proof: See appendix D. □

In the next section we present another corollary from Lemma 7.47: Our calculus for the modal λ-calculus is a conservative extension of the simply typed λ-calculus.

9 Conservative extension theorem

We arrived at the final result of this paper: Our calculus is a conservative extension of the simply-typed λ-calculus. By definition it is clear the the language of objects and types of the modal λ-calculus extends the formulation of the simply typed λ-calculus. It follows quite naturally that every typing derivation in the simply-typed calculus can be represented in our system: Using the empty modal context, StpVar must be replaced by TpVarReg, StpConst by TpConst, StpLam by TpLam, and finally StpApp by TpApp.

Lemma 9.1 (Typing extension)

If $\Psi \vdash M : B$ *then* $\cdot; \Psi \vdash M : B$

Proof: See appendix E. □

Let M be an object of pure type B with free variables from a pure context Ψ. M itself need not be pure but rather some term in the modal λ-calculus including boxes, lets, iterators, and definition by cases. We have seen that M has a canonical form V, and Lemma 7.2 (1) shows that V must be a term in the simply typed λ-calculus.

Theorem 9.2 (Conservative Extension)

If $\cdot; \Psi \vdash M : B$ *then* $\Psi \vdash M \Uparrow V : B$ *and* $\Psi \vdash V \Uparrow B$

Proof: See appendix E. □

This concludes the discussion of the meta-theoretic properties of the modal λ-calculus which we presented in this paper.

10 Conclusion and Future Work

We have presented a calculus for primitive recursive functionals over higher-order abstract syntax which guarantees that the adequacy of encodings remains intact. The requisite conservative extension theorem is technically deep and requires a careful system design and analysis of the properties of a modal operator $\Box$ and its interaction with function definition by iteration and cases. To our knowledge, this is the first system in which it is possible to safely program functionally with higher-order abstract syntax representations. It thus complements and refines the logic programming approach to programming with such representations [Mil92, Pfe91].

Our work was inspired by Miller's system [Mil90], which was presented in the context of ML. Due to the presence of unrestricted recursion and the absence of a modal operator, Miller's system is computationally adequate, but has a much weaker meta-theory which would not be sufficient for direct use in a logical framework. The system of Meijer and Hutton [MH95] and its refinement by Fegaras and Sheard [FS96] are also related in that they extend primitive recursion to encompass functional objects. However, they treat functional objects extensionally, while our primitives are designed so we can analyze the internal structure of λ-abstractions directly. Fegaras and Sheard also note the problem with adequacy and design more stringent type-checking rules in Section 3.4 of [FS96] to circumvent this problem. In contrast to our system, their proposal does not appear to have a logical interpretation. Furthermore, they neither claim nor prove type preservation or an appropriate analogue of conservative extension—critical properties which are not obvious in the presence of their internal type tags and **Place** constructor.

Our system is satisfactory from the theoretical point of view and could be the basis for a practical implementation. Such an implementation would allow the definition of functions of arbitrary types, while data constructors are constrained to have pure type. Many natural functions over higher-order representations turn out to be directly definable (e.g., one-step parallel reduction or conversion to de Bruijn indices), others require explicit counters to guarantee termination (e.g., multi-step reduction or full evaluation). On the other hand, it appears that some natural *algorithms* (e.g., a structural equality check which traverses two expressions simultaneously) are not implementable, even though the underlying function is certainly definable (e.g., via a translation to de Bruijn indices). For larger applications, writing programs by iteration becomes tedious and error-prone and a pattern-matching calculus such as employed in ALF [CNSvS94] or proposed by Jouannaud and Okada [JO91] seems more practical. Our informal notation in the examples provides some hints what concrete syntax one might envision for an implementation along these lines.

The present paper is a first step towards a system with dependent types in which proofs of meta-logical properties of higher-order encodings can be expressed directly by dependently typed, total functions. The meta-theory of such a system appears to be highly complex, since the modal operators necessitate a *let box* construct which, *prima facie*, requires commutative conversions. Martin Hofmann[1] has proposed a semantical explanation for our iteration operator which has led him to discover an equational formulation of the laws for iteration. This may be the critical insight required for a dependently typed version of our calculus. We also plan to reexamine applications in the realm of functional programming [Mil90, FS96] and related work on reasoning about higher-order abstract syntax with explicit induction [DH94, DFH95] or definitional reflection [MM96].

Acknowledgments. The work reported here took a long time to come to fruition, largely due to the complex nature of the technical development. During this time we have discussed various

[1] personal communication

aspects of higher-order abstract syntax, iteration, and induction with too many people to acknowledge them individually. Special thanks go to Gérard Huet and Chet Murthy, who provided the original inspiration, and Hao-Chi Wong who helped us understand the nature of modality in this context.

A Definition modal λ-calculus

$$
\begin{array}{llcl}
\text{Types:} & A & ::= & a \mid A_1 \to A_2 \mid \Box A \mid A_1 \times A_2 \\
\text{Pure types:} & B & ::= & a \mid B_1 \to B_2 \\
\text{Objects:} & M & ::= & c \mid x \mid \lambda x\!:\!A.\, M \mid M_1\ M_2 \\
 & & & \mid \text{box } M \mid \text{let box } x = M_1 \text{ in } M_2 \mid \langle M_1, M_2 \rangle \mid \text{fst } M \mid \text{snd } M \\
 & & & \mid \text{it } \langle \omega \rangle\ M\ \langle \Omega \rangle \mid \text{case } \langle A \rangle\ M\ \langle \Xi \rangle \\
\text{Term replacement:} & \Omega & ::= & \cdot \mid (\Omega \mid c \mapsto M) \mid (\Omega \mid x \mapsto x') \\
\text{Match:} & \Xi & ::= & \cdot \mid (\Xi \mid c \Rightarrow M) \mid (\Xi \mid x \Rightarrow x') \\
\text{Contexts:} & \Gamma & ::= & \cdot \mid \Gamma, x : A \\
\text{Pure Context:} & \Psi & ::= & \cdot \mid \Psi, x : B \\
\text{Signature:} & \Sigma & ::= & \cdot \mid \Sigma, a : \text{type} \mid \Sigma, c : B
\end{array}
$$

Definition 2.6 (Atomic and canonical forms)

1. $\Psi \vdash V \downarrow B$ *(V is atomic of type B in* Ψ*)*
2. $\Psi \vdash V \Uparrow B$ *(V is canonical of type B in* Ψ*)*

are defined by:

$$
\frac{\Psi(x) = B}{\Psi \vdash x \downarrow B}\,\mathsf{AtVar}
\qquad
\frac{\Sigma(c) = B}{\Psi \vdash c \downarrow B}\,\mathsf{AtConst}
\qquad
\frac{\Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash V_2 \Uparrow B_2}{\Psi \vdash V_1\ V_2 \downarrow B_1}\,\mathsf{AtApp}
$$

$$
\frac{\Psi \vdash V \downarrow a}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt}
\qquad
\frac{\Psi, x : B_1 \vdash V \Uparrow B_2}{\Psi \vdash \lambda x\!:\!B_1.\, V \Uparrow B_1 \to B_2}\,\mathsf{CanLam}
$$

Definition 3.1 (Typing judgment)

1. $\Delta; \Gamma \vdash M : A$ *(M has type A in context* $\Delta; \Gamma$*)*
2. $\Delta; \Gamma \vdash \Omega : \langle \omega \rangle (\Sigma)$ *(*Ω *is a well-typed term replacement with respect to context* $\Delta; \Gamma$*, type replacement* ω*, and signature* Σ*)*
3. $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle (\Sigma)$ *(*Ξ *is a well-typed match with respect to context* $\Delta; \Gamma$*, subject type B, goal type A, and signature* Σ*)*

$$
\frac{\Gamma(x) = A}{\Delta; \Gamma \vdash x : A}\,\mathsf{TpVarReg}
\qquad
\frac{\Delta(x) = A}{\Delta; \Gamma \vdash x : A}\,\mathsf{TpVarMod}
\qquad
\frac{\Sigma(c) = B}{\Delta; \Gamma \vdash c : B}\,\mathsf{TpConst}
$$

$$
\frac{\Delta; \Gamma, x : A_1 \vdash M : A_2}{\Delta; \Gamma \vdash \lambda x\!:\!A_1.\, M : A_1 \to A_2}\,\mathsf{TpLam}
\qquad
\frac{\Delta; \Gamma \vdash M_1 : A_2 \to A_1 \qquad \Delta; \Gamma \vdash M_2 : A_2}{\Delta; \Gamma \vdash M_1\ M_2 : A_1}\,\mathsf{TpApp}
$$

$$\frac{\Delta; \Gamma \vdash M_1 : A_1 \qquad \Delta; \Gamma \vdash M_2 : A_2}{\Delta; \Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2}\ \mathsf{TpPair}$$

$$\frac{\Delta; \Gamma \vdash M : A_1 \times A_2}{\Delta; \Gamma \vdash \mathit{fst}\ M : A_1}\ \mathsf{TpFst} \qquad \frac{\Delta; \Gamma \vdash M : A_1 \times A_2}{\Delta; \Gamma \vdash \mathit{snd}\ M : A_2}\ \mathsf{TpSnd}$$

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathit{box}\ M : \Box A}\ \mathsf{TpBox} \qquad \frac{\Delta; \Gamma \vdash M_1 : \Box A_1 \qquad \Delta, x : A_1; \Gamma \vdash M_2 : A_2}{\Delta; \Gamma \vdash \mathit{let}\ \mathit{box}\ x = M_1\ \mathit{in}\ M_2 : A_2}\ \mathsf{TpLet}$$

$$\frac{\Delta; \Gamma \vdash M : \Box B \qquad \vdash \omega : \alpha \qquad \Delta; \Gamma \vdash \Omega : \langle \omega \rangle (\Sigma')}{\Delta; \Gamma \vdash \mathit{it}\ \langle \omega \rangle\ M\ \langle \Omega \rangle : \langle \omega \rangle (B)}\ \mathsf{TpIt}$$

$$\mathit{where}\ \alpha = \mathcal{I}(\Sigma; B)\ \mathit{and}\ \Sigma' = \mathcal{S}^*(\Sigma; \alpha)$$

$$\frac{\Delta; \Gamma \vdash M : \Box B \qquad \Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle (\Sigma')}{\Delta; \Gamma \vdash \mathit{case}\ \langle A \rangle\ M\ \langle \Xi \rangle : \mathcal{C}^*\ (B, A, B)}\ \mathsf{TpCase}$$

$$\mathit{where}\ \Sigma' = \mathcal{S}(\Sigma; \tau(B))$$

$$\frac{}{\Delta; \Gamma \vdash \cdot : \langle \omega \rangle (\cdot)}\ \mathsf{TrBase} \qquad \frac{\Delta; \Gamma \vdash \Omega : \langle \omega \rangle (\Sigma) \qquad \Delta; \Gamma \vdash M : \langle \omega \rangle (B')}{\Delta; \Gamma \vdash (\Omega \mid c \mapsto M) : \langle \omega \rangle (\Sigma, c : B')}\ \mathsf{TrInd}$$

$$\frac{}{\Delta; \Gamma \vdash \cdot : \langle B \Rightarrow A \rangle (\cdot)}\ \mathsf{TmBase} \qquad \frac{\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle (\Sigma) \qquad \Delta; \Gamma \vdash M : \mathcal{C}\ (B, A, B')}{\Delta; \Gamma \vdash (\Xi \mid c \Rightarrow M) : \langle B \Rightarrow A \rangle (\Sigma, c : B')}\ \mathsf{TmInd}$$

Definition 4.28 (Elimination)

$$\begin{array}{rcll}
\langle \omega; \Omega \rangle (c) & = & \begin{cases} M & \mathit{if}\ \Omega(c) = M \\ c & \mathit{otherwise} \end{cases} & (\mathsf{ElConst}) \\
\langle \omega; \Omega \rangle (x) & = & \Omega(x) & (\mathsf{ElVar}) \\
\langle \omega; \Omega \rangle (\lambda x{:}B.\, V) & = & \lambda u{:}\langle \omega \rangle (B).\, \langle \omega; \Omega \mid x \mapsto u \rangle (V) & (\mathsf{ElLam}) \\
\langle \omega; \Omega \rangle (V_1\, V_2) & = & \langle \omega; \Omega \rangle (V_1)\ \langle \omega; \Omega \rangle (V_2) & (\mathsf{ElApp})
\end{array}$$

Definition 5.15 (Selection)

$$\begin{array}{rcll}
\{B \Rightarrow A; \Xi; \Psi\}(c) & = & \Xi(c) & (\mathsf{SeConst}) \\
\{B \Rightarrow A; \Xi; \Psi\}(x) & = & \Xi(x) & (\mathsf{SeVar}) \\
\{B \Rightarrow A; \Xi; \Psi\}(\lambda x{:}B'.\, V) & = & \lambda u{:}\mathcal{C}\ (B, A, B').\, \{B \Rightarrow A; \Xi, x \Rightarrow u; (\Psi, x : B')\}(V) & (\mathsf{SeLam}) \\
\{B \Rightarrow A; \Xi; \Psi\}(V_1\, V_2) & = & \{B \Rightarrow A; \Xi; \Psi\}(V_1)\ (\mathit{box}\ \lambda\{\Psi\}.\, V_2) & (\mathsf{SeApp})
\end{array}$$

Definition 3.3 (Evaluation judgments)

1. $\Psi \vdash M \hookrightarrow V : A$ *(M evalulates to V of type A in context Ψ)*
2. $\Psi \vdash M \Uparrow V : B$ *(M evalulates to a canonical form V of pure type B in context Ψ)*

$$\frac{\Psi \vdash M \hookrightarrow V : a}{\Psi \vdash M \Uparrow V : a}\,\textsf{EcAtomic} \qquad \frac{\Psi, x : B_1 \vdash M\ x \Uparrow V : B_2}{\Psi \vdash M \Uparrow \lambda x{:}B_1.\, V : B_1 \to B_2}\,\textsf{EcArrow}$$

$$\frac{\Psi(x) = A}{\Psi \vdash x \hookrightarrow x : A}\,\textsf{EvVar} \qquad \frac{\Sigma(c) = B}{\Psi \vdash c \hookrightarrow c : B}\,\textsf{EvConst}$$

$$\frac{\cdot; \Psi, x : A_1 \vdash M : A_2}{\Psi \vdash \lambda x{:}A_1.\, M \hookrightarrow \lambda x{:}A_1.\, M : A_1 \to A_2}\,\textsf{EvLam}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow \lambda x{:}A_2.\, M_1' : A_2 \to A_1 \qquad \Psi \vdash M_2 \hookrightarrow V_2 : A_2 \qquad \Psi \vdash [V_2/x](M_1') \hookrightarrow V : A_1}{\Psi \vdash M_1\ M_2 \hookrightarrow V : A_1}\,\textsf{EvApp}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow V_1 : B_2 \to B_1 \qquad \Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash M_2 \Uparrow V_2 : B_2}{\Psi \vdash M_1\ M_2 \hookrightarrow V_1\ V_2 : B_1}\,\textsf{EvAtomic}$$

$$\frac{\cdot; \Psi \vdash M_1 : A_1 \qquad \cdot; \Psi \vdash M_2 : A_2}{\Psi \vdash \langle M_1, M_2 \rangle \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2}\,\textsf{EvPair}$$

$$\frac{\Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2 \qquad \Psi \vdash M_1 \hookrightarrow V : A_1}{\Psi \vdash \mathit{fst}\ M \hookrightarrow V : A_1}\,\textsf{EvFst}$$

$$\frac{\Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2 \qquad \Psi \vdash M_2 \hookrightarrow V : A_2}{\Psi \vdash \mathit{snd}\ M \hookrightarrow V : A_2}\,\textsf{EvSnd}$$

$$\frac{\cdot; \cdot \vdash M : A}{\Psi \vdash \mathit{box}\ M \hookrightarrow \mathit{box}\ M : \Box A}\,\textsf{EvBox}$$

$$\frac{\Psi \vdash M_1 \hookrightarrow \mathit{box}\ M_1' : \Box A \qquad \Psi \vdash [M_1'/x](M_2) \hookrightarrow V : A_2}{\Psi \vdash \mathit{let\ box}\ x = M_1\ \mathit{in}\ M_2 \hookrightarrow V : A_2}\,\textsf{EvLet}$$

$$\frac{\Psi \vdash M \hookrightarrow \mathit{box}\ M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \langle \omega; \Omega \rangle (V') \hookrightarrow V : \langle \omega \rangle (B)}{\Psi \vdash \mathit{it}\ \langle \omega \rangle\ M\ \langle \Omega \rangle \hookrightarrow V : \langle \omega \rangle (B)}\,\textsf{EvIt}$$

$$\frac{\Psi \vdash M \hookrightarrow \mathit{box}\ M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}{\Psi \vdash \mathit{case}\ \langle A \rangle\ M\ \langle \Xi \rangle \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}\,\textsf{EvCase}$$

B Preliminary results

Lemma 6.2 (Every context extends the empty context)

$\Gamma \geq \cdot$

Proof: by induction over Γ

Case: $\Gamma = \cdot$

$\cdot \geq \cdot$ by application of CeBase

Case: $\Gamma = \Gamma', x : A$

$\Gamma' \geq \cdot$ by ind. hyp.

$\Rightarrow \quad \Gamma', x : A \geq \cdot$ by application of CeInd

□

Lemma 6.4 (Context form) *If* $\bar{\Gamma}'' \geq \Gamma \cup \bar{\Gamma}$ *then* $\bar{\Gamma}'' = \Gamma \cup \bar{\Gamma}'$ *and* $\bar{\Gamma}' \geq \bar{\Gamma}$

Proof: by induction over $\mathcal{D} :: \bar{\Gamma}'' \geq \Gamma \cup \bar{\Gamma}$:

Case: $\mathcal{D} = \dfrac{}{\Gamma \cup \bar{\Gamma} \geq \Gamma \cup \bar{\Gamma}}\,\mathsf{CeBase}$:

$\bar{\Gamma}'' = \Gamma \cup \bar{\Gamma}$ by definition

$\bar{\Gamma} \geq \bar{\Gamma}$ by application of CeInd

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \bar{\Gamma}'' \geq \Gamma \cup \bar{\Gamma}\end{array}}{\bar{\Gamma}'', x : A \geq \Gamma \cup \bar{\Gamma}}\,\mathsf{CeInd}$:

$\bar{\Gamma}'' = \Gamma \cup \bar{\Gamma}'$ by ind. hyp. on $\mathcal{D}_1$

$\bar{\Gamma}' \geq \bar{\Gamma}$ by ind. hyp. on $\mathcal{D}_1$

$\Rightarrow \quad \bar{\Gamma}'', x : A = (\Gamma \cup \bar{\Gamma}'), x : A$ by definition

$\Rightarrow \quad \bar{\Gamma}'', x : A = \Gamma \cup \bar{\Gamma}', x : A$ by application of CuInd

$\bar{\Gamma}', x : A \geq \bar{\Gamma}$ by application of CeInd

□

Lemma 6.19 (Modal substitution restriction)

If $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ *then* $\Delta'; \cdot \vdash (\theta; \cdot) \ : \ (\Delta; \cdot)$

Proof: by induction over $\mathcal{D} :: \Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$

Case: $\mathcal{D} = \dfrac{}{\Delta'; \Gamma' \vdash (\cdot; \cdot) \ : \ (\cdot; \cdot)}\,\mathsf{TSBase}$

$\Delta; \Gamma = \cdot; \cdot$ by assumption

$\Rightarrow \quad \Delta'; \cdot \vdash (\cdot; \cdot) \ : \ (\cdot; \cdot)$ by definition TSBase

Case: $\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Delta'; \cdot \vdash M : A & \Delta'; \Gamma' \vdash (\theta; \varrho) \;:\; (\Delta; \Gamma) \end{array}}{\Delta'; \Gamma' \vdash (\theta, M/x; \varrho) \;:\; (\Delta, x : A; \Gamma)}\,\mathsf{TSMod}$

$\Delta'; \cdot \vdash (\theta; \cdot) \;:\; (\Delta; \cdot)$ by ind. hyp. on $\mathcal{D}_2$

$\Rightarrow \quad \Delta'; \cdot \vdash (\theta, M/x; \cdot) \;:\; (\Delta, x : A; \cdot)$ by application of TSMod with $\mathcal{D}_1$

Case: $\mathcal{D} = \dfrac{\begin{array}{cc} & \mathcal{D}_1 \\ \Delta'; \Gamma' \vdash M : A & \Delta'; \Gamma' \vdash (\theta; \varrho) \;:\; (\Delta; \Gamma) \end{array}}{\Delta'; \Gamma' \vdash (\theta; \varrho, M/x) \;:\; (\Delta; \Gamma, x : A)}\,\mathsf{TSReg}$

$\Delta'; \cdot \vdash (\theta; \cdot) \;:\; (\Delta; \cdot)$ by ind. hyp. on $\mathcal{D}_1$

□

Lemma 6.20 (Properties of typing relation for substitutions)

1. *If* $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\Delta'; \Gamma' \vdash (\theta; \varrho) \;:\; (\Delta; \Gamma)$ *then* $\theta(x) = M$ *and* $\Delta'; \cdot \vdash M : A$
2. *If* $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Delta'; \Gamma' \vdash (\theta; \varrho) \;:\; (\Delta; \Gamma)$ *then* $\varrho(x) = M$ *and* $\Delta'; \Gamma' \vdash M : A$

Proof: by induction over $\mathcal{D} :: \Delta'; \Gamma' \vdash (\theta; \varrho) \;:\; (\Delta; \Gamma)$

Case: $\mathcal{D} = \dfrac{}{\Delta'; \Gamma' \vdash (\cdot; \cdot) \;:\; (\cdot; \cdot)}\,\mathsf{TSBase}$

cannot occur.

Case: $\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Delta'; \cdot \vdash M' : A' & \Delta'; \Gamma' \vdash (\theta'; \varrho) \;:\; (\Delta''; \Gamma) \end{array}}{\Delta'; \Gamma' \vdash (\theta', M'/y; \varrho) \;:\; (\Delta'', y : A'; \Gamma)}\,\mathsf{TSMod}$

$\Delta = \Delta'', y : A'$ by assumption

$\theta = \theta', M'/y$ by assumption

1. **Case:** $\Delta_2 = \cdot$
   - $\Delta = (\Delta_1, x : A) \cup \cdot$ by assumption
   - $\Rightarrow \quad \Delta = \Delta_1, x : A$ by definition CuBase
   - $\Rightarrow \quad \Delta'' = \Delta_1$ by definition
   - $\Rightarrow \quad x = y$ by definition
   - $\Rightarrow \quad A' = A$ by definition
   - $\Rightarrow \quad (\theta', M'/x)(x) = M'$ by definition SbaInd
   - $\Rightarrow \quad \theta(x) = M'$ by definition
   - $\Rightarrow \quad \Delta'; \cdot \vdash M' : A$ by assumption

   Case: $\Delta_2 = \Delta_2', z : A''$, $x \neq z$
   - $\Delta = (\Delta_1, x : A) \cup (\Delta_2', z : A'')$ by assumption
   - $\Rightarrow \quad \Delta = (\Delta_1, x : A) \cup \Delta_2', z : A''$ by definition CuInd
   - $\Rightarrow \quad \Delta'' = (\Delta_1, x : A) \cup \Delta_2'$ by definition
   - $\Rightarrow \quad y = z$ by definition

$\Rightarrow A'' = A'$ by definition
$\Rightarrow \theta'(x) = M$ by ind. hyp. (1) on $\mathcal{D}_2$
$\Rightarrow \Delta'; \cdot \vdash M : A$ by ind. hyp. (1) on $\mathcal{D}_2$
$\Rightarrow (\theta', M'/y)(x) = M$ by definition SbaInd
$\Rightarrow \theta(x) = M$ by definition

2. $\varrho(x) = M$ by ind. hyp. (2) on $\mathcal{D}_2$
$\cdot; \Psi \vdash M : A$ by ind. hyp. (2) on $\mathcal{D}_2$

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Delta'; \Gamma' \vdash M' : A'} \qquad \overset{\mathcal{D}_2}{\Delta'; \Gamma' \vdash (\theta; \varrho') \ : \ (\Delta; \Gamma')}}{\Delta'; \Gamma' \vdash (\theta; \varrho', M'/y) \ : \ (\Delta; \Gamma'', y : A')}\ \mathsf{TSReg}$

$\Gamma = \Gamma'', y : A'$ by assumption
$\varrho = \varrho', M'/y$ by assumption

1. $\theta(x) = M$ by ind. hyp. (1) on $\mathcal{D}_2$
$\Delta'; \cdot \vdash M : A$ by ind. hyp. (1) on $\mathcal{D}_2$

2. **Case:** $\Gamma_2 = \cdot$
$\Gamma = (\Gamma_1, x : A) \cup \cdot$ by assumption
$\Rightarrow \Gamma = \Gamma_1, x : A$ by definition CuBase
$\Rightarrow \Gamma'' = \Gamma_1$ by definition
$\Rightarrow x = y$ by definition
$\Rightarrow A' = A$ by definition
$\Rightarrow (\varrho', M'/x)(x) = M'$ by definition SbaInd
$\Rightarrow \varrho(x) = M'$ by definition
$\Rightarrow \Delta'; \Gamma' \vdash M' : A$ by definition $\mathcal{D}_1$

Case: $\Gamma_2 = \Gamma_2', z : A''$, $x \neq z$
$\Gamma = (\Gamma_1, x : A) \cup (\Gamma_2', z : A'')$ by assumption
$\Gamma = (\Gamma_1, x : A) \cup \Gamma_2', z : A''$ by definition CuInd
$\Rightarrow \Gamma'' = (\Gamma_1, x : A) \cup \Gamma_2'$ by definition
$\Rightarrow y = z$ by definition
$\Rightarrow A'' = A'$ by definition
$\Rightarrow \varrho'(x) = M$ by ind. hyp. (2) on $\mathcal{D}_2$
$\Rightarrow \Delta'; \Gamma' \vdash M : A$ by ind. hyp. (2) on $\mathcal{D}_2$
$\Rightarrow (\varrho', M'/y)(x) = M$ by definition SbaInd
$\Rightarrow \varrho(x) = M$ by definition

□

Lemma 6.21 (Substitution lemma for typing relation)
Let $\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$, then the following holds:

1. *If $\Delta; \Gamma \vdash M : A$ then $\Delta'; \Gamma' \vdash [\theta; \varrho](M) : A$*

2. *If $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma')$ then $\Delta'; \Gamma' \vdash [\theta; \varrho](\Xi) : \langle B \Rightarrow A \rangle(\Sigma')$*

3. *If* $\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$ *then* $\Delta'; \Gamma' \vdash [\theta; \varrho](\Omega) : \langle\omega\rangle(\Sigma')$

Proof: by induction over $\mathcal{D} :: \Delta; \Gamma \vdash M : A$, $\mathcal{E} :: \Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$ and $\mathcal{F} :: \Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$:

1. **Case:** $\mathcal{D} = \dfrac{\Gamma(x) = A}{\Delta; \Gamma \vdash x : A}$ $\mathsf{TpVarReg}$

$\Gamma(x) = A$	by assumption
$\Rightarrow \quad \Gamma = \Gamma_1, x : A \cup \Gamma_2$	by definition 2.2
$\Rightarrow \quad \varrho(x) = M$	by lemma 6.20 (1)
$\Rightarrow \quad \Delta'; \Gamma' \vdash M : A$	by lemma 6.20 (1)
$\Rightarrow \quad [\theta; \varrho](x) = M$	by definition SBVar
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](x) : A$	by definition

Case: $\mathcal{D} = \dfrac{\Delta(x) = A}{\Delta; \Gamma \vdash x : A}$ $\mathsf{TpVarMod}$

$\Delta(x) = A$	by assumption
$\Delta = \Delta_1, x : A \cup \Delta_2$	by definition
$\Rightarrow \quad \theta(x) = M$	by lemma 6.20 (2)
$\Rightarrow \quad \Delta'; \cdot \vdash M : A$	by lemma 6.20 (2)
$\Rightarrow \quad [\theta; \varrho](x) = M$	by definition SBVar
$\Rightarrow \quad \Delta'; \cdot \vdash [\theta; \varrho](x) : A$	by definition
$\Rightarrow \quad \Gamma' \geq \cdot$	by lemma 6.2
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](x) : A$	by lemma 6.8 (2)

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\Delta; \Gamma \vdash c : B}$ $\mathsf{TpConst}$

$\Sigma(c) = B$	by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash c : B$	by application of $\mathsf{TpConst}$
$\Rightarrow \quad [\theta; \varrho](c) = c$	by application of $\mathsf{SBConst}$
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](c) : B$	by definition

Case: $\mathcal{D} = \dfrac{\Delta; \Gamma, x : A_1 \vdash M : A_2}{\Delta; \Gamma \vdash \lambda x : A_1.\, M : A_1 \to A_2}$ TpLam

$\Delta'; \Gamma' \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$	by assumption
$\Gamma' \geq \Gamma'$	by application of CeBase
$\Rightarrow \quad \Gamma', x : A_1 \geq \Gamma'$	by application of CeInd
$\Rightarrow \quad \Delta'; \Gamma', x : A_1 \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$	by lemma 6.18 (2)
$\Rightarrow \quad (\Gamma', x : A_1)(x) = A_1$	by definition
$\Rightarrow \quad \Delta'; \Gamma', x : A_1 \vdash x : A_1$	by application of $\mathsf{TpVarReg}$
$\Rightarrow \quad \Delta'; \Gamma', x : A_1 \vdash (\theta; \varrho, x/x) \ : \ (\Delta; \Gamma, x : A_1)$	by application of TSReg
$\Rightarrow \quad \Delta'; \Gamma', x : A_1 \vdash [\theta; \varrho, x/x](M) : A_2$	by ind. hyp. (1)
$\Rightarrow \quad \Delta'; \Gamma' \vdash \lambda x : A_1.\, [\theta; \varrho, x/x](M) : A_1 \to A_2$	by application of TpLam
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\lambda x : A_1.\, M) : A_1 \to A_2$	by inversion using SBLam

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Delta;\Gamma \vdash M_1 : A_2 \to A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2 \\ \Delta;\Gamma \vdash M_2 : A_2\end{array}}{\Delta;\Gamma \vdash M_1\ M_2 : A_1}\ \mathsf{TpApp}$

$\Delta';\Gamma' \vdash [\theta;\varrho](M_1) : A_2 \to A_1$	by ind. hyp. (1)
$\Delta';\Gamma' \vdash [\theta;\varrho](M_2) : A_2$	by ind. hyp. (1)
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](M_1)\ [\theta;\varrho](M_2) : A_1$	by application of TpApp
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](M_1\ M_2) : A_1$	by inversion using SBApp

Case: $\mathcal{D} = \dfrac{\Delta;\cdot \vdash M : A}{\Delta;\Gamma \vdash \text{box } M : \Box A}\ \mathsf{TpBox}$

$\Delta';\Gamma' \vdash (\theta;\varrho)\ :\ (\Delta;\Gamma)$	by assumption
$\Rightarrow\ \Delta';\cdot \vdash (\theta;\cdot)\ :\ (\Delta;\cdot)$	by lemma 6.19
$\Rightarrow\ \Delta';\cdot \vdash [\theta;\cdot](M) : A$	by ind. hyp. (1)
$\Rightarrow\ \Delta';\Gamma' \vdash \text{box } [\theta;\cdot](M) : \Box A$	by application of TpBox
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](\text{box } M) : \Box A$	by definition SBBox

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Delta;\Gamma \vdash M_1 : \Box A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2 \\ \Delta, x : A_1;\Gamma \vdash M_2 : A_2\end{array}}{\Delta;\Gamma \vdash \text{let box } x = M_1 \text{ in } M_2 : A_2}\ \mathsf{TpLet}$

$\Delta';\Gamma' \vdash [\theta;\varrho](M_1) : \Box A_1$	by ind. hyp. on $\mathcal{D}_1$
$\Delta';\Gamma' \vdash (\theta;\varrho)\ :\ (\Delta;\Gamma)$	by assumption
$\Delta' \geq \Delta'$	by application of CeBase
$\Rightarrow\ \Delta', x : A_1 \geq \Delta'$	by application of CeInd
$\Rightarrow\ \Delta', x : A_1;\Gamma' \vdash (\theta;\varrho)\ :\ (\Delta;\Gamma)$	by lemma 6.18 (1)
$\Rightarrow\ (\Delta', x : A_1)(x) = A_1$	by definition
$\Rightarrow\ \Delta', x : A_1;\cdot \vdash x : A_1$	by application of $\mathsf{TpVarMod}$
$\Rightarrow\ \Delta', x : A_1;\Gamma' \vdash (\theta, x/x;\varrho)\ :\ (\Delta, x : A_1;\Gamma)$	by application of TSMod
$\Rightarrow\ \Delta', x : A_1;\Gamma' \vdash [\theta, x/x;\varrho](M_2) : A_2$	by ind. hyp. on $\mathcal{D}_2$
$\Rightarrow\ \Delta';\Gamma' \vdash \text{let box } x = [\theta;\varrho](M_1) \text{ in } [\theta, x/x;\varrho](M_2) : A_2$	by application of TpLet
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](\text{let box } x = M_1 \text{ in } M_2) : A_2$	by definition SBLet

Case: $\mathcal{D} = \dfrac{\Delta;\Gamma \vdash M : \Box B \qquad \Delta;\Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')}{\Delta;\Gamma \vdash \text{case } \langle A\rangle\ M\ \langle\Xi\rangle : \mathcal{C}^*\ (B, A, B)}\ \mathsf{TpCase}$

$\Delta;\Gamma \vdash M : \Box B$	by assumption
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](M) : \Box B$	by ind. hyp. (1)
$\Delta;\Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$	by assumption
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](\Xi) : \langle B \Rightarrow A\rangle(\Sigma')$	by ind. hyp. (2)
$\Rightarrow\ \Delta';\Gamma' \vdash \text{case } \langle A\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Xi)\rangle : \mathcal{C}^*\ (B, A, B)$	by application of TpCase
$\Rightarrow\ \Delta';\Gamma' \vdash [\theta;\varrho](\text{case } \langle A\rangle\ M\ \langle\Xi\rangle) : \mathcal{C}^*\ (B, A, B)$	by application of SBCase

Case: $\mathcal{D} = \dfrac{\Delta;\Gamma \vdash M : \Box B \qquad \vdash \omega : \alpha \qquad \Delta;\Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')}{\Delta;\Gamma \vdash \text{it } \langle\omega\rangle\ M\ \langle\Omega\rangle : \langle\omega\rangle(B)}\ \mathsf{TpIt}$

$\Delta;\Gamma \vdash M : \Box B$	by assumption

$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](M) : \Box B$ by ind. hyp. (1)
$\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\Omega) : \langle\omega\rangle(\Sigma')$ by ind. hyp. (3)
$\vdash \omega : \alpha$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash \text{it } \langle\omega\rangle \; [\theta; \varrho](M) \; \langle[\theta; \varrho](\Omega)\rangle : \langle\omega\rangle(B)$ by application of TpIt
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\text{it } \langle\omega\rangle \; M \; \langle\Omega\rangle) : \langle\omega\rangle(B)$ by application of SBIt

2. **Case:** $\mathcal{E} = \dfrac{}{\Delta; \Gamma \vdash \cdot : \langle B \Rightarrow A\rangle(\cdot)}\,\mathsf{TmBase}$

$\Delta'; \Gamma' \vdash \cdot : \langle B \Rightarrow A\rangle(\cdot)$ by application of TmBase
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\cdot) : \langle B \Rightarrow A\rangle(\cdot)$ by application of $\mathsf{SBXiEmpty}$

Case: $\mathcal{E} = \dfrac{\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma) \qquad \Delta; \Gamma \vdash M : \mathcal{C}\;(B, A, B')}{\Delta; \Gamma \vdash (\Xi \mid c \Rightarrow M) : \langle B \Rightarrow A\rangle(\Sigma, c : B')}\,\mathsf{TmInd}$

$\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma)$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\Xi) : \langle B \Rightarrow A\rangle(\Sigma)$ by ind. hyp. (2)
$\Delta; \Gamma \vdash M : \mathcal{C}\;(B, A, B')$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](M) : \mathcal{C}\;(B, A, B')$ by ind. hyp. (1)
$\Rightarrow \quad \Delta'; \Gamma' \vdash ([\theta; \varrho](\Xi) \mid c \Rightarrow [\theta; \varrho](M)) : \langle B \Rightarrow A\rangle(\Sigma, c : B')$ by application of TmInd
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\Xi \mid c \Rightarrow M) : \langle B \Rightarrow A\rangle(\Sigma, c : B')$ by application of SBXi

3. **Case:** $\mathcal{F} = \dfrac{}{\Delta; \Gamma \vdash \cdot : \langle\omega\rangle(\cdot)}\,\mathsf{TrBase}$

$\Delta'; \Gamma' \vdash \cdot : \langle\omega\rangle(\cdot)$ by application of TrBase
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\cdot) : \langle\omega\rangle(\cdot)$ by application of $\mathsf{SBOmegaEmpty}$

Case: $\mathcal{F} = \dfrac{\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma) \qquad \Delta; \Gamma \vdash M : \langle\omega\rangle(B')}{\Delta; \Gamma \vdash (\Omega \mid c \mapsto M) : \langle\omega\rangle(\Sigma, c : B')}\,\mathsf{TrInd}$

$\Delta; \Gamma \vdash \Omega : \langle\omega\rangle(\Sigma)$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\Omega) : \langle\omega\rangle(\Sigma)$ by ind. hyp. (3)
$\Delta; \Gamma \vdash M : \langle\omega\rangle(B')$ by assumption
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](M) : \langle\omega\rangle(B')$ by ind. hyp. (1)
$\Rightarrow \quad \Delta'; \Gamma' \vdash ([\theta; \varrho](\Omega) \mid c \mapsto [\theta; \varrho](M)) : \langle\omega\rangle(\Sigma, c : B')$ by application of TrInd
$\Rightarrow \quad \Delta'; \Gamma' \vdash [\theta; \varrho](\Omega \mid c \mapsto M) : \langle\omega\rangle(\Sigma, c : B')$ by application of $\mathsf{SBOmega}$

$\Box$

Lemma 6.25 (Type preservation of atomic and canonical types)

1. *If* $\Psi \vdash V \downarrow B$ *then* $\cdot; \Psi \vdash V : B$
2. *If* $\Psi \vdash V \Uparrow B$ *then* $\cdot; \Psi \vdash V : B$

Proof: by induction over $\mathcal{D} :: \Psi \vdash V \downarrow B$ and $\mathcal{E} :: \Psi \vdash V \Uparrow B$

Case: $\mathcal{D} = \dfrac{\Psi(x) = B}{\Psi \vdash x \downarrow B}\,\mathsf{AtVar}$

$\Psi(x) = B$ by assumption

$\Rightarrow\ \cdot; \Psi \vdash x : B$ by application of $\mathsf{TpVarReg}$

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\Psi \vdash c \downarrow B}\,\mathsf{AtConst}$

$\Sigma(c) = B$ by assumption

$\Rightarrow\ \cdot; \Psi \vdash c : B$ by application of $\mathsf{TpConst}$

Case: $\mathcal{D} = \dfrac{\Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash V_2 \Uparrow B_2}{\Psi \vdash V_1\ V_2 \downarrow B_1}\,\mathsf{AtApp}$

$\Psi \vdash V_1 \downarrow B_2 \to B_1$ by assumption

$\Rightarrow\ \cdot; \Psi \vdash V_1 : B_2 \to B_1$ by ind. hyp. (1)

$\Psi \vdash V_2 \Uparrow B_2$ by assumption

$\Rightarrow\ \cdot; \Psi \vdash V_2 : B_2$ by ind. hyp. (2)

$\Rightarrow\ \cdot; \Psi \vdash V_1\ V_2 : B_1$ by application of TpApp

Case: $\mathcal{D} = \dfrac{\Psi \vdash V \downarrow a}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt}$

$\Psi \vdash V \downarrow a$ by assumption

$\Rightarrow\ \cdot; \Psi \vdash V : a$ by ind. hyp. (1)

Case: $\mathcal{D} = \dfrac{\Psi, x : B_1 \vdash V \Uparrow B_2}{\Psi \vdash \lambda x{:}B_1.\, V \Uparrow B_1 \to B_2}\,\mathsf{CanLam}$

$\Psi, x : B_1 \vdash V \Uparrow B_2$ by assumption

$\Rightarrow\ \cdot; \Psi, x : B_1 \vdash V : B_2$ by ind. hyp. (2)

$\Rightarrow\ \cdot; \Psi \vdash \lambda x{:}B_1.\, V : B_1 \to B_2$ by application of TpLam

□

Lemma 6.28 (Goal types and abstraction closure types)

$\tau(\Pi\{\Psi\}.\, B) = \tau(B)$

Proof:

Case: $\Psi = \cdot$:

$\tau(\Pi\{\cdot\}.\, B) = \tau(B)$ by definition 5.9

Case: $\Psi = \Psi', x : B'$:

$\tau(\Pi\{\Psi', x : B'\}.\, B) = \tau(\Pi\{\Psi'\}.\, B' \to B)$ by definition 5.9

$\Rightarrow\ \tau(\Pi\{\Psi', x : B'\}.\, B) = \tau(B' \to B)$ by ind. hyp.

$\Rightarrow\ \tau(\Pi\{\Psi', x : B'\}.\, B) = \tau(B)$ by definition 4.15

□

Lemma 6.30 (Properties of subordination) *Let $c : C \in \Sigma$, B a pure type.*

1. *If $a \in Source(C)$ then $a \blacktriangleleft_B \tau(C)$*
2. *If $a_1 \blacktriangleleft_B a_2$ and $a_2 \trianglelefteq_B a_3$ then $a_1 \blacktriangleleft_B a_3$*

Proof:

1. $a \in \text{Source}(C)$ — by assumption
   $\Rightarrow\ a <_C \tau(C)$ — by definition 4.19
   $\Rightarrow\ a \lhd_\Sigma \tau(C)$ — by definition 4.20
   $\Rightarrow\ a \blacktriangleleft_B \tau(C)$ — by definition 4.24
2. **Case:** $a_2 \blacktriangleleft_B a_3$:
   $\Rightarrow\ a_1 \blacktriangleleft_B a_3$ — Transitivity
   Case: $a_2 = a_3$:
   $\Rightarrow\ a_1 \blacktriangleleft_B a_3$ — by definition

□

Lemma 6.32 (Properties of context subordination) *Let B be a pure type.*

If $\Psi = (\Psi_1, x : B') \cup \Psi_2$ and $\Psi \blacktriangleleft_B \tau(B)$ then $\tau(B') \trianglelefteq_B \tau(B)$ implies that forall $y \in Source(B')$: $y \blacktriangleleft_B \tau(B)$

Proof:

Case: $\Psi_2 = \cdot$

$\Psi = (\Psi_1, x : B') \cup \cdot$ — by assumption
$\Rightarrow\ \Psi = \Psi_1, x : B'$ — by definition CuBase
$\Rightarrow\ \Psi_1, x : B' \blacktriangleleft_B \tau(B)$ — by assumption
$\Rightarrow$ If $\tau(B') \trianglelefteq_B \tau(B)$ then forall $y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$ — by definition 6.31

Case: $\Psi_2 = \Psi_2', z : B''$, $x \neq z$

$\Psi = (\Psi_1, x : B') \cup (\Psi_2', z : B'')$ — by assumption
$\Psi = (\Psi_1, x : B') \cup \Psi_2', z : B''$ — by definition CuInd
$\Rightarrow$ If $\tau(B') \trianglelefteq_B \tau(B)$ then forall $y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$ — by ind. hyp.

□

Lemma 6.34 (Property of dynamic typing)

If $B' \in PCT(B)$ then $a <_{B'} \tau(B')$ implies $a \lhd_B \tau(B')$

Proof: proof by induction over B:

$B' \in \mathrm{PCT}(B)$	by assumption
$\Rightarrow \quad B = B_1 \to B_2$	by definition 4.22
$\Rightarrow \quad \mathrm{PCT}(B) = \{B_1\} \cup \mathrm{PCT}(B_2)$	by definition 4.22

Case: $B' = B_1$

$a <_{B_1} \tau(B_1)$	by assumption
$\Rightarrow \quad a \lhd_{B_1 \to B_2} \tau(B_1)$	by definition 4.23
$\Rightarrow \quad a \lhd_B \tau(B')$	by definition

Case: $B' \neq B_1$

$B' \in \mathrm{PCT}(B_2)$	by assumption
$a <_{B'} \tau(B')$	by assumption
$\Rightarrow \quad a \lhd_{B_2} \tau(B')$	by ind. hyp.
$\Rightarrow \quad a \lhd_{B_1 \to B_2} \tau(B')$	by definition 4.23
$\Rightarrow \quad a \lhd_B \tau(B')$	by definition

□

Lemma 6.35 (Independence)

If $\tau(B) \not\blacktriangleleft_B \tau(C)$ *then* $Source(C) \cap \mathcal{I}(\Sigma; B) = \emptyset$

Proof: proof by contradiction:

Suppose $\mathrm{Source}(C) \cap \mathcal{I}(\Sigma; B) \neq \emptyset$

Let $a \in \mathrm{Source}(C) \cap \mathcal{I}(\Sigma; B)$	by definition
$\Rightarrow \quad a \in \mathrm{Source}(C)$	by definition
$\Rightarrow \quad a \blacktriangleleft_B \tau(C)$	by lemma 6.30
$\Rightarrow \quad a \in \mathcal{I}(\Sigma; B)$	by definition
$\Rightarrow \quad \tau(B) \blacktriangleleft_B a$	by definition 4.25
$\Rightarrow \quad \tau(B) \blacktriangleleft_B \tau(C)$	by definition 4.24
$\Rightarrow \quad Contradiction$	

□

Lemma 6.36 (Properties of Join) *Let* $c : C \in \Sigma$, α *arbitrary and* $\vdash \omega : \alpha$

If $Source(C) \cap \alpha = \emptyset$ *and* $\tau(C) \notin \alpha$ *then* $\langle \omega \rangle (C) = C$

Proof: proof by induction on C:

Case: $C = a$:

$\tau(C) = a$	by definition 4.15
$\Rightarrow\ a \notin \alpha$	by assumption
$\Rightarrow\ \omega(a)$ not defined	
$\Rightarrow\ \langle\omega\rangle(a) = a$	by definition 4.26
$\Rightarrow\ \langle\omega\rangle(C) = \langle\omega\rangle(a) = a = C$	by definition

Case: $C = C_1 \to C_2$:

$\mathrm{Source}(C) = \mathrm{Source}(C_1) \cup \{\tau(C_1)\} \cup \mathrm{Source}(C_2)$	by definition 4.16
$\Rightarrow\ \mathrm{Source}(C_1) \subset \mathrm{Source}(C)$	by definition
$\Rightarrow\ \mathrm{Source}(C_1) \cap \alpha = \emptyset$	by definition
$\Rightarrow\ \tau(C_1) \in \mathrm{Source}(C)$	by definition
$\Rightarrow\ \tau(C_1) \notin \alpha$	by definition
$\Rightarrow\ \langle\omega\rangle(C_1) = C_1$	by ind. hyp.
$\Rightarrow\ \mathrm{Source}(C_2) \subset \mathrm{Source}(C)$	by definition
$\Rightarrow\ \mathrm{Source}(C_2) \cap \alpha = \emptyset$	by definition
$\Rightarrow\ \tau(C_2) = \tau(C_1 \to C_2)$	by definition 4.15
$\Rightarrow\ \tau(C_2) \notin \alpha$	by assumption
$\Rightarrow\ \langle\omega\rangle(C_2) = C_2$	by ind. hyp.
$\Rightarrow\ \langle\omega\rangle(C) = \langle\omega\rangle(C_1 \to C_2)$	by assumption
$\Rightarrow\ \langle\omega\rangle(C) = \langle\omega\rangle(C_1) \to \langle\omega\rangle(C_2)$	by definition 4.26
$\Rightarrow\ \langle\omega\rangle(C) = C_1 \to C_2$	by definition
$\Rightarrow\ \langle\omega\rangle(C) = C$	by assumption

□

Lemma 6.37 (Properties of subordination)

If $\tau(C) \notin \mathcal{I}(\Sigma; B)$ *and* $\tau(C) \trianglelefteq_B \tau(B)$ *then* $\tau(B) \not\blacktriangleleft_B \tau(C)$

Proof: proof by contradiction:

Case: $\tau(C) \blacktriangleleft_B \tau(B)$

Suppose $\tau(B) \blacktriangleleft_B \tau(C)$	by assumption
$\Rightarrow\ \tau(C) \in \mathcal{I}(\Sigma; B)$	by definition 4.25
$\Rightarrow$ Contradiction	

Case: $\tau(C) = \tau(B)$

Suppose $\tau(B) \blacktriangleleft_B \tau(C)$	by assumption
$\Rightarrow$ $\tau(C) \blacktriangleleft_B \tau(B)$	by definition
$\Rightarrow$ $\tau(C) \in \mathcal{I}(\Sigma; B)$	by definition 4.25
$\Rightarrow$ Contradiction	

$\square$

C Canonical form theorem

Lemma 7.1 (Self evaluation)

1. *If* $\Psi \vdash M \hookrightarrow V : B$ *and* $\Psi \vdash V \Uparrow B$ *then* $\Psi \vdash M \Uparrow V : B$
2. *If* $\Psi \vdash V \Uparrow B$ *then* $\Psi \vdash V \hookrightarrow V : B$
3. *If* $\Psi \vdash V \downarrow B$ *then* $\Psi \vdash V \hookrightarrow V : B$

Proof: by mutual induction over $\mathcal{D} :: \Psi \vdash V \Uparrow B$ and $\mathcal{E} :: \Psi \vdash V \Uparrow B$ and $\mathcal{F} :: \Psi \vdash V \downarrow B$

1. **Case:** $\mathcal{D} = \dfrac{\Psi \vdash V \downarrow a}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt}$

$\Psi \vdash M \hookrightarrow V : a$ — by assumption
$\Rightarrow \quad \Psi \vdash M \Uparrow V : a$ — by application of $\mathsf{EcAtomic}$

Case: $\mathcal{D} = \dfrac{\Psi, x : B_1 \vdash V_2 \Uparrow B_2}{\Psi \vdash \lambda x{:}B_1.\, V_2 \Uparrow B_1 \to B_2}\,\mathsf{CanLam}$

$\Psi \vdash M \hookrightarrow \lambda x{:}B_1.\, V_2 : B_1 \to B_2$ — by assumption
$\Psi \geq \Psi$ — by application of CeBase
$\Rightarrow \quad \Psi, x : B_1 \geq \Psi$ — by application of CeInd
$\Rightarrow \quad \Psi, x : B_1 \vdash M \hookrightarrow \lambda x{:}B_1.\, V_2 : B_1 \to B_2$ — by lemma 6.26
$\Rightarrow \quad \Psi, x : B_1 \vdash V_2 \hookrightarrow V_2 : B_2$ — by ind. hyp. (2)
$\Psi, x : B_1 \vdash x \hookrightarrow x : B_1$ — by application of EvVar
$\Rightarrow \quad \Psi, x : B_1 \vdash [x/x](V_2) \hookrightarrow V_2 : B_2$ — by definition
$\Rightarrow \quad \Psi, x : B_1 \vdash M\ x \hookrightarrow V_2 : B_2$ — by application of EvApp
$\Rightarrow \quad \Psi, x : B_1 \vdash M\ x \Uparrow V_2 : B_2$ — by ind. hyp. (1)
$\Rightarrow \quad \Psi \vdash M \Uparrow \lambda x{:}B_1.\, V_2 : B_2$ — by application of $\mathsf{EcArrow}$

2. **Case:** $\mathcal{E} = \dfrac{\Psi \vdash V \downarrow a}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt}$

$\Psi \vdash V \hookrightarrow V : a$ — by ind. hyp. (3)

Case: $\mathcal{E} = \dfrac{\Psi, x : B_1 \vdash V_2 \Uparrow B_2}{\Psi \vdash \lambda x{:}B_1.\, V_2 \Uparrow B_1 \to B_2}\,\mathsf{CanLam}$

$\cdot; \Psi, x : B_1 \vdash V_2 : B_2$ — by lemma 6.25
$\Rightarrow \quad \Psi \vdash \lambda x{:}B_1.\, V_2 \hookrightarrow \lambda x{:}B_1.\, V_2 : B_1 \to B_2$ — by application of EvLam

3. **Case:** $\mathcal{F} = \dfrac{\Psi(x) = B}{\Psi \vdash x \downarrow B}\,\mathsf{AtVar}$

$\Psi \vdash x \hookrightarrow x : B$ — by application of EvVar

Case: $\mathcal{F} = \dfrac{\Sigma(c) = B}{\Psi \vdash c \downarrow B}\,\mathsf{AtConst}$

$\Psi \vdash c \hookrightarrow c : B$ — by application of $\mathsf{EvConst}$

Case: $\mathcal{F} = \dfrac{\Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash V_2 \Uparrow B_2}{\Psi \vdash V_1\ V_2 \downarrow B_1}\,\mathsf{AtApp}$

$\Psi \vdash V_1 \hookrightarrow V_1 : B_2 \to B_1$ by ind. hyp. (3)

$\Rightarrow \quad \Psi \vdash V_2 \hookrightarrow V_2 : B_2$ by ind. hyp. (2)

$\Rightarrow \quad \Psi \vdash V_2 \Uparrow V_2 : B_2$ by ind. hyp. (1)

$\Rightarrow \quad \Psi \vdash V_1\ V_2 \hookrightarrow V_1\ V_2 : B_1$ by application of EvAtomic

□

Lemma 7.2 (Property of evaluation results)

1. *If* $\Psi \vdash M \Uparrow V : B$ *then* $\Psi \vdash V \Uparrow B$
2. *If* $\Psi \vdash M \hookrightarrow V : a$ *then* $\Psi \vdash V \downarrow a$

Proof: by mutual induction over $\mathcal{D} :: \Psi \vdash M \Uparrow V : B$ and $\mathcal{E} :: \Psi \vdash M \hookrightarrow V : B$

1. **Case:** $\mathcal{D} = \dfrac{\Psi \vdash M \hookrightarrow V : a}{\Psi \vdash M \Uparrow V : a}\,\mathsf{EcAtomic}$

   $\Psi \vdash V \downarrow a$ by ind. hyp. (2)

   $\Rightarrow \quad \Psi \vdash V \Uparrow a$ by application of CanAt

   Case: $\mathcal{D} = \dfrac{\Psi, x : B_1 \vdash M\ x \Uparrow V_2 : B_2}{\Psi \vdash M \Uparrow \lambda x\!:\!B_1.\, V_2 : B_1 \to B_2}\,\mathsf{EcArrow}$

   $\Psi, x : B_1 \vdash V_2 \Uparrow B_2$ by ind. hyp. (1)

   $\Rightarrow \quad \Psi \vdash \lambda x\!:\!B_1.\, V_2 \Uparrow B_1 \to B_2$ by application of CanLam

2. **Case:** $\mathcal{E} = \dfrac{\Psi(x) = a}{\Psi \vdash x \hookrightarrow x : a}\,\mathsf{EvVar}$

   $\Psi \vdash x \downarrow a$ by application of AtVar

   Case: $\mathcal{E} = \dfrac{\Sigma(c) = a}{\Psi \vdash c \hookrightarrow c : a}\,\mathsf{EvConst}$

   $\Psi \vdash c \downarrow a$ by application of AtConst

   Case: no rules for EvLam

   Case: $\mathcal{E} = \dfrac{\Psi \vdash M_1 \hookrightarrow \lambda x\!:\!A_2.\, M_1' : A_2 \to a \qquad \Psi \vdash M_2 \hookrightarrow V_2 : A_2 \qquad \overset{\mathcal{D}_1}{\Psi \vdash [V_2/x](M_1') \hookrightarrow V : a}}{\Psi \vdash M_1\ M_2 \hookrightarrow V : a}\,\mathsf{EvApp}$

   $\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_1$

   Case: $\mathcal{E} = \dfrac{\Psi \vdash M_1 \hookrightarrow V_1 : B_2 \to a \qquad \overset{\mathcal{D}_1}{\Psi \vdash V_1 \downarrow B_2 \to a} \qquad \overset{\mathcal{D}_2}{\Psi \vdash M_2 \Uparrow V_2 : B_2}}{\Psi \vdash M_1\ M_2 \hookrightarrow V_1\ V_2 : a}\,\mathsf{EvAtomic}$

   $\Psi \vdash V_2 \Uparrow B_2$ by ind. hyp. (1) on $\mathcal{D}_2$

   $\Rightarrow \quad \Psi \vdash V_1\ V_2 \downarrow a$ by application of AtApp on $\mathcal{D}_1$

Case: $\mathcal{E} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : a \times A_2 & \Psi \vdash M_1 \hookrightarrow V : a \end{array}}{\Psi \vdash \text{fst } M \hookrightarrow V : a}\ \mathsf{EvFst}$

$\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_2$

Case: $\mathcal{E} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Psi \vdash M \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times a & \Psi \vdash M_2 \hookrightarrow V : a \end{array}}{\Psi \vdash \text{snd } M \hookrightarrow V : a}\ \mathsf{EvSnd}$

$\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_2$

Case: no rules for EvBox

Case: $\mathcal{E} = \dfrac{\begin{array}{cc} & \mathcal{D}_1 \\ \Psi \vdash M_1 \hookrightarrow \text{box } M_1' : \Box A & \Psi \vdash [M_1'/x](M_2) \hookrightarrow V : a \end{array}}{\Psi \vdash \text{let box } x = M_1 \text{ in } M_2 \hookrightarrow V : a}\ \mathsf{EvLet}$

$\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_1$

Case: $\mathcal{E} = \dfrac{\begin{array}{ccc} & & \mathcal{D}_1 \\ \Psi \vdash M \hookrightarrow \text{box } M' : \Box B & \cdot \vdash M' \Uparrow V' : B & \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \hookrightarrow V : a \end{array}}{\Psi \vdash \text{case } \langle A \rangle\ M\ \langle \Xi \rangle \hookrightarrow V : a}\ \mathsf{EvCase}$

$\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_1$

Case: $\mathcal{E} = \dfrac{\begin{array}{ccc} & & \mathcal{D}_1 \\ \Psi \vdash M \hookrightarrow \text{box } M' : \Box B & \cdot \vdash M' \Uparrow V' : B & \Psi \vdash \langle \omega; \Omega \rangle(V') \hookrightarrow V : a \end{array}}{\Psi \vdash \text{it } \langle \omega \rangle\ M\ \langle \Omega \rangle \hookrightarrow V : a}\ \mathsf{EvIt}$

$\Psi \vdash V \downarrow a$ by ind. hyp. (1) on $\mathcal{D}_1$

□

Lemma 7.3 (Evaluation to atomic forms implies evaluation to canonical forms)

If $\Psi \vdash M \hookrightarrow V : B$ and $\Psi \vdash V \downarrow B$ then $\Psi \vdash M \Uparrow V' : B$ for a V'

Proof: by induction over B:

Case: $B = a$:

$\Psi \vdash M \hookrightarrow V : a$ by assumption

$\Rightarrow \ \Psi \vdash M \Uparrow V : a$ by application of $\mathsf{EcAtomic}$

Case: $B = B_1 \to B_2$:

Let $\Psi' = \Psi, x : B_1$ by definition

$\Rightarrow \ \Psi'(x) = B_1$ by definition 2.2

$\Rightarrow \ \Psi' \vdash x \hookrightarrow x : B_1$ by application of EvVar

$\Rightarrow \ \Psi' \vdash x \downarrow B_1$ by application of AtVar

$\Rightarrow \ \Psi' \vdash x \Uparrow V_x : B_1$ by ind. hyp.

$\Rightarrow \ \Psi' \vdash V_x \Uparrow B_1$ by lemma 7.2 (1)

$\Psi \vdash M \hookrightarrow V : B_1 \to B_2$	by assumption
$\Psi \geq \Psi$	by definition CeBase
$\Rightarrow \quad \Psi' \geq \Psi$	by definition CeInd
$\Rightarrow \quad \Psi' \vdash M \hookrightarrow V : B_1 \to B_2$	by lemma 6.26
$\Psi' \vdash V \downarrow B_1 \to B_2$	by assumption
$\Rightarrow \quad \Psi' \vdash M\ x \hookrightarrow V\ V_x : B_2$	by application of EvAtomic
$\Rightarrow \quad \Psi' \vdash V\ V_x \downarrow B_2$	by application of AtApp
$\Rightarrow \quad \Psi' \vdash M\ x \Uparrow V' : B_2$	by ind. hyp.
$\Rightarrow \quad \Psi, x : B_1 \vdash M\ x \Uparrow V' : B_2$	by definition
$\Rightarrow \quad \Psi \vdash M \Uparrow \lambda x{:}B_1.\, V' : B_1 \to B_2$	by application of EcArrow

□

Lemma 7.6 (Type preservation for replacements) *If* $\Psi + \cdot \vdash \Omega \in [\![\langle\omega\rangle(\Sigma;\cdot)]\!]$ *then* $\cdot;\Psi \vdash \Omega : \langle\omega\rangle(\Sigma)$

Proof: by induction over Σ

Case: $\Sigma = \cdot$

$\cdot;\Psi \vdash \cdot : \langle\omega\rangle(\cdot)$	by application of TrBase

Case: $\Sigma = \Sigma', c : B$

$\Psi + \cdot \vdash \Omega \in [\![\langle\omega\rangle(\Sigma', c : B;\cdot)]\!]$	by assumption
$\Rightarrow \quad \Omega = \Omega' \mid c \mapsto M$	by definition 7.5
$\Rightarrow \quad \Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$	by definition 7.5
$\Rightarrow \quad \cdot;\Psi \vdash M : \langle\omega\rangle(B)$	by definition 7.4
$\Rightarrow \quad \Psi + \cdot \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma';\cdot)]\!]$	by definition 7.5
$\Rightarrow \quad \cdot;\Psi \vdash \Omega' : \langle\omega\rangle(\Sigma')$	by ind. hyp.
$\Rightarrow \quad \cdot;\Psi \vdash (\Omega' \mid c \mapsto M) : \langle\omega\rangle(\Sigma', c : B)$	by application of TrInd

□

Lemma 7.8 (Type preservation for matches) *If* $\Psi + \cdot \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma;\cdot)]\!]$ *then* $\cdot;\Psi \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma)$

Proof: by induction over Σ

Case: $\Sigma = \cdot$

$\cdot;\Psi \vdash \cdot : \langle B \Rightarrow A\rangle(\cdot)$	by application of TmBase

Case: $\Sigma = \Sigma', c : B'$

$\Psi + \cdot \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma', c : B; \cdot)]\!]$	by assumption
$\Rightarrow \quad \Xi = \Xi' \mid c \Rightarrow M$	by definition 7.7
$\Rightarrow \quad \Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$	by definition 7.7
$\Rightarrow \quad \cdot; \Psi \vdash M : \mathcal{C}\ (B, A, B')$	by definition 7.4
$\Rightarrow \quad \Psi + \cdot \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$	by definition 7.7
$\Rightarrow \quad \cdot; \Psi \vdash \Xi' : \langle B \Rightarrow A\rangle(\Sigma')$	by ind. hyp.
$\Rightarrow \quad \cdot; \Psi \vdash (\Xi' \mid c \Rightarrow M) : \langle B \Rightarrow A\rangle(\Sigma', c : B')$	by application of TmInd

□

Lemma 7.9 (Weakening for logical relations)

1. *If* $\Psi \vdash M \in [\![A]\!]$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash M \in [\![A]\!]$
2. *If* $\Psi \vdash V \in |A|$ *and* $\Psi' \geq \Psi$ *then* $\Psi' \vdash V \in |A|$

Proof: by induction over A:

1. $\Psi \vdash M \in [\![A]\!]$	by assumption
$\Rightarrow \quad \cdot; \Psi \vdash M : A$	by definition 7.4
$\Rightarrow \quad \Psi \vdash M \hookrightarrow V : A$	by definition 7.4
$\Rightarrow \quad \Psi \vdash V \in \|A\|$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi' \vdash M : A$	by lemma 6.8
$\Rightarrow \quad \Psi' \vdash M \hookrightarrow V : A$	by lemma 6.26
$\Rightarrow \quad \Psi' \vdash V \in \|A\|$	by ind. hyp. (2)
$\Psi' \vdash M \in [\![A]\!]$	by definition 7.4

2. **Case:** $A = a$:

$\Psi \vdash V \Uparrow a$	by definition 7.4
$\Psi' \vdash V \Uparrow a$	by lemma 6.24 (2)
$\Psi' \vdash V \in \|a\|$	by definition 7.4

Case: $A = A_1 \rightarrow A_2$:

Case: $V = \lambda x : A_1 .\, M$:

Let $\Psi'' \geq \Psi'$ and $\Psi \vdash V' \in \|A_1\|$	
$\Rightarrow \quad \Psi'' \geq \Psi$	by lemma 6.3
$\Rightarrow \quad \Psi'' \vdash [V'/x](M) \in [\![A_2]\!]$	by definition 7.4
$\Rightarrow \quad \Psi' \vdash \lambda x : A_1 .\, M \in \|A_1 \rightarrow A_2\|$	by definition 7.4

Case: $\Psi \vdash V \downarrow A_1 \rightarrow A_2$:

Let $\Psi'' \geq \Psi'$ and $\Psi \vdash V' \in \|A_1\|$	
$\Rightarrow \quad \Psi'' \geq \Psi$	by lemma 6.3
$\Rightarrow \quad \Psi'' \vdash V\ V' \in \|A_2\|$	by definition 7.4
$\Rightarrow \quad \Psi' \vdash V \in \|A_1 \rightarrow A_2\|$	by definition 7.4

Case: $A = \Box A'$:

$\Rightarrow \quad \Psi \vdash \text{box } M \in |\Box A'|$ — by assumption

$\Rightarrow \quad \cdot \vdash M \in |A'|$ — by definition 7.4

$\Rightarrow \quad \Psi' \vdash \text{box } M \in |\Box A'|$ — by definition 7.4

$\Box$

Lemma 7.10 (Weakening for logical relations for replacements)

If $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *and* $\tilde{\Psi}' \geq \tilde{\Psi}$ *then* $\Psi + \tilde{\Psi}' \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$

Proof: by induction over $\Sigma, \hat{\Psi}$:

Case: $\Sigma = \cdot, \hat{\Psi} = \cdot$:

$\Omega = \cdot$ — by definition 7.5

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \cdot \in [\![\langle\omega\rangle(\cdot; \cdot)]\!]$ — by definition 7.5

Case: $\Sigma = \Sigma', c : B, \hat{\Psi} = \cdot$:

$\Omega = \Omega' \mid c \mapsto M$ — by definition 7.5

$\Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ — by definition 7.5

$\Psi + \tilde{\Psi} \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma'; \cdot)]\!]$ — by definition 7.5

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma'; \cdot)]\!]$ — by ind. hyp.

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Omega' \mid c \mapsto M \in [\![\langle\omega\rangle(\Sigma', c : B; \cdot)]\!]$ — by definition 7.5

Case: $\hat{\Psi} = \hat{\Psi}', x : B$:

$\Omega = \Omega' \mid x \mapsto u$ — by definition 7.5

$\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ — by definition 7.5

$\Psi + \tilde{\Psi} \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi}')]\!]$ — by definition 7.5

$\Rightarrow \quad \tilde{\Psi}' \vdash u \in [\![\langle\omega\rangle(B)]\!]$ — by lemma 7.9 (1)

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Omega' \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi}')]\!]$ — by ind. hyp.

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Omega' \mid x \mapsto u \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi}', x : B)]\!]$ — by definition 7.5

$\Box$

Lemma 7.11 (Weakening for logical relations for matches)

If $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ *and* $\tilde{\Psi}' \geq \tilde{\Psi}$ *then* $\Psi + \tilde{\Psi}' \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$

Proof: by induction over $\Sigma, \hat{\Psi}$:

Case: $\Sigma = \cdot, \hat{\Psi} = \cdot$:

$\Xi = \cdot$ — by definition 7.7

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \cdot \in [\![\langle B \Rightarrow A\rangle(\cdot; \cdot)]\!]$ — by definition 7.7

Case: $\Sigma = \Sigma', c : B', \hat{\Psi} = \cdot$:

$\Xi = \Xi' \mid c \mapsto M$ by definition 7.7

$\Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$ by definition 7.7

$\Psi + \tilde{\Psi} \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$ by definition 7.7

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$ by ind. hyp.

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Xi' \mid c \mapsto M \in [\![\langle B \Rightarrow A\rangle(\Sigma', c : B'; \cdot)]\!]$ by definition 7.7

Case: $\hat{\Psi} = \hat{\Psi}', x : B'$:

$\Xi = \Xi' \mid x \mapsto u$ by definition 7.7

$\tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$ by definition 7.7

$\Psi + \tilde{\Psi} \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi}')]\!]$ by definition 7.7

$\Rightarrow \quad \tilde{\Psi}' \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$ by lemma 7.9 (1)

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Xi' \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi}')]\!]$ by ind. hyp.

$\Rightarrow \quad \Psi + \tilde{\Psi}' \vdash \Xi' \mid x \mapsto u \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi}', x : B')]\!]$ by definition 7.7

□

Lemma 7.12 (Access to logical relations for replacements I) *If* $\Sigma = \Sigma_1, c : B \cup \Sigma_2$ *and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ *and* $M = \langle\omega; \Omega\rangle(c)$

Proof: by induction over the structure of $\hat{\Psi}$, Σ_2:

Case: $\hat{\Psi} = \cdot$:

Case: $\Sigma_2 = \cdot$:

$\Sigma = \Sigma_1, x : B$ by definition CuBase

$\Rightarrow \quad \Omega = \Omega' \mid c \mapsto M$ by definition 7.5

$\Rightarrow \quad \langle\omega; \Omega\rangle(c) = M$ by definition ElVar

$\Rightarrow \quad \Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ by definition 7.5

Case: $\Sigma_2 = \Sigma_2', c' : B'$:

$\Sigma = \Sigma_1, x : B \cup \Sigma_2', y : B'$ by definition

$\Rightarrow \quad \Sigma = (\Sigma_1, x : B \cup \Sigma_2'), y : B'$ by definition CuInd

$\Rightarrow \quad \Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ by ind. hyp.

$\Rightarrow \quad M = \langle\omega; \Omega\rangle(c)$ by ind. hyp.

Case: $\hat{\Psi} = \hat{\Psi}', y : B'$:

$\Rightarrow \quad \Psi \vdash M \in [\![\langle\omega\rangle(B)]\!]$ by ind. hyp.

$\Rightarrow \quad M = \langle\omega; \Omega\rangle(c)$ by ind. hyp.

□

Lemma 7.13 (Access to logical relations for replacements II) *If* $\Sigma(c)$ *is undefined and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Omega(c)$ *is undefined*

Proof: by induction over the structure of $\hat{\Psi}$, Σ:

Case: $\hat{\Psi} = \cdot$:

Case: $\Sigma = \cdot$:

$\Sigma(c)$ is undefined — by definition

Case: $\Sigma = \Sigma', c' : B'$:

$\Sigma(c)$ is undefined — by assumption

$\Rightarrow$ $(\Sigma', c' : B')(c)$ is undefined — by definition

$\Rightarrow$ $\Sigma'(c)$ is undefined — by definition

$\Rightarrow$ $\Omega(c)$ is undefined — by ind. hyp.

Case: $\hat{\Psi} = \hat{\Psi}', y : B'$:

$\Rightarrow$ $\Omega(c)$ is undefined — by ind. hyp.

□

Lemma 7.14 (Access to logical relations for replacements III) *If* $\hat{\Psi} = \hat{\Psi}_1, x : B \cup \hat{\Psi}_2$ *and* $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ *and* $\tilde{\Psi} = \tilde{\Psi}_1, u : \langle\omega\rangle(B) \cup \tilde{\Psi}_2$ *and* $u = \langle\omega; \Omega\rangle(x)$

Proof: by induction over the structure of $\hat{\Psi}_2$:

Case: $\hat{\Psi}_2 = \cdot$:

$\hat{\Psi} = \hat{\Psi}_1, x : B$ — by definition CuBase

$\Rightarrow$ $\Omega = \Omega' \mid x \mapsto u$ — by definition 7.5

$\Rightarrow$ $\langle\omega; \Omega\rangle(x) = u$ — by definition ElVar

$\Rightarrow$ $\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ — by definition 7.5

$\Rightarrow$ $\tilde{\Psi} \vdash u \hookrightarrow u : \langle\omega\rangle(B)$ — by definition 7.4

$\Rightarrow$ $\tilde{\Psi}(u) = \langle\omega\rangle(B)$ — by inversion using EvVar

$\Rightarrow$ $\tilde{\Psi} = \tilde{\Psi}_1, u : \langle\omega\rangle(B) \cup \tilde{\Psi}_2$ — by definition CuInd

Case: $\hat{\Psi}_2 = \hat{\Psi}'_2, y : B'$:

$\hat{\Psi} = \hat{\Psi}_1, x : B \cup \hat{\Psi}'_2, y : B'$ — by definition

$\Rightarrow$ $\hat{\Psi} = (\hat{\Psi}_1, x : B \cup \hat{\Psi}'_2), y : B'$ — by definition CuInd

$\Rightarrow$ $\tilde{\Psi} \vdash u \in [\![\langle\omega\rangle(B)]\!]$ — by ind. hyp.

$\Rightarrow$ $\tilde{\Psi} = \tilde{\Psi}_1, u : \langle\omega\rangle(B) \cup \tilde{\Psi}'_2$ — by ind. hyp.

$\Rightarrow$ $u = \langle\omega; \Omega\rangle(x)$ — by ind. hyp.

□

Lemma 7.15 (Access to logical relations for matches I) *If* $\Sigma = \Sigma_1, c : B' \cup \Sigma_2$ *and* $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$ *and* $M = \{B \Rightarrow A; \Xi; \Psi'\}(c)$ *for an arbitrary* Ψ'.

Proof: by induction over the structure of $\hat{\Psi}$, Σ_2:

Case: $\hat{\Psi} = \cdot$:

Case: $\Sigma_2 = \cdot$:

$\Sigma = \Sigma_1, x : B'$	by definition CuBase
$\Rightarrow \quad \Xi = \Xi' \mid c \mapsto M$	by definition 7.7
$\Rightarrow \quad \{B \Rightarrow A; \Xi; \Psi'\}(c) = M$	by definition SeVar
$\Rightarrow \quad \Psi \vdash M \in [\![\mathcal{C}\ (B, A, B)]\!]$	by definition 7.7

Case: $\Sigma_2 = \Sigma_2', y : B''$:

$\Sigma = \Sigma_1, x : B' \cup \Sigma_2', y : B''$	by definition
$\Rightarrow \quad \Sigma = (\Sigma_1, x : B' \cup \Sigma_2'), y : B''$	by definition CuInd
$\Rightarrow \quad \Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$	by ind. hyp.
$\Rightarrow \quad M = \{B \Rightarrow A; \Xi; \Psi'\}(c)$	by ind. hyp.

Case: $\hat{\Psi} = \hat{\Psi}', y : B''$:

$\Rightarrow \quad \Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$	by ind. hyp.
$\Rightarrow \quad M = \{B \Rightarrow A; \Xi; \Psi'\}(c)$	by ind. hyp.

□

Lemma 7.16 (Access to logical relations for matches I) *If* $\hat{\Psi} = \hat{\Psi}_1, x : B' \cup \hat{\Psi}_2$ *and* $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma; \hat{\Psi})]\!]$ *then* $\tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$ *and* $\tilde{\Psi} = \tilde{\Psi}_1, u : \mathcal{C}\ (B, A, B') \cup \tilde{\Psi}_2$ *and* $u = \{B \Rightarrow A; \Xi; \Psi'\}(x)$ *for an arbitrary* Ψ'*.*

Proof: by induction over the structure of $\hat{\Psi}_2$:

Case: $\hat{\Psi}_2 = \cdot$:

$\hat{\Psi} = \hat{\Psi}_1, x : B'$	by definition CuBase
$\Rightarrow \quad \Xi = \Xi' \mid x \mapsto u$	by definition 7.7
$\Rightarrow \quad \{B \Rightarrow A; \Xi; \Psi'\}(x) = u$	by definition SeVar
$\Rightarrow \quad \tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$	by definition 7.7
$\Rightarrow \quad \tilde{\Psi} \vdash u \hookrightarrow u : \mathcal{C}\ (B, A, B')$	by definition 7.4
$\Rightarrow \quad \tilde{\Psi}(u) = \mathcal{C}\ (B, A, B')$	by inversion using EvVar
$\Rightarrow \quad \tilde{\Psi} = \tilde{\Psi}_1, u : \mathcal{C}\ (B, A, B') \cup \tilde{\Psi}_2$	by definition CuInd

Case: $\hat{\Psi}_2 = \hat{\Psi}_2', y : B''$:

$\hat{\Psi} = \hat{\Psi}_1, x : B' \cup \hat{\Psi}_2', y : B''$	by definition
$\Rightarrow \quad \hat{\Psi} = (\hat{\Psi}_1, x : B' \cup \hat{\Psi}_2'), y : B''$	by definition CuInd
$\Rightarrow \quad \tilde{\Psi} \vdash u \in [\![\mathcal{C}\ (B, A, B')]\!]$	by ind. hyp.
$\Rightarrow \quad \tilde{\Psi} = \tilde{\Psi}_1, u : \mathcal{C}\ (B, A, B') \cup \tilde{\Psi}_2'$	by ind. hyp.
$\Rightarrow \quad u = \{B \Rightarrow A; \Xi; \Psi'\}(x)$	by ind. hyp.

□

Lemma 7.17 (Logical relations and canonical forms)

1. *If* $\Psi \vdash M \in [\![B]\!]$ *then* $\Psi \vdash M \Uparrow V : B$
2. *If* $\Psi \vdash V \downarrow B$ *then* $\Psi \vdash V \in |B|$

Proof: by induction over B:

Case: $B = a$:

1. $\Psi \vdash M \in [\![a]\!]$	by assumption
$\Rightarrow$ $\Psi \vdash M \hookrightarrow V' : a$ and $\Psi \vdash V' \in \lvert a \rvert$	by definition 7.4
$\Rightarrow$ $\Psi \vdash V' \Uparrow a$	by definition 7.4
$\Rightarrow$ $\Psi \vdash V' \downarrow a$	by inversion using CanAt
$\Rightarrow$ $\Psi \vdash M \Uparrow V : a$	by lemma 7.3
2. $\Psi \vdash V \downarrow a$	by assumption
$\Rightarrow$ $\Psi \vdash V \Uparrow a$	by application of CanAt
$\Rightarrow$ $\Psi \vdash V \in \lvert a \rvert$	by definition 7.4

Case: $B = B_1 \rightarrow B_2$:

1. $\Psi \vdash M \in [\![B_1 \rightarrow B_2]\!]$	by assumption
$\Rightarrow$ $\cdot; \Psi \vdash M : B_1 \rightarrow B_2$	by definition 7.4
$\Rightarrow$ $\Psi \vdash M \hookrightarrow V : B_1 \rightarrow B_2$	by definition 7.4
$\Rightarrow$ $\Psi \vdash V \in \lvert B_1 \rightarrow B_2 \rvert$	by definition 7.4
Case: $V = \lambda x{:}B_1.\, M'$	by definition 7.4
Let $\Psi' = \Psi, x : B_1$	by definition
$\Rightarrow$ $\Psi'(x) = B_1$	by definition
$\Rightarrow$ $\Psi' \vdash x \hookrightarrow x : B_1$	by application of EvVar
$\Rightarrow$ $\Psi' \vdash x \downarrow B_1$	by application of AtVar
$\Rightarrow$ $\Psi' \vdash x \in \lvert B_1 \rvert$	by ind. hyp. (2)
$\Rightarrow$ $\Psi' \vdash [x/x](M') \in [\![B_2]\!]$	by definition 7.4
$\Rightarrow$ $\Psi' \vdash [x/x](M') \hookrightarrow V' : B_2$	by definition 7.4
$\Rightarrow$ $\Psi' \vdash V' \in \lvert B_2 \rvert$	by definition 7.4
$\Rightarrow$ $\Psi \geq \Psi$	by application of CeBase
$\Rightarrow$ $\Psi' \geq \Psi$	by application of CeInd
$\Rightarrow$ $\Psi' \vdash M \hookrightarrow V : B_1 \rightarrow B_2$	by lemma 6.26
$\Rightarrow$ $\Psi' \vdash M\ x \hookrightarrow V' : B_2$	by application of EvApp
$\Rightarrow$ $\cdot; \Psi' \vdash x : B_1$	by application of TpVarReg
$\Rightarrow$ $\cdot; \Psi' \vdash M : B_1 \rightarrow B_2$	by lemma 6.8 (2)
$\Rightarrow$ $\cdot; \Psi' \vdash M\ x : B_2$	by application of TpApp
$\Rightarrow$ $\Psi' \vdash M\ x \in [\![B_2]\!]$	by definition 7.4
$\Rightarrow$ $\Psi' \vdash M\ x \Uparrow V' : B_2$	by ind. hyp. (1)
$\Rightarrow$ $\Psi, x : B_1 \vdash M\ x \Uparrow V' : B_2$	by definition
$\Rightarrow$ $\Psi \vdash M \Uparrow \lambda x{:}B_1.\, V' : B_2$	by application of EcArrow

Case: $\Psi \vdash V \downarrow B_1 \to B_2$	by definition 7.4
Let $\Psi' = \Psi, x : B_1$	by definition
$\Rightarrow \quad \Psi'(x) = B_1$	by definition
$\Rightarrow \quad \Psi' \vdash x \hookrightarrow x : B_1$	by application of EvVar
$\Rightarrow \quad \Psi' \vdash x \downarrow B_1$	by application of AtVar
$\Rightarrow \quad \Psi' \vdash x \Uparrow V_x : B_1$	by lemma 7.3
$\Rightarrow \quad \Psi \geq \Psi$	by application of CeBase
$\Rightarrow \quad \Psi' \geq \Psi$	by application of CeInd
$\Rightarrow \quad \Psi' \vdash M \hookrightarrow V : B_1 \to B_2$	by lemma 6.26
$\Rightarrow \quad \Psi' \vdash M\ x \hookrightarrow V\ V_x : B_2$	by application of EvAtomic
$\Rightarrow \quad \Psi' \vdash V_x \Uparrow B_1$	by lemma 7.2 (1)
$\Rightarrow \quad \Psi' \vdash V\ V_x \in \lvert B_1 \to B_2 \rvert$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi' \vdash x : B_1$	by application of TpVarReg
$\Rightarrow \quad \cdot; \Psi' \vdash M : B_1 \to B_2$	by lemma 6.8 (2)
$\Rightarrow \quad \cdot; \Psi' \vdash M\ x : B_2$	by application of TpApp
$\Rightarrow \quad \Psi' \vdash M\ x \in [\![B_2]\!]$	by definition 7.4
$\Rightarrow \quad \Psi' \vdash M\ x \Uparrow V' : B_2$	by ind. hyp.
$\Rightarrow \quad \Psi, x : B_1 \vdash M\ x \Uparrow V' : B_2$	by definition
$\Rightarrow \quad \Psi \vdash M \Uparrow \lambda x{:}B_1 . V' : B_2$	by application of EcArrow

2. $\Psi \vdash V \downarrow B_1 \to B_2$

Let Ψ' context, s.t. $\Psi' \geq \Psi$	
Let V' s.t. $\Psi' \vdash V' \Uparrow B_1$	
$\Rightarrow \quad \Psi' \vdash V \downarrow B_1 \to B_2$	by lemma 6.24
$\Rightarrow \quad \Psi' \vdash V\ V' \downarrow B_2$	by application of AtApp
$\Rightarrow \quad \Psi' \vdash V\ V' \in \lvert B_2 \rvert$	by ind. hyp. (2)
$\Rightarrow \quad \Psi \vdash V \in \lvert B_1 \to B_2 \rvert$	by definition 7.4

□

Lemma 7.18 (Types of atomic objects are pure)

If $\Psi \vdash V \downarrow A$ then A is pure.

Proof: by induction over $\mathcal{D} :: \Psi \vdash V \downarrow A$:

Case: $\mathcal{D} = \dfrac{\Psi(x) = A}{\Psi \vdash x \downarrow A}\,\mathsf{AtVar}$

$\Psi(x) = B$	by assumption on Ψ
$\Rightarrow \quad A = B$ and A is pure	by definition signature

Case: $\mathcal{D} = \dfrac{\Sigma(c) = A}{\Psi \vdash c \downarrow A}\,\mathsf{AtConst}$

$\Sigma(c) = B$	by assumption on Σ
$\Rightarrow \quad A = B$ and A is pure	by definition context

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ \Psi \vdash V_1 \downarrow B_2 \rightarrow A\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Psi \vdash V_2 \Uparrow B_2\end{array}}{\Psi \vdash V_1\, V_2 \downarrow A}\,\mathsf{AtApp}$

$B_2 \rightarrow A$ is pure	by ind. hyp. on $\mathcal{D}_1$
$\Rightarrow$ A is pure	by definition pure types

□

Lemma 7.19 (Well-typedness of logical relations)

If $\Psi \vdash V \in |A|$ *then* $\cdot; \Psi \vdash V : A$

Proof: by induction over A:

Case: $A = a$:

$\Psi \vdash V \in \lvert a \rvert$	by assumption
$\Rightarrow$ $\Psi \vdash V \Uparrow a$	by definition 7.4
$\Rightarrow$ $\cdot; \Psi \vdash V : a$	by lemma 6.25

Case: $A = A_1 \rightarrow A_2$:

$\Psi \vdash V \in \lvert A_1 \rightarrow A_2 \rvert$	by assumption
Case: $V = \lambda x{:}A_1.\, M$	by definition 7.4
Let $\Psi' = \Psi, x : A_1$	by definition
$\Rightarrow$ $\Psi'(x) = A_1$	by definition
$\Rightarrow$ $\Psi' \vdash x \downarrow A_1$	by application of AtVar
$\Rightarrow$ $\Psi' \vdash x \in \lvert A_1 \rvert$	by lemma 7.17
$\Rightarrow$ $\Psi' \vdash \underbrace{[x/x](M)}_{=M} \in [\![A_2]\!]$	by definition 7.4
$\Rightarrow$ $\cdot; \Psi, x : A_1 \vdash M : A_2$	by definition 7.4
$\Rightarrow$ $\cdot; \Psi \vdash \lambda x{:}A_1.\, M : A_1 \rightarrow A_2$	by application of TpLam
Case: $\Psi \vdash V \downarrow A_1 \rightarrow A_2$	by definition 7.4
$\Rightarrow$ $\Psi \vdash V \downarrow B_1 \rightarrow B_2$ and $A_1 = B_1, A_2 = B_2$	by lemma 7.18
$\Rightarrow$ $\cdot; \Psi \vdash V : B_1 \rightarrow B_2$	by lemma 6.25

Case: $A = \Box A'$:

$\Psi \vdash V \in \lvert \Box A' \rvert$	by assumption
$\Rightarrow$ $V = \text{box } M$	by definition 7.4
$\Rightarrow$ $\Psi \vdash \text{box } M \in \lvert \Box A' \rvert$	by definition 7.4
$\Rightarrow$ $\cdot \vdash M \in [\![A']\!]$	by definition 7.4
$\Rightarrow$ $\cdot; \cdot \vdash M : A'$	by definition 7.4
$\Rightarrow$ $\cdot; \Psi \vdash \text{box } M : \Box A'$	by application of TpBox

□

Lemma 7.20 (Logical relations: Self evaluation of values)

If $\Psi \vdash V \in |A|$ *then* $\Psi \vdash V \hookrightarrow V : A$

Proof: by induction over A:

Case: $A = a$:

$\Psi \vdash V \in \lvert a \rvert$	by assumption
$\Rightarrow \quad \Psi \vdash V \Uparrow a$	by definition 7.4
$\Rightarrow \quad \Psi \vdash V \hookrightarrow V : a$	by lemma 7.1

Case: $A = A_1 \to A_2$:

$\Psi \vdash V \in \lvert A_1 \to A_2 \rvert$	by assumption
Case: $V = \lambda x{:}A_1.\, M$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi \vdash \lambda x{:}A_1.\, M : A_1 \to A_2$	by lemma 7.19
$\Rightarrow \quad \Psi \vdash \lambda x{:}A_1.\, M \hookrightarrow \lambda x{:}A_1.\, M : A_1 \to A_2$	by application of EvLam
Case: $\Psi \vdash V \downarrow A_1 \to A_2$	by definition 7.4
$\Rightarrow \quad \Psi \vdash V \downarrow B_1 \to B_2$ and $A_1 = B_1, A_2 = B_2$	by lemma 7.18
$\Rightarrow \quad \Psi \vdash V \hookrightarrow V : B_1 \to B_2$	by lemma 7.1

Case: $A = A_1 \times A_2$:

$\Psi \vdash V \in \lvert A_1 \times A_2 \rvert$	by assumption
$\Rightarrow \quad V = \langle M_1, M_2 \rangle$	by definition 7.4
$\Rightarrow \quad \Psi \vdash M_1 \in [\![A_1]\!]$	by definition 7.4
$\Rightarrow \quad \Psi \vdash M_2 \in [\![A_2]\!]$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi \vdash M_1 : A_1$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi \vdash M_2 : A_2$	by definition 7.4
$\Rightarrow \quad \Psi \vdash \langle M_1, M_2 \rangle \hookrightarrow \langle M_1, M_2 \rangle : A_1 \times A_2$	by application of EvPair

Case: $A = \Box A'$:

$\Psi \vdash V \in \lvert \Box A' \rvert$	by assumption
$\Rightarrow \quad V = \mathrm{box}\ M$	by definition 7.4
$\Rightarrow \quad \Psi \vdash \mathrm{box}\ M \in \lvert \Box A' \rvert$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi \vdash \mathrm{box}\ M : \Box A'$	by lemma 7.19
$\Rightarrow \quad \cdot; \cdot \vdash M : A'$	by inversion using TpBox
$\Rightarrow \quad \Psi \vdash \mathrm{box}\ M \hookrightarrow \mathrm{box}\ M : \Box A'$	by application of EvBox

□

Lemma 7.21 (Logical relation subsumption)

If $\Psi \vdash V \in |A|$ *then* $\Psi \vdash V \in [\![A]\!]$

Proof: $\Psi \vdash V \in |A|$ by assumption
$\Rightarrow \Psi \vdash V \hookrightarrow V : A$ by lemma 7.20
$\Rightarrow \cdot; \Psi \vdash V : A$ by lemma 7.19
$\Rightarrow \Psi \vdash V \in [\![A]\!]$ by definition 7.4
$\Box$

Lemma 7.22 (Logical relation is closed under application)

If $\Psi \vdash M_1 \in [\![A_2 \to A_1]\!]$ *and* $\Psi \vdash M_2 \in [\![A_2]\!]$ *then* $\Psi \vdash M_1\ M_2 \in [\![A_1]\!]$

Proof: $\Psi \vdash M_1 \in [\![A_2 \to A_1]\!]$ by assumption
$\Rightarrow \cdot; \Psi \vdash M_1 : A_2 \to A_1$ by definition 7.4
$\Rightarrow \Psi \vdash M_1 \hookrightarrow V_1 : A_2 \to A_1$ by definition 7.4
$\Rightarrow \Psi \vdash V_1 \in |A_2 \to A_1|$ by definition 7.4
$\Psi \vdash M_2 \in [\![A_2]\!]$ by assumption
$\Rightarrow \cdot; \Psi \vdash M_2 : A_2$ by definition 7.4
$\Rightarrow \Psi \vdash M_2 \hookrightarrow V_2 : A_2$ by definition 7.4
$\Rightarrow \Psi \vdash V_2 \in |A_2|$ by definition 7.4
$\Rightarrow \cdot; \Psi \vdash M_1\ M_2 : A_1$ by application of TpApp

Case: $V_1 = \lambda x : A_2.\, M_1'$
$\Rightarrow \Psi \vdash [V_2/x](M_1') \in [\![A_1]\!]$ by definition 7.4
$\Rightarrow \cdot; \Psi \vdash [V_2/x](M_1') : A_1$ by definition 7.4
$\Rightarrow \Psi \vdash [V_2/x](M_1') \hookrightarrow V : A_1$ by definition 7.4
$\Rightarrow \Psi \vdash V \in |A_1|$ by definition 7.4
$\Rightarrow \Psi \vdash M_1\ M_2 \hookrightarrow V : A_1$ by application of EvApp
$\Rightarrow \Psi \vdash M_1\ M_2 \in [\![A_1]\!]$ by definition 7.4

Case: $\Psi \vdash V_1 \downarrow A_2 \to A_1$
$\Rightarrow \Psi \vdash V_1 \downarrow B_2 \to B_1$ and $A_2 = B_2, A_1 = B_1$ by lemma 7.18
$\Rightarrow \Psi \vdash M_2 \Uparrow V_2' : B_2$ by lemma 7.17 (1)
$\Rightarrow \Psi \vdash V_2' \Uparrow B_2$ by lemma 7.2 (1)
$\Rightarrow \Psi \vdash V_1\ V_2' \in |B_1|$ by definition 7.4
$\Rightarrow \Psi \vdash M_1\ M_2 \hookrightarrow V_1\ V_2' : B_1$ by application of EvAtomic
$\Rightarrow \Psi \vdash M_1\ M_2 \in [\![B_1]\!]$ by definition 7.4

$\Box$

Lemma 7.27 (Modal substitution restriction)

If $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\cdot \vdash \theta; \cdot \in [\Delta; \cdot]$

Proof: $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ by assumption

$\Rightarrow \vdash \theta \in [\![\Delta]\!]$ by definition 7.23

$\Rightarrow \cdot \vdash \cdot \in |\cdot|$ by definition 7.24

$\Rightarrow \cdot \vdash \theta; \cdot \in [\Delta; \cdot]$ by definition 7.25

□

Lemma 7.28 (Well-typedness of modal substitutions in logical relations:)

If $\vdash \theta \in [\![\Delta]\!]$ *then* $\cdot; \cdot \vdash (\theta; \cdot) : (\Delta; \cdot)$

Proof: by induction over Δ:

Case: $\Delta = \cdot$:

$\vdash \theta \in [\![\cdot]\!]$ by assumption

$\Rightarrow \theta = \cdot$ by definition 7.23

$\Rightarrow \cdot; \cdot \vdash (\cdot; \cdot) : (\cdot; \cdot)$ by application of TSBase

Case: $\Delta = \Delta', x : A$:

$\vdash \theta \in [\![\Delta', x : A]\!]$ by assumption

$\Rightarrow \theta = \theta', M/x$ by definition 7.23

$\Rightarrow \cdot \vdash M \in [\![A]\!]$ by definition 7.23

$\Rightarrow \vdash \theta' \in [\![\Delta']\!]$ by definition 7.23

$\Rightarrow \cdot; \cdot \vdash (\theta'; \cdot) : (\Delta'; \cdot)$ by ind. hyp.

$\Rightarrow \cdot; \cdot \vdash M : A$ by definition 7.4

$\Rightarrow \cdot; \cdot \vdash (\theta', M/x; \cdot) : (\Delta', x : A; \cdot)$ by application of TSMod

$\Rightarrow \cdot; \cdot \vdash (\theta; \cdot) : (\Delta; \cdot)$ by definition

□

Lemma 7.29 (Well-typedness of arbitrary substitutions in logical relations:)

If $\Psi \vdash \varrho \in |\Gamma|$ *then* $\cdot; \Psi \vdash (\cdot; \varrho) : (\cdot; \Gamma)$

Proof: by induction over Γ:

Case: $\Gamma = \cdot$:

$\Psi \vdash \varrho \in |\cdot|$ by assumption

$\Rightarrow \varrho = \cdot$ by definition 7.24

$\Rightarrow \cdot; \Psi \vdash (\cdot; \cdot) : (\cdot; \cdot)$ by application of TSBase

Case: $\Gamma = \Gamma', x : A$:

$\Psi \vdash \varrho \in \|\Gamma', x : A\|$	by assumption
$\Rightarrow \quad \varrho = \varrho', M/x$	by definition 7.24
$\Rightarrow \quad \Psi \vdash M \in \|A\|$	by definition 7.24
$\Rightarrow \quad \Psi \vdash M \in [\![A]\!]$	by lemma 7.21
$\Rightarrow \quad \Psi \vdash \varrho' \in \|\Gamma'\|$	by definition 7.24
$\Rightarrow \quad \cdot; \Psi \vdash (\cdot; \varrho') \; : \; (\cdot; \Gamma')$	by ind. hyp.
$\Rightarrow \quad \cdot; \Psi \vdash M : A$	by definition 7.4
$\Rightarrow \quad \cdot; \Psi \vdash (\cdot; \varrho', M/x) \; : \; (\cdot; \Gamma', x : A)$	by application of TSReg
$\Rightarrow \quad \cdot; \Psi \vdash (\cdot; \varrho) \; : \; (\cdot; \Gamma)$	by definition

□

Lemma 7.30 (Combination of two substitutions)

If $\cdot; \cdot \vdash (\theta; \cdot) \; : \; (\Delta; \cdot)$ *and* $\cdot; \Psi \vdash (\cdot; \varrho) \; : \; (\cdot; \Gamma)$ *then* $\cdot; \Psi \vdash (\theta; \varrho) \; : \; (\Delta; \Gamma)$

Proof: by induction over $\mathcal{D} :: \cdot; \cdot \vdash (\theta; \cdot) \; : \; (\Delta; \cdot)$

Case: $\mathcal{D} = \dfrac{}{\cdot; \cdot \vdash (\cdot; \cdot) \; : \; (\cdot; \cdot)}\,\mathsf{TSBase}$:

$\cdot; \Psi \vdash (\cdot; \varrho) \; : \; (\cdot; \Gamma)$	by assumption

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \cdot; \cdot \vdash M : A\end{array} \qquad \begin{array}{c}\mathcal{D}_2 \\ \cdot; \cdot \vdash (\theta'; \cdot) \; : \; (\Delta'; \cdot)\end{array}}{\cdot; \cdot \vdash (\theta', M/x; \cdot) \; : \; (\Delta', x : A; \cdot)}\,\mathsf{TSMod}$:

$\cdot; \Psi \vdash (\theta'; \varrho) \; : \; (\Delta'; \Gamma)$	by ind. hyp.
$\cdot; \Psi \vdash (\theta; \varrho) \; : \; (\Delta; \Gamma)$	by application of TSMod

□

Lemma 7.31 (Well-typedness of substitutions in logical relations:)

If $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\cdot; \Psi \vdash (\theta; \varrho) \; : \; (\Delta; \Gamma)$

Proof: $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$	by assumption
$\Rightarrow \quad \vdash \theta \in [\![\Delta]\!]$	by definition 7.25
$\Rightarrow \quad \Psi \vdash \varrho \in \|\Gamma\|$	by definition 7.25
$\Rightarrow \quad \cdot; \cdot \vdash (\theta; \cdot) \; : \; (\Delta; \cdot)$	by lemma 7.28
$\Rightarrow \quad \cdot; \Psi \vdash (\cdot; \varrho) \; : \; (\cdot; \Gamma)$	by lemma 7.29
$\Rightarrow \quad \cdot; \Psi \vdash (\theta; \varrho) \; : \; (\Delta; \Gamma)$	by lemma 7.29

□

Lemma 7.32 (Properties of logical relation for modal contexts)

If $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\vdash \theta \in [\![\Delta]\!]$ *then* $\theta(x) = M$ *and* $\cdot \vdash M \in [\![A]\!]$

Proof: by induction over Δ_2

1. **Case:** $\Delta_2 = \cdot$

$\Delta = (\Delta_1, x : A) \cup \cdot$	by assumption
$\Delta = \Delta_1, x : A$	by definition CuBase
$\vdash \theta \in [\![\Delta_1, x : A]\!]$	by assumption
$\Rightarrow \quad \theta = \theta_1, M/x$	by definition 7.23
$\Rightarrow \quad \cdot \vdash M \in [\![A]\!]$	by definition 7.23
$\Rightarrow \quad (\theta_1, M/x)(x) = M$	by definition SbaInd
$\Rightarrow \quad \theta(x) = M$	by definition

Case: $\Delta_2 = \Delta_2', y : A', x \neq y$

$\Delta = (\Delta_1, x : A) \cup (\Delta_2', y : A')$	by assumption
$\Delta = (\Delta_1, x : A) \cup \Delta_2', y : A'$	by definition CuInd
$\vdash \theta \in [\![(\Delta_1, x : A) \cup \Delta_2', y : A']\!]$	by assumption
$\Rightarrow \quad \theta = \theta_1', M'/y$	by definition 7.23
$\Rightarrow \quad \cdot \vdash M' \in [\![A']\!]$	by definition 7.23
$\Rightarrow \quad \vdash \theta_1' \in [\![(\Delta_1, x : A) \cup \Delta_2']\!]$	by definition 7.23
$\Rightarrow \quad \theta_1'(x) = M$	by ind. hyp.
$\Rightarrow \quad \cdot \vdash M \in [\![A]\!]$	by ind. hyp.
$\Rightarrow \quad (\theta_1', M'/y)(x) = M$	by definition SbaInd
$\Rightarrow \quad \theta(x) = M$	by definition

□

Lemma 7.33 (Properties of logical relation for arbitrary contexts)

If $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Psi \vdash \varrho \in |\Gamma|$ *then* $\varrho(x) = M$ *and* $\Psi \vdash M \in |A|$

Proof: by induction over Γ_2

1. **Case:** $\Gamma_2 = \cdot$

$\Gamma = (\Gamma_1, x : A) \cup \cdot$	by assumption
$\Gamma = \Gamma_1, x : A$	by definition CuBase
$\Psi \vdash \varrho \in |\Gamma_1, x : A|$	by assumption
$\Rightarrow \quad \varrho = \varrho_1, M/x$	by definition 7.24
$\Rightarrow \quad \Psi \vdash M \in |A|$	by definition 7.24
$\Rightarrow \quad (\varrho_1, M/x)(x) = M$	by definition SbaInd
$\Rightarrow \quad \varrho(x) = M$	by definition

Case: $\Gamma_2 = \Gamma_2', y : A', x \neq y$

$\Gamma = (\Gamma_1, x : A) \cup (\Gamma_2', y : A')$	by assumption
$\Gamma = (\Gamma_1, x : A) \cup \Gamma_2', y : A'$	by definition CuInd
$\Psi \vdash \varrho \in \lvert(\Gamma_1, x : A) \cup \Gamma_2', y : A'\rvert$	by assumption
$\Rightarrow \quad \varrho = \varrho_1', M'/y$	by definition 7.24
$\Rightarrow \quad \Psi \vdash M' \in \lvert A'\rvert$	by definition 7.24
$\Rightarrow \quad \Psi \vdash \varrho_1' \in \lvert(\Gamma_1, x : A) \cup \Gamma_2'\rvert$	by definition 7.24
$\Rightarrow \quad \varrho_1'(x) = M$	by ind. hyp.
$\Rightarrow \quad \Psi \vdash M \in \lvert A\rvert$	by ind. hyp.
$\Rightarrow \quad (\varrho_1', M'/y)(x) = M$	by definition SbaInd
$\Rightarrow \quad \varrho(x) = M$	by definition

□

Lemma 7.34 (Properties of logical relation for contexts)

1. *If* $\Delta = (\Delta_1, x : A) \cup \Delta_2$ *and* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\theta(x) = M$ *and* $\cdot \vdash M \in [\![A]\!]$
2. *If* $\Gamma = (\Gamma_1, x : A) \cup \Gamma_2$ *and* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *then* $\varrho(x) = M$ *and* $\Psi \vdash M \in \lvert A\rvert$

Proof:

1. $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$	by assumption
$\Rightarrow \quad \vdash \theta \in [\![\Delta]\!]$	by definition 7.25
$\Rightarrow \quad \theta(x) = M$	by lemma 7.32
$\Rightarrow \quad \cdot \vdash M \in [\![A]\!]$	by lemma 7.32
2. $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$	by assumption
$\Rightarrow \quad \Psi \vdash \varrho \in \lvert\Gamma\rvert$	by definition 7.25
$\Rightarrow \quad \varrho(x) = M$	by lemma 7.33
$\Rightarrow \quad \Psi \vdash M \in \lvert A\rvert$	by lemma 7.33

□

Lemma 7.35 (Extending logical relations for contexts) *If* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ *and* $\Psi \vdash V \in \lvert A\rvert$ *then* $\Psi \vdash \theta; \varrho, V/x \in [\Delta; \Gamma, x : A]$

Proof:

$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$	by assumption
$\Rightarrow \quad \vdash \theta \in [\![\Delta]\!]$	by definition 7.25
$\Rightarrow \quad \Psi \vdash \varrho \in \lvert\Gamma\rvert$	by definition 7.25
$\Rightarrow \quad \Psi \vdash \varrho, V/x \in \lvert\Gamma, x : A\rvert$	by definition 7.24
$\Rightarrow \quad \Psi \vdash \theta; \varrho, V/x \in [\Delta; \Gamma, x : A]$	by definition 7.25

□

Lemma 7.36 (Identity substitution for arbitrary context)

For all Ψ *the following holds:* $\Psi \vdash \mathrm{id}_\Psi \in \lvert\Psi\rvert$

Proof: by induction over Ψ:

Case: $\Psi = \cdot$:

$\cdot \vdash \cdot \in |\cdot|$ — by definition 7.24

$\cdot \vdash \mathrm{id}_{\cdot} \in |\cdot|$ — by definition $\mathsf{IdEmpty}$

Case: $\Psi = \Psi', x : B$:

$\Psi' \vdash \mathrm{id}_{\Psi'} \in |\Psi'|$ — by ind. hyp.

$\Rightarrow \quad \Psi', x : B \vdash \mathrm{id}_{\Psi'} \in |\Psi'|$ — by lemma 7.26 (1)

$\Psi', x : B = \Psi', x : B \cup \cdot$ — by definition CuBase

$\Rightarrow \quad (\Psi', x : B)(x) = B$ — by definition 2.2

$\Rightarrow \quad \Psi', x : B \vdash x \downarrow B$ — by application of AtVar

$\Rightarrow \quad \Psi', x : B \vdash x \in |B|$ — by lemma 7.17 (2)

$\Rightarrow \quad \Psi', x : B \vdash \mathrm{id}_{\Psi'}, x/x \in |\Psi', x : B|$ — by definition 7.24

$\Rightarrow \quad \Psi', x : B \vdash \mathrm{id}_{\Psi', x:B} \in |\Psi', x : B|$ — by application of $\mathsf{IdNonEmpty}$

□

Lemma 7.37 (Identity substitution for context)

For all Ψ *the following holds:* $\Psi \vdash \cdot; \mathrm{id}_{\Psi} \in [\cdot; \Psi]$

Proof: $\Psi \vdash \mathrm{id}_{\Psi} \in |\Psi|$ — by lemma 7.36

$\vdash \cdot \in [\![\cdot]\!]$ — by definition 7.23

$\Rightarrow \quad \Psi \vdash \cdot; \mathrm{id}_{\Psi} \in [\cdot; \Psi]$ — by definition 7.25

□

Lemma 7.38 (Strengthening lemma)

Let $\hat{\Delta}; \Gamma \cup \Gamma^* \cup \hat{\Gamma} \vdash (\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}) \ : \ (\hat{\Delta}; \Gamma \cup \tilde{\Gamma} \cup \hat{\Gamma})$

1. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : A$ *then* $M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$
2. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$ *then* $\Xi = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi)$
3. *If* $\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Omega : \langle \omega\rangle(\Sigma')$ *then* $\Omega = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega)$

Proof: by induction over $\mathcal{D} :: \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : A$, $\mathcal{D} :: \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$ and $\mathcal{D} :: \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Omega : \langle \omega\rangle(\Sigma')$:

1. **Case:** $\mathcal{D} = \dfrac{\Gamma \cup \hat{\Gamma}(x) = A}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash x : A}\,\mathsf{TpVarReg}$:

   $\Gamma \cup \hat{\Gamma}(x) = A$ — by assumption

   $\Rightarrow \quad \Gamma_1, x : A \cup \Gamma_2$ or $\hat{\Gamma}_1, x : A \cup \hat{\Gamma}_2$ — by lemma 6.7

(a) $\Gamma = \Gamma_1, x : A \cup \Gamma_2$:
$\Rightarrow \quad x = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](x)$ by definition

(b) $\hat{\Gamma} = \hat{\Gamma_1}, x : A \cup \hat{\Gamma_2}$:
$\Rightarrow \quad x = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](x)$ by definition

Case: $\mathcal{D} = \dfrac{\hat{\Delta}(x) = A}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash x : A}$ TpVarMod:

$\hat{\Delta}(x) = A$ by assumption
$\Rightarrow \quad x = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](x)$ by definition

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash c : B}$ TpConst:

$c = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](c)$ by definition SBConst

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma}, x : A_1 \vdash M : A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \lambda x : A_1 .\, M : A_1 \to A_2}$ TpLam:

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}, x:A_1}](M)$ by ind. hyp. (1)
$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}, x/x](M)$ by definition IdNonEmpty
$\Rightarrow \quad \lambda x : A_1 .\, M = \lambda x : A_1 .\, [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}, x/x](M)$ by definition
$\Rightarrow \quad \lambda x : A_1 .\, M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\lambda x : A_1 .\, M)$ by definition SBLam

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_1 : A_2 \to A_1 \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_2 : A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_1\ M_2 : A_1}$ TpApp:

$M_1 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_1)$ by ind. hyp. (1)
$M_2 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_2)$ by ind. hyp. (1)
$\Rightarrow \quad M_1\ M_2 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_1\ M_2)$ by definition SBApp

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_1 : A_1 \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_2 : A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \langle M_1, M_2 \rangle : A_1 \times A_2}$ TpPair:

$M_1 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_1)$ by ind. hyp. (1)
$M_2 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_2)$ by ind. hyp. (1)
$\Rightarrow \quad \langle M_1, M_2 \rangle = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\langle M_1, M_2 \rangle)$ by definition SBPair

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : A_1 \times A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \mathrm{fst}\ M : A_1}$ TpFst:

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ by ind. hyp. (1)
$\Rightarrow \quad \mathrm{fst}\ M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\mathrm{fst}\ M)$ by definition SBFst

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : A_1 \times A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \mathrm{snd}\ M : A_2}$ TpSnd:

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ by ind. hyp. (1)
$\Rightarrow \quad \mathrm{snd}\ M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_{\Gamma} \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\mathrm{snd}\ M)$ by definition SBSnd

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \cdot \vdash M : A}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \mathrm{box}\ M : \Box A}$ TpBox:

$\hat{\Delta}; \Gamma \cup \Gamma^* \cup \hat{\Gamma} \vdash (\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}) \ : \ (\hat{\Delta}; \Gamma \cup \tilde{\Gamma} \cup \hat{\Gamma})$ by assumption
$\Rightarrow \ \hat{\Delta}; \cdot \vdash (\mathrm{id}_{\hat{\Delta}}; \cdot) \ : \ (\hat{\Delta}; \cdot)$ by lemma 6.19
$\Rightarrow \ \hat{\Delta}; \cdot \cup \cdot \cup \cdot \vdash (\mathrm{id}_{\hat{\Delta}}; \cdot \cup \cdot \cup \cdot) \ : \ (\hat{\Delta}; \cdot \cup \cdot \cup \cdot)$ by definition
$\Rightarrow \ M = [\mathrm{id}_{\hat{\Delta}}; \cdot \cup \cdot \cup \cdot](M)$ by ind. hyp. (1)
$\Rightarrow \ M = [\mathrm{id}_{\hat{\Delta}}; \cdot](M)$ by definition
$\Rightarrow \ \text{box } M = \text{box } [\mathrm{id}_{\hat{\Delta}}; \cdot](M)$ by definition
$\Rightarrow \ \text{box } M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\text{box } M)$ by definition SBBox

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M_1 : \Box A_1 \qquad \hat{\Delta}, x : A_1; \Gamma \cup \hat{\Gamma} \vdash M_2 : A_2}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \text{let box } x = M_1 \text{ in } M_2 : A_2}$ TpLet:

$M_1 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_1)$ by ind. hyp.
$M_2 = [\mathrm{id}_{\hat{\Delta}, x:A_1}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_2)$ by ind. hyp. (1)
$M_2 = [\mathrm{id}_{\hat{\Delta}}, x/x; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_2)$ by definition IdNonEmpty
$\Rightarrow \ (\text{let box } x = M_1 \text{ in } M_2) =$
$(\text{let box } x = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_1) \text{ in } [\mathrm{id}_{\hat{\Delta}}, x/x; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M_2))$ by definition
$\Rightarrow \ \text{let box } x = M_1 \text{ in } M_2 = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\text{let box } x = M_1 \text{ in } M_2)$ by definition SBLet

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : \Box B \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma')}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \text{case } \langle A \rangle \ M \ \langle \Xi \rangle : \mathcal{C}^* \ (B, A, B)}$ TpCase:

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ by ind. hyp. (1)
$\Xi = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi)$ by ind. hyp. (2)
$\Rightarrow \ \text{case } \langle A \rangle \ M \ \langle \Xi \rangle = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\text{case } \langle A \rangle \ M \ \langle \Xi \rangle)$ by definition SBCase

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : \Box B \qquad \vdash \omega : \alpha \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Omega : \langle \omega \rangle(\Sigma')}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \text{it } \langle \omega \rangle \ M \ \langle \Omega \rangle : \langle \omega \rangle(B)}$ TpIt:

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ by ind. hyp. (1)
$\Omega = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega)$ by ind. hyp. (3)
$\Rightarrow \ \text{it } \langle \omega \rangle \ M \ \langle \Omega \rangle = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\text{it } \langle \omega \rangle \ M \ \langle \Omega \rangle)$ by definition SBIt

2. **Case:** $\mathcal{D} = \dfrac{}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \cdot : \langle B \Rightarrow A \rangle(\cdot)}$ TmBase:

$\cdot = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\cdot)$ by definition SBXiEmpty

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma) \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : \mathcal{C} \ (B, A, B')}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash (\Xi \mid c \Rightarrow M) : \langle B \Rightarrow A \rangle(\Sigma, c : B')}$ TmInd:

$\Xi = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi)$ by ind. hyp. (2)
$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ by ind. hyp. (1)
$\Rightarrow \ (\Xi \mid c \Rightarrow M) =$
$([\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi) \mid c \Rightarrow [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M))$ by definition
$\Rightarrow \ (\Xi \mid c \Rightarrow M) = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Xi \mid c \Rightarrow M)$ by definition SBXi

3. **Case:** $\mathcal{D} = \dfrac{}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \cdot : \langle \omega \rangle(\cdot)}$ TrBase:

$\cdot = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\cdot)$ by definition SBOmegaEmpty

Case: $\mathcal{D} = \dfrac{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash \Omega : \langle\omega\rangle(\Sigma) \qquad \hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash M : \langle\omega\rangle(B')}{\hat{\Delta}; \Gamma \cup \hat{\Gamma} \vdash (\Omega \mid c \mapsto M) : \langle\omega\rangle(\Sigma, c : B')}$ TrInd:

$\Omega = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega)$ — by ind. hyp. (3)

$M = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M)$ — by ind. hyp. (1)

$\Rightarrow \quad (\Omega \mid c \mapsto M) =$
$([\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega) \mid c \mapsto [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](M))$ — by definition

$\Rightarrow \quad (\Omega \mid c \mapsto M) = [\mathrm{id}_{\hat{\Delta}}; \mathrm{id}_\Gamma \cup \varrho \cup \mathrm{id}_{\hat{\Gamma}}](\Omega \mid c \mapsto M)$ — by definition SBOmega

□

Lemma 7.42 (Preservation of Preconditions and Postconditions)

1. *$Pre{\downarrow_B}\ (\Psi, B')$ and $\Psi(x) = B'$ then $Post{\downarrow_B}\ (B')$*
2. *$Pre{\downarrow_B}\ (\Psi, B')$ and $\Sigma(c) = B'$ then $Post{\downarrow_B}\ (B')$*
3. *$Pre{\downarrow_B}\ (\Psi, B_2)$ implies $Pre{\downarrow_B}\ (\Psi, B_1 \rightarrow B_2)$*
4. *$Pre{\downarrow_B}\ (\Psi, B_2)$ and $Post{\downarrow_B}\ (B_1 \rightarrow B_2)$ implies $Pre{\Uparrow_B}\ (\Psi, B_1)$ and $Post{\downarrow_B}\ (B_2)$*
5. *$Pre{\Uparrow_B}\ (\Psi, B_1 \rightarrow B_2)$ implies $Pre{\Uparrow_B}\ (\Psi, x : B_1, B_2)$*
6. *For all pure types B: $Pre{\Uparrow_B}\ (\cdot, B)$*

Proof:

1. $\mathrm{Pre}{\downarrow_B}\ (\Psi, B')$ — by assumption (1)
   $\Rightarrow \quad \tau(B') \trianglelefteq_B \tau(B)$ — by definition 7.39
   $\Rightarrow \quad \Psi \blacktriangleleft_B \tau(B)$ — by definition 7.39
   $\Psi(x) = B'$ — by assumption (1)
   $\Rightarrow \quad \Psi = (\Psi_1, x : B') \cup \Psi_2$ — by definition 2.2
   $\Rightarrow \quad \tau(B') \trianglelefteq_B \tau(B)$ implies that forall $y \in \mathrm{Source}(B') : y \blacktriangleleft_B \tau(B)$ — by lemma 6.32
   $\Rightarrow \quad$ forall $y \in \mathrm{Source}(B') : y \blacktriangleleft_B \tau(B)$ — by definition
   $\Rightarrow \quad \mathrm{Post}{\downarrow_B}\ (B')$ — by definition
2. $\Sigma(c) = B'$ — by assumption (2)
   Let $y \in \mathrm{Source}(B')$ — by assumption
   $\Rightarrow \quad y \blacktriangleleft_B \tau(B')$ — by lemma 6.30
   $\mathrm{Pre}{\downarrow_B}\ (\Psi, B')$ — by assumption (2)
   $\Rightarrow \quad \tau(B') \trianglelefteq_B \tau(B)$ — by definition 7.39
   $\Rightarrow \quad y \blacktriangleleft_B \tau(B)$ — by lemma 6.30
   $\Rightarrow \quad$ forall $y \in \mathrm{Source}(B') : y \blacktriangleleft_B \tau(B)$ — by definition
   $\Rightarrow \quad \mathrm{Post}{\downarrow_B}\ (B')$ — by definition

3. $\text{Pre}{\downarrow_B}\ (\Psi, B_2)$ by assumption (3)
   $\Rightarrow\ \tau(B_2) \trianglelefteq_B \tau(B)$ by definition 7.39
   $\Rightarrow\ \tau(B_2) = \tau(B_1 \to B_2)$ by definition 4.15
   $\Rightarrow\ \tau(B_1 \to B_2) \trianglelefteq_B \tau(B)$ by definition
   $\Rightarrow\ \Psi \blacktriangleleft_B \tau(B)$ by definition 7.39
   $\Rightarrow\ \text{Pre}{\downarrow_B}\ (\Psi, B_1 \to B_2)$ by definition 7.39

4. $\text{Pre}{\downarrow_B}\ (\Psi, B_2)$ by assumption (4)
   $\Rightarrow\ \tau(B_2) \trianglelefteq_B \tau(B)$ by definition 7.39
   $\Rightarrow\ \Psi \blacktriangleleft_B \tau(B)$ by definition 7.39
   $\text{Post}{\downarrow_B}\ (B_1 \to B_2)$ by assumption (4)
   $\Rightarrow\ \text{forall } y \in \text{Source}(B_1 \to B_2) : y \blacktriangleleft_B \tau(B)$ by definition 7.41
   $\Rightarrow\ \text{Source}(B_1 \to B_2) = \text{Source}(B_1) \cup \{\tau(B_1)\} \cup \text{Source}(B_2)$ by definition 4.16
   $\Rightarrow\ \text{forall } y \in \text{Source}(B_1) : y \blacktriangleleft_B \tau(B)$ by definition
   $\Rightarrow\ \text{forall } y \in \text{Source}(B_2) : y \blacktriangleleft_B \tau(B)$ by definition
   $\Rightarrow\ \text{Post}{\downarrow_B}\ (B_2)$ by definition 7.41
   $\Rightarrow\ \text{forall } y \in \{\tau(B_1)\} : y \blacktriangleleft_B \tau(B)$ by definition
   $\Rightarrow\ \tau(B_1) \blacktriangleleft_B \tau(B)$ by definition
   $\Rightarrow\ \tau(B_1) \trianglelefteq_B \tau(B)$ by definition
   $\Rightarrow\ \text{Pre}{\downarrow_B}\ (\Psi, B_1)$ by definition 7.39
   $\text{forall } B' \in \text{PCT}(B_1) : \text{Source}(B') \subset \text{Source}(B_1)$ by lemma 6.33
   $\Rightarrow\ \text{forall } B' \in \text{PCT}(B_1) : \text{forall } y \in \text{Source}(B') : y \in \text{Source}(B_1)$ by definition
   $\Rightarrow\ \text{forall } B' \in \text{PCT}(B_1) : \text{forall } y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$ by definition
   $\Rightarrow\ \text{forall } B' \in \text{PCT}(B_1) : \text{if } \tau(B'') \trianglelefteq_B \tau(B) \text{then forall } y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$
   by definition
   $\Rightarrow\ \text{Pre}{\Uparrow_B}\ (\Psi, B_1)$ by definition 7.40

5. $\text{Pre}{\Uparrow_B}\ (\Psi, B_1 \to B_2)$ by assumption (5)
   $\Rightarrow\ \text{Pre}{\downarrow_B}\ (\Psi, B_1 \to B_2)$ by definition 7.40
   $\Rightarrow\ \text{forall } B' \in \text{PCT}(B_1 \to B_2) : \text{if } \tau(B') \trianglelefteq_B \tau(B) \text{ then forall } y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$
   by definition 7.40
   $\Rightarrow\ \tau(B_1 \to B_2) \trianglelefteq_B \tau(B)$ by definition 7.39
   $\Rightarrow\ \Psi \blacktriangleleft_B \tau(B)$ by definition 7.39
   $\Rightarrow\ \tau(B_1 \to B_2) = \tau(B_2)$ by definition 4.15
   $\Rightarrow\ \tau(B_2) \trianglelefteq_B \tau(B)$ by definition
   $\Rightarrow\ \text{PCT}(B_1 \to B_2) = \{B_1\} \cup \text{PCT}(B_2)$ by definition 4.22

$\Rightarrow$ if $\tau(B_1) \trianglelefteq_B \tau(B)$ then forall $y \in \text{Source}(B_1) : y \blacktriangleleft_B \tau(B)$ by definition

$\Rightarrow$ $\Psi, x : B_1 \blacktriangleleft_B \tau(B)$ by definition 7.39

$\Rightarrow$ $\text{Pre}{\downarrow_B}\ (\Psi, x : B_1, B_2)$ by definition 7.39

$\Rightarrow$ forall $B' \in \text{PCT}(B_2)$: if $\tau(B') \trianglelefteq_B \tau(B)$ then forall $y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$

by definition 7.40

$\Rightarrow$ $\text{Pre}{\Uparrow_B}\ (\Psi, x : B_1, B_2)$ by definition 7.40

6. Let B be a pure Type by assumption (6)

$\tau(B) = \tau(B)$ by definition

$\Rightarrow$ $\tau(B) \trianglelefteq_B \tau(B)$ by definition 6.29

$\cdot \blacktriangleleft_B \tau(B)$ by definition 6.31

$\Rightarrow$ $\text{Pre}{\downarrow_B}\ (\cdot, B)$ by definition 7.39

Let $B' \in \text{PCT}(B)$ by assumption

Assume $\tau(B') \trianglelefteq_B \tau(B)$ by assumption

Let $y \in \text{Source}(B')$ by assumption

$\Rightarrow$ $y <_{B'} \tau(B')$ by definition 4.19

$\Rightarrow$ $y \lhd_B \tau(B')$ by lemma 6.34

$\Rightarrow$ $y \blacktriangleleft_B \tau(B')$ by definition 4.24

$\Rightarrow$ $y \blacktriangleleft_B \tau(B)$ by lemma 6.30 (2)

$\Rightarrow$ forall $B' \in \text{PCT}(B)$: if $\tau(B') \trianglelefteq_B \tau(B)$ then forall $y \in \text{Source}(B') : y \blacktriangleleft_B \tau(B)$

by logic

$\Rightarrow$ $\text{Pre}{\Uparrow_B}\ (\cdot, B)$ by definition 7.40

$\square$

Lemma 7.43 (Auxiliary lemma for iterator)
If $\Psi \cup \bar{\Psi} \vdash \cdot; \text{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ *and* $\Sigma' = \mathcal{S}^*(\Sigma; \mathcal{I}(\Sigma; B))$

1. *If* $\hat{\Psi} \vdash V \downarrow B'$ *and Pre*$\downarrow_B$ $(\hat{\Psi}, B')$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \text{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(V)) \in [\![\langle\omega\rangle(B')]\!]$ *and Post*$\downarrow_B$ (B')

2. *If* $\hat{\Psi} \vdash V \Uparrow B'$ *and Pre*$\Uparrow_B$ $(\hat{\Psi}, B')$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \text{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(V)) \in [\![\langle\omega\rangle(B')]\!]$

Proof: by induction over $\mathcal{D} :: \hat{\Psi} \vdash V \Uparrow B'$ and $\mathcal{E} :: \hat{\Psi} \vdash V \downarrow B'$

1. **Case:** $\mathcal{E} = \dfrac{\hat{\Psi}(x) = B'}{\hat{\Psi} \vdash x \downarrow B'}\,\mathsf{AtVar}$

$\hat{\Psi}(x) = B'$ by assumption

$\Rightarrow$ $\hat{\Psi} = \hat{\Psi}_1, x : B' \cup \hat{\Psi}_2$ by definition 2.2

$\Rightarrow$ $\tilde{\Psi} = \tilde{\Psi}_1, u : \langle\omega\rangle(B') \cup \tilde{\Psi}'_2$ by lemma 7.14

$\Rightarrow\; u = \langle\omega;\Omega\rangle(x)$ — by lemma 7.14

$\Rightarrow\; (\mathrm{id}_\Psi \cup \varrho)(u) = M$ — by lemma 7.34

$\Rightarrow\; \Psi \cup \bar{\Psi} \vdash M \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by lemma 7.34

$\Rightarrow\; [\cdot; \mathrm{id}_\Psi \cup \varrho](u) = M$ — by definition SBVar

$\Rightarrow\; \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](u) \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by definition

$\Rightarrow\; \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(x)) \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by definition

$\mathrm{Pre}{\downarrow_B}\,(\hat{\Psi}, B')$ — by assumption

$\Rightarrow\; \mathrm{Post}{\downarrow_B}\,(B')$ — by lemma 7.42 (1)

Case: $\mathcal{E} = \dfrac{\Sigma(c) = B'}{\hat{\Psi} \vdash c \downarrow B'}\ \mathsf{AtConst}$

$\Sigma(c) = B'$ — by assumption

$\mathrm{Pre}{\downarrow_B}\,(\hat{\Psi}, B')$ — by assumption

$\Rightarrow\; \mathrm{Post}{\downarrow_B}\,(B')$ — by lemma 7.42 (2)

Case: $\tau(B') \in \mathcal{I}(\Sigma; B)$:

$\Sigma'(c) = B'$ — by assumption

$\Sigma' = \Sigma_1', c : B' \cup \Sigma_2'$ — by definition 4.17

$\Rightarrow\; \Psi \vdash M \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by lemma 7.12

$\Rightarrow\; M = \langle\omega;\Omega\rangle(c)$ — by lemma 7.12

$\Rightarrow\; \cdot; \Psi \vdash M : \langle\omega\rangle(B')$ — by definition 7.4

$\Rightarrow\; \cdot; \Psi \cup \cdot \vdash M : \langle\omega\rangle(B')$ — by definition CuBase

$\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$ — by assumption

$\Rightarrow\; \cdot; \Psi \cup \bar{\Psi} \vdash (\cdot; \mathrm{id}_\Psi \cup \varrho) \;:\; (\cdot; \Psi \cup \tilde{\Psi})$ — by lemma 7.31

$\Rightarrow\; \cdot; \Psi \cup \bar{\Psi} \cup \cdot \vdash (\cdot; \mathrm{id}_\Psi \cup \varrho \cup \cdot) \;:\; (\cdot; \Psi \cup \tilde{\Psi} \cup \cdot)$ — by definition CuBase

$\Rightarrow\; M = [\cdot; \mathrm{id}_\Psi \cup \varrho \cup \cdot](M)$ — by lemma 7.38

$\Rightarrow\; M = [\cdot; \mathrm{id}_\Psi \cup \varrho](M)$ — by definition CuBase

$\Rightarrow\; M = [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(c))$ — by definition

$\Rightarrow\; \bar{\Psi} \geq \cdot$ — by lemma 6.2

$\Rightarrow\; \Psi \cup \bar{\Psi} \geq \Psi \cup \cdot$ — by lemma 6.5

$\Rightarrow\; \Psi \cup \bar{\Psi} \geq \Psi$ — by definition CuBase

$\Rightarrow\; \Psi \cup \bar{\Psi} \vdash M \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by lemma 7.9 (1)

$\Rightarrow\; \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(c)) \in \llbracket\langle\omega\rangle(B')\rrbracket$ — by definition

Case: $\tau(B') \notin \mathcal{I}(\Sigma; B)$:

$\Rightarrow\; c : B' \notin \Sigma'$ — by definition

$\Rightarrow\; \Sigma'(c)$ is undefined — by definition

$\mathrm{Pre}{\downarrow_B}\,(\hat{\Psi}, B')$ — by assumption

$\Rightarrow\; \tau(B') \;\underline{\blacktriangleleft}_B\; \tau(B)$ — by definition 7.39

$\Rightarrow\; \tau(B) \;\not\blacktriangleleft_B\; \tau(B')$ — by lemma 6.37

$\Rightarrow\; \mathrm{Source}(B') \cap \mathcal{I}(\Sigma; B) = \emptyset$ — by lemma 6.35

$\Rightarrow\; \langle\omega\rangle(B') = B'$ — by lemma 6.36

$\Psi + \tilde{\Psi} \vdash \Omega \in \llbracket\langle\omega\rangle(\Sigma'; \hat{\Psi})\rrbracket$ — by assumption

$\Rightarrow\; \Omega(c)$ is undefined — by lemma 7.13

$\Rightarrow\; \langle\omega;\Omega\rangle(c) = c$ — by definition ElConst

$\Rightarrow\; \Sigma(c) = B'$ — by assumption

$\Rightarrow \quad \Psi \vdash c \downarrow B'$ by application of AtVar
$\Rightarrow \quad \Psi \vdash c \in |B'|$ by lemma 7.17
$\Rightarrow \quad \Psi \vdash c \in [\![B']\!]$ by lemma 7.21
$\Rightarrow \quad \Psi \vdash c \in [\![\langle\omega\rangle(B')]\!]$ by definition 4.26
$[\cdot; \mathrm{id}_\Psi \cup \varrho](c) = c$ by definition SBConst
$\Rightarrow \quad \Psi \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](c) \in [\![\langle\omega\rangle(B')]\!]$ by lemma 7.21
$\Rightarrow \quad \Psi \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(c)) \in [\![\langle\omega\rangle(B')]\!]$ by definition

Case: $\mathcal{E} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \hat{\Psi} \vdash V_1 \downarrow B'_2 \to B'_1 & \hat{\Psi} \vdash V_2 \Uparrow B'_2 \end{array}}{\hat{\Psi} \vdash V_1\ V_2 \downarrow B'_1}\ \mathsf{AtApp}$

$\mathrm{Pre}{\downarrow_B}\ (\hat{\Psi}, B'_1)$ by assumption
$\Rightarrow \quad \mathrm{Pre}{\downarrow_B}\ (\hat{\Psi}, B'_2 \to B'_1)$ by lemma 7.42 (3)
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_1)) \in [\![\langle\omega\rangle(B'_2 \to B'_1)]\!]$ by ind. hyp. (1) on $\mathcal{D}_1$
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_1)) \in [\![\langle\omega\rangle(B'_2) \to \langle\omega\rangle(B'_1)]\!]$ by definition 4.26
$\Rightarrow \quad \mathrm{Post}{\downarrow_B}\ (B'_2 \to B'_1)$ by ind. hyp. (1) on $\mathcal{D}_1$
$\Rightarrow \quad \mathrm{Pre}{\Uparrow_B}\ (\hat{\Psi}, B'_2)$ by lemma 7.42 (4)
$\Rightarrow \quad \mathrm{Post}{\downarrow_B}\ (B'_1)$ by lemma 7.42 (4)
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_2)) \in [\![\langle\omega\rangle(B'_2)]\!]$ by ind. hyp. (2) on $\mathcal{D}_2$
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_1))\ [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_2)) \in [\![\langle\omega\rangle(B'_1)]\!]$ by lemma 7.22
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_1)\ \langle\omega;\Omega\rangle(V_2)) \in [\![\langle\omega\rangle(B'_1)]\!]$ by application of SbApp
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V_1\ V_2)) \in [\![\langle\omega\rangle(B'_1)]\!]$ by application of ElApp

2. **Case:** $\mathcal{D} = \dfrac{\begin{array}{c} \mathcal{D}_1 \\ \hat{\Psi} \vdash V \downarrow a \end{array}}{\hat{\Psi} \vdash V \Uparrow a}\ \mathsf{CanAt}$

$\mathrm{Pre}{\Uparrow_B}\ (\hat{\Psi}, a)$ by assumption
$\Rightarrow \quad \mathrm{Pre}{\downarrow_B}\ (\hat{\Psi}, a)$ by definition 7.40
$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega;\Omega\rangle(V)) \in [\![\langle\omega\rangle(a)]\!]$ by ind. hyp. (1) on $\mathcal{D}_1$

Case: $\mathcal{D} = \dfrac{\hat{\Psi}, x : B'_1 \vdash V \Uparrow B'_2}{\hat{\Psi} \vdash \lambda x{:}B'_1.\, V \Uparrow B'_1 \to B'_2}\ \mathsf{CanLam}$

$(\tilde{\Psi}, u : \langle\omega\rangle(B'_1))(u) = \langle\omega\rangle(B'_1)$ by definition
$\Rightarrow \quad \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \vdash u \downarrow \langle\omega\rangle(B'_1)$ by application of AtVar
$\Rightarrow \quad \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \vdash u \in |\langle\omega\rangle(B'_1)|$ by lemma 7.17
$\Rightarrow \quad \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \vdash u \in [\![\langle\omega\rangle(B'_1)]\!]$ by lemma 7.21
$\Rightarrow \quad \Psi + \tilde{\Psi} \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ by assumption
$\tilde{\Psi} \geq \tilde{\Psi}$ by definition CeBase
$\Rightarrow \quad \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \geq \tilde{\Psi}$ by definition CeInd
$\Rightarrow \quad \Psi + \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \vdash \Omega \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi})]\!]$ by lemma 7.10
$\Rightarrow \quad \Psi + \tilde{\Psi}, u : \langle\omega\rangle(B'_1) \vdash \Omega \mid x \mapsto u \in [\![\langle\omega\rangle(\Sigma'; \hat{\Psi}, x : B'_1)]\!]$ by definition 7.5
Let $\bar{\Psi}'' \geq \Psi \cup \bar{\Psi}$

$\Rightarrow\quad \bar{\Psi}'' = \Psi \cup \bar{\Psi}'$ by lemma 6.4

$\Rightarrow\quad \bar{\Psi}' \geq \bar{\Psi}$ by lemma 6.4

Let $\Psi \cup \bar{\Psi}' \vdash V' \in |\langle\omega\rangle(B_1')|$

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$ by assumption

$\Rightarrow\quad \Psi \cup \bar{\Psi}' \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$ by lemma 7.26

$\Rightarrow\quad \Psi \cup \bar{\Psi}' \vdash \cdot; (\mathrm{id}_\Psi \cup \varrho), V'/u \in [\cdot; (\Psi \cup \tilde{\Psi}), u : \langle\omega\rangle(B_1')]$ by lemma 7.35

$\Rightarrow\quad \Psi \cup \bar{\Psi}' \vdash \cdot; \mathrm{id}_\Psi \cup \varrho, V'/u \in [\cdot; \Psi \cup \tilde{\Psi}, u : \langle\omega\rangle(B_1')]$ by definition CeInd

$\mathrm{Pre}\Uparrow_B (\hat{\Psi}, B_1' \to B_2')$ by assumption

$\Rightarrow\quad \mathrm{Pre}\Uparrow_B (\hat{\Psi}, x : B_1', B_2')$ by lemma 7.42 (5)

$\Rightarrow\quad \Psi \cup \bar{\Psi}' \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho, V'/u](\langle\omega; \Omega \mid x \mapsto u\rangle(V)) \in [\![\langle\omega\rangle(B_2')]\!]$ by ind. hyp. (2)

$\Rightarrow\quad \Psi \cup \bar{\Psi}' \vdash [V'/u]([\cdot; \mathrm{id}_\Psi \cup \varrho, u/u](\langle\omega; \Omega \mid x \mapsto u\rangle(V))) \in [\![\langle\omega\rangle(B_2')]\!]$

by lemma 6.17 (1)

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash \lambda u{:}\langle\omega\rangle(B_1').\, [\cdot; \mathrm{id}_\Psi \cup \varrho, u/u](\langle\omega; \Omega \mid x \mapsto u\rangle(V)) \in |\langle\omega\rangle(B_1') \to \langle\omega\rangle(B_2')|$

by definition 7.4

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash \lambda u{:}\langle\omega\rangle(B_1').\, [\cdot; (\mathrm{id}_\Psi \cup \varrho), u/u](\langle\omega; \Omega \mid x \mapsto u\rangle(V)) \in |\langle\omega\rangle(B_1') \to \langle\omega\rangle(B_2')|$

by definition CuInd

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\lambda u{:}\langle\omega\rangle(B_1').\, \langle\omega; \Omega \mid x \mapsto u\rangle(V)) \in |\langle\omega\rangle(B_1') \to \langle\omega\rangle(B_2')|$

by definition SbLam

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(\lambda x{:}B_1'.\, V)) \in |\langle\omega\rangle(B_1') \to \langle\omega\rangle(B_2')|$ by definition ElLam

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(\lambda x{:}B_1'.\, V)) \in |\langle\omega\rangle(B_1' \to B_2')|$ by definition 4.26

$\Rightarrow\quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\langle\omega; \Omega\rangle(\lambda x{:}B_1'.\, V)) \in [\![\langle\omega\rangle(B_1' \to B_2')]\!]$ by lemma 7.21

□

Lemma 7.44 (Every canonical element is member of the logical relation)

1. *If* $\Psi \vdash V \downarrow B$ *and* $\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$ *then* $\Psi' \vdash [\cdot; \varrho](V) \in [\![B]\!]$

2. *If* $\Psi \vdash V \Uparrow B$ *and* $\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$ *then* $\Psi' \vdash [\cdot; \varrho](V) \in [\![B]\!]$

Proof: by induction over $\mathcal{D} :: \Psi \vdash V \downarrow B$ and $\mathcal{E} :: \Psi \vdash V \Uparrow B$:

1. **Case:** $\mathcal{D} = \dfrac{\Psi(x) = B}{\Psi \vdash x \downarrow B}\,\mathsf{AtVar}$

$\Psi(x) = B$ by assumption

$\Rightarrow\quad \Psi = \Psi_1, x : B \cup \Psi_2$ by definition 2.2

$\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$ by assumption

$\Rightarrow\quad \varrho(x) = M$ by lemma 7.34 (2)

$\Rightarrow\quad [\cdot; \varrho](x) = M$ by definition SBVar

$\Rightarrow\quad \Psi' \vdash M \in [\![B]\!]$ by lemma 7.34 (2)

$\Rightarrow\quad \Psi' \vdash [\cdot; \varrho](x) \in [\![B]\!]$ by definition

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\Psi \vdash c \downarrow B}\,\mathsf{AtConst}$

$\Sigma(c) = B$	by assumption
$\Rightarrow\ \ \Psi' \vdash c \downarrow B$	by application of $\mathsf{AtConst}$
$\Rightarrow\ \ \Psi' \vdash c \in \lvert B \rvert$	by lemma 7.17 (2)
$\Rightarrow\ \ \Psi' \vdash c \in [\![B]\!]$	by definition 7.4
$\Rightarrow\ \ \Psi' \vdash [\cdot; \varrho](c) \in [\![B]\!]$	by application of SBConst

Case: $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Psi \vdash V_1 \downarrow B_2 \to B_1\end{matrix} \qquad \begin{matrix}\mathcal{E}_1 \\ \Psi \vdash V_2 \Uparrow B_2\end{matrix}}{\Psi \vdash V_1\ V_2 \downarrow B_1}\,\mathsf{AtApp}$

$\Psi' \vdash [\cdot; \varrho](V_1) \in [\![B_2 \to B_1]\!]$	by ind. hyp. (1) on $\mathcal{D}_1$
$\Psi' \vdash [\cdot; \varrho](V_2) \in [\![B_2]\!]$	by ind. hyp. (2) on $\mathcal{E}_1$
$\Rightarrow\ \ \Psi' \vdash [\cdot; \varrho](V_1)\ [\cdot; \varrho](V_2) \in [\![B_1]\!]$	by lemma 7.22
$\Rightarrow\ \ \Psi' \vdash [\cdot; \varrho](V_1\ V_2) \in [\![B_1]\!]$	by definition SBApp

2. **Case:** $\mathcal{E} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Psi \vdash V \downarrow a\end{matrix}}{\Psi \vdash V \Uparrow a}\,\mathsf{CanAt}$

$\Psi' \vdash [\cdot; \varrho](V) \in [\![a]\!]$	by ind. hyp. (1) on $\mathcal{D}_1$

Case: $\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ \Psi, x : B_1 \vdash V \Uparrow B_2\end{matrix}}{\Psi \vdash \lambda x\!:\!B_1.\, V \Uparrow B_1 \to B_2}\,\mathsf{CanLam}$

Let $\Psi'' \geq \Psi'$	by assumption
Let $\Psi'' \vdash W \in \lvert B_1 \rvert$	by assumption
$\Psi' \vdash \cdot; \varrho \in [\cdot; \Psi]$	by assumption
$\Rightarrow\ \ \Psi'' \vdash \cdot; \varrho \in [\cdot; \Psi]$	by lemma 7.26 (1)
$\Rightarrow\ \ \Psi'' \vdash \cdot; \varrho, W/x \in [\cdot; \Psi, x : B_1]$	by lemma 7.35
$\Rightarrow\ \ \Psi'' \vdash [\cdot; \varrho, W/x](V) \in [\![B_2]\!]$	by ind. hyp. (2)
$\Rightarrow\ \ \Psi'' \vdash [W/x]([\cdot; \varrho, x/x](V)) \in [\![B_2]\!]$	by lemma 6.17 (1)
$\Rightarrow\ \ \Psi' \vdash \lambda x\!:\!B_1.\, [\cdot; \varrho, x/x](V) \in \lvert B_1 \to B_2 \rvert$	by definition 7.4
$\Rightarrow\ \ \Psi' \vdash [\cdot; \varrho](\lambda x\!:\!B_1.\, V) \in \lvert B_1 \to B_2 \rvert$	by definition SBLam
$\Rightarrow\ \ \Psi' \vdash [\cdot; \varrho](\lambda x\!:\!B_1.\, V) \in [\![B_1 \to B_2]\!]$	by lemma 7.21

□

Lemma 7.45 (Properties of transformation types:)

If $\Psi \vdash V \Uparrow B$ *then* $\cdot \vdash \lambda\{\Psi\}.\, V \in [\![\Pi\{\Psi\}.\, B]\!]$

Proof: by induction over Ψ:

Case: $\Psi = \cdot$:

$\Rightarrow\ \cdot \vdash \cdot;\cdot \in [\cdot;\cdot]$	by definition 7.23, 7.24, 7.25
$\Rightarrow\ \cdot \vdash [\cdot;\cdot](V) \in [\![B]\!]$	by lemma 7.44
$\Rightarrow\ \cdot;\cdot \vdash V : B$	by lemma 6.25
$\Rightarrow\ [\cdot;\cdot](V) = V$	by lemma 6.23
$\Rightarrow\ \cdot \vdash V \in [\![B]\!]$	by definition
$\Rightarrow\ \cdot \vdash \lambda\{\cdot\}.\, V \in [\![B]\!]$	by definition 5.8
$\Rightarrow\ \cdot \vdash \lambda\{\cdot\}.\, V \in [\![\Pi\{\cdot\}.\, B]\!]$	by definition 5.9

Case: $\Psi = \Psi', x : B'$:

$\Rightarrow\ \Psi', x : B' \vdash V \Uparrow B$	by assumption
$\Rightarrow\ \Psi' \vdash \lambda x{:}B'.\, V \Uparrow B' \to B$	by application of CanLam
$\Rightarrow\ \cdot \vdash \lambda\{\Psi'\}.\, \lambda x{:}B'.\, V \in [\![\Pi\{\Psi'\}.\,(B' \to B)]\!]$	by ind. hyp.
$\Rightarrow\ \cdot \vdash \lambda\{\Psi', x : B'\}.\, V \in [\![\Pi\{\Psi'\}.\,(B' \to B)]\!]$	by definition 5.8
$\Rightarrow\ \cdot \vdash \lambda\{\Psi', x : B'\}.\, V \in [\![\Pi\{\Psi', x : B'\}.\, B]\!]$	by definition 5.9

□

Lemma 7.46 (Auxiliary lemma for case)

If $\Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$, $\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ *and* $\Sigma' = \mathcal{S}(\Sigma; \tau(B))$

1. *If* $\hat{\Psi} \vdash V \downarrow B'$ *and* $\Pi\{\hat{\Psi}\}.\, \tau(B) = B$ *and* $\tau(B') = \tau(B)$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}\ (B, A, B')]\!]$
2. *If* $\hat{\Psi} \vdash V \Uparrow B'$ *and* $\Pi\{\hat{\Psi}\}.\, B' = B$ *then* $\Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V)) \in [\![\mathcal{C}^*\ (B, A, B')]\!]$

Proof: by induction over $\mathcal{D} :: \Psi \vdash V \downarrow B$ and $\mathcal{E} :: \Psi \vdash V \Uparrow B$:

1. **Case:** $\mathcal{D} = \dfrac{\Psi'(x) = B'}{\hat{\Psi} \vdash x \downarrow B'}\,\mathsf{AtVar}$

$\hat{\Psi}(x) = B'$	by assumption
$\Rightarrow\ \hat{\Psi} = \hat{\Psi}_1, x : B' \cup \hat{\Psi}_2$	by definition 2.2
$\Rightarrow\ \tilde{\Psi} = \tilde{\Psi}_1, u : \mathcal{C}\ (B, A, B') \cup \tilde{\Psi}'_2$	by lemma 7.16
$\Rightarrow\ u = \{B \Rightarrow A; \Xi; \hat{\Psi}\}(x)$	by lemma 7.16
$\Rightarrow\ (\mathrm{id}_\Psi \cup \varrho)(u) = M$	by lemma 7.34
$\Rightarrow\ \Psi \cup \bar{\Psi} \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!]$	by lemma 7.34
$\Rightarrow\ [\cdot; \mathrm{id}_\Psi \cup \varrho](u) = M$	by definition SBVar
$\Rightarrow\ \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](u) \in [\![\mathcal{C}\ (B, A, B')]\!]$	by definition
$\Rightarrow\ \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(x)) \in [\![\mathcal{C}\ (B, A, B')]\!]$	by definition

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B'}{\Psi' \vdash c \downarrow B'}\,\mathsf{AtConst}$

$$
\begin{array}{llr}
 & \tau(B') = \tau(B) & \text{by assumption}\\
\Rightarrow & c : B' \in \mathcal{S}(\Sigma; \tau(B)) & \text{by definition 4.17}\\
\Rightarrow & c : B' \in \Sigma' & \text{by definition}\\
\Rightarrow & \Sigma' = \Sigma'_1, c : B' \cup \Sigma'_2 & \text{by definition 2.2}\\
\Rightarrow & \Psi \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!] & \text{by lemma 7.15}\\
\Rightarrow & M = \{B \Rightarrow A; \Omega; \hat{\Psi}\}(c) & \text{by lemma 7.15}\\
\Rightarrow & \cdot; \Psi \vdash M : \mathcal{C}\ (B, A, B') & \text{by definition 7.4}\\
\Rightarrow & \cdot; \Psi \cup \cdot \vdash M : \mathcal{C}\ (B, A, B') & \text{by definition } \mathsf{CuBase}\\
 & \Psi \cup \bar{\Psi} \vdash \cdot; \mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}] & \text{by assumption}\\
\Rightarrow & \cdot; \Psi \cup \bar{\Psi} \vdash (\cdot; \mathrm{id}_\Psi \cup \varrho)\ :\ (\cdot; \Psi \cup \tilde{\Psi}) & \text{by lemma 7.31}\\
\Rightarrow & \cdot; \Psi \cup \bar{\Psi} \cup \cdot \vdash (\cdot; \mathrm{id}_\Psi \cup \varrho \cup \cdot)\ :\ (\cdot; \Psi \cup \tilde{\Psi} \cup \cdot) & \text{by definition } \mathsf{CuBase}\\
\Rightarrow & M = [\cdot; \mathrm{id}_\Psi \cup \varrho \cup \cdot](M) & \text{by lemma 7.38}\\
\Rightarrow & M = [\cdot; \mathrm{id}_\Psi \cup \varrho](M) & \text{by definition } \mathsf{CuBase}\\
\Rightarrow & M = [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Omega; \hat{\Psi}\}(c)) & \text{by definition}\\
\Rightarrow & \bar{\Psi} \geq \cdot & \text{by lemma 6.2}\\
\Rightarrow & \Psi \cup \bar{\Psi} \geq \Psi \cup \cdot & \text{by lemma 6.5}\\
\Rightarrow & \Psi \cup \bar{\Psi} \geq \Psi & \text{by definition } \mathsf{CuBase}\\
\Rightarrow & \Psi \cup \bar{\Psi} \vdash M \in [\![\mathcal{C}\ (B, A, B')]\!] & \text{by lemma 7.9 (1)}\\
\Rightarrow & \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Omega; \hat{\Psi}\}(c)) \in [\![\mathcal{C}\ (B, A, B')]\!] & \text{by definition}
\end{array}
$$

Case: $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{E}_1\\ \hat{\Psi} \vdash V_1 \downarrow B'_2 \to B'_1 & \hat{\Psi} \vdash V_2 \Uparrow B'_2\end{array}}{\hat{\Psi} \vdash V_1\ V_2 \downarrow B'_1}\,\mathsf{AtApp}$

$$
\begin{array}{llr}
 & \tau(B'_1) = \tau(B) & \text{by assumption}\\
\Rightarrow & \tau(B'_2 \to B'_1) = \tau(B) & \text{by definition 4.15}\\
 & \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1)) \in [\![\mathcal{C}\ (B, A, (B'_2 \to B'_1))]\!] & \\
 & & \text{by ind. hyp. (1) on } \mathcal{D}_1\\
 & \Pi\{\hat{\Psi}\}.\, \tau(B) = B & \text{by assumption}\\
\Rightarrow & \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1)) \in [\![\mathcal{C}\ (\Pi\{\hat{\Psi}\}.\, \tau(B), A, (B'_2 \to B'_1))]\!] & \\
 & & \text{by definition}\\
\Rightarrow & \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1)) \in [\![\Box\Pi\{\hat{\Psi}\}.\, B'_2 \to \mathcal{C}\ (\Pi\{\hat{\Psi}\}.\, \tau(B), A, B'_1)]\!] & \\
 & & \text{by definition}\\
\Rightarrow & \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1)) \in [\![\Box\Pi\{\hat{\Psi}\}.\, B'_2 \to \mathcal{C}\ (B, A, B'_1)]\!] & \\
 & & \text{by definition}\\
 & \hat{\Psi} \vdash V_2 \Uparrow B'_2 & \text{by assumption}\\
\Rightarrow & \cdot \vdash \lambda\{\hat{\Psi}\}.\, V_2 \in [\![\Pi\{\hat{\Psi}\}.\, B'_2]\!] & \text{by lemma 7.45}\\
\Rightarrow & \cdot; \cdot \vdash \lambda\{\hat{\Psi}\}.\, V_2 : \Pi\{\hat{\Psi}\}.\, B'_2 & \text{by definition 7.4}\\
\Rightarrow & \lambda\{\hat{\Psi}\}.\, V_2 = [\cdot; \cdot](\lambda\{\hat{\Psi}\}.\, V_2) & \text{by lemma 6.23}\\
\Rightarrow & \cdot \vdash [\cdot; \cdot](\lambda\{\hat{\Psi}\}.\, V_2) \in [\![\Pi\{\hat{\Psi}\}.\, B'_2]\!] & \text{by definition}
\end{array}
$$

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash \text{box } [\cdot;\cdot](\lambda\{\hat{\Psi}\}.\, V_2) \in |\Box(\Pi\{\hat{\Psi}\}.\, B_2')|$ by definition 7.4

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash \text{box } [\cdot;\cdot](\lambda\{\hat{\Psi}\}.\, V_2) \in [\![\Box(\Pi\{\hat{\Psi}\}.\, B_2')]\!]$ by lemma 7.21

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\text{box } \lambda\{\hat{\Psi}\}.\, V_2) \in [\![\Box(\Pi\{\hat{\Psi}\}.\, B_2')]\!]$ by definition SBBox

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1))\ [\cdot;\mathrm{id}_\Psi \cup \varrho](\text{box } \lambda\{\hat{\Psi}\}.\, V_2) \in [\![\mathcal{C}\ (B, A, B_1')]\!]$
by lemma 7.22

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1)\ (\text{box } \lambda\{\hat{\Psi}\}.\, V_2)) \in [\![\mathcal{C}\ (B, A, B_1')]\!]$
by definition SBApp

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(V_1\ V_2)) \in [\![\mathcal{C}\ (B, A, B_1')]\!]$ by definition SeApp

2. **Case:** $\mathcal{E} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \hat{\Psi} \vdash V \downarrow a\end{matrix}}{\hat{\Psi} \vdash V \Uparrow a}\,\mathsf{CanAt}$

$\Pi\{\hat{\Psi}\}.\, a = B$ by assumption

$\Rightarrow \quad \tau(B) = \tau(\Pi\{\hat{\Psi}\}.\, a) = \tau(a) = a$ by lemma 6.28, by definition 4.15

$\Rightarrow \quad \Pi\{\hat{\Psi}\}.\, \tau(B) = B$ by definition

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; V\}(\hat{\Psi})) \in [\![\mathcal{C}\ (B, A, a)]\!]$ by ind. hyp. (1)

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; V\}(\hat{\Psi})) \in [\![\mathcal{C}^*\ (B, A, a)]\!]$ by definition 5.12

Case: $\mathcal{E} = \dfrac{\begin{matrix}\mathcal{E}_1 \\ \hat{\Psi}, x : B_1' \vdash V \Uparrow B_2'\end{matrix}}{\hat{\Psi} \vdash \lambda x\!:\!B_1'.\, V \Uparrow B_1' \to B_2'}\,\mathsf{CanLam}$

$(\tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1'))(u) = \mathcal{C}\ (B, A, B_1')$ by definition

$\Rightarrow \quad \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \vdash u \downarrow \mathcal{C}\ (B, A, B_1')$ by application of AtVar

$\Rightarrow \quad \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \vdash u \in |\mathcal{C}\ (B, A, B_1')|$ by lemma 7.17

$\Rightarrow \quad \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \vdash u \in [\![\mathcal{C}\ (B, A, B_1')]\!]$ by lemma 7.21

$\Psi + \tilde{\Psi} \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ by assumption

$\tilde{\Psi} \geq \tilde{\Psi}$ by definition CeBase

$\Rightarrow \quad \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \geq \tilde{\Psi}$ by definition CeInd

$\Rightarrow \quad \Psi + \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \vdash \Xi \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi})]\!]$ by lemma 7.11

$\Rightarrow \quad \Psi + \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1') \vdash \Xi \mid x \Rightarrow u \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \hat{\Psi}, x : B_1')]\!]$ by definition 7.7

Let $\bar{\Psi}'' \geq \Psi \cup \bar{\Psi}$

$\Rightarrow \quad \bar{\Psi}'' = \Psi \cup \bar{\Psi}'$ by lemma 6.4

$\Rightarrow \quad \bar{\Psi}' \geq \bar{\Psi}$ by lemma 6.4

Let $\Psi \cup \bar{\Psi}' \vdash V' \in |\mathcal{C}\ (B, A, B_1')|$

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash \cdot;\mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$ by assumption

$\Rightarrow \quad \Psi \cup \bar{\Psi}' \vdash \cdot;\mathrm{id}_\Psi \cup \varrho \in [\cdot; \Psi \cup \tilde{\Psi}]$ by lemma 7.26

$\Rightarrow \quad \Psi \cup \bar{\Psi}' \vdash \cdot;(\mathrm{id}_\Psi \cup \varrho), V'/u \in [\cdot; (\Psi \cup \tilde{\Psi}), u : \mathcal{C}\ (B, A, B_1')]$ by lemma 7.35

$\Rightarrow \quad \Psi \cup \bar{\Psi}' \vdash \cdot;\mathrm{id}_\Psi \cup \varrho, V'/u \in [\cdot; \Psi \cup \tilde{\Psi}, u : \mathcal{C}\ (B, A, B_1')]$ by definition CeInd

$\Pi\{\hat{\Psi}\}.\, B_1' \to B_2' = B$ by assumption

$\Rightarrow \quad \Pi\{\hat{\Psi}, x : B_1'\}.\, B_2' = B$ by definition 5.9

$\Rightarrow \quad \Psi \cup \bar{\Psi}' \vdash [\cdot;\mathrm{id}_\Psi \cup \varrho, V'/u](\{B \Rightarrow A; \Xi \mid x \Rightarrow u; \hat{\Psi}, x : B_1'\}(V)) \in [\![\mathcal{C}^*\ (B, A, B_2')]\!]$

by ind. hyp. (2)

$\Rightarrow \quad \Psi \cup \bar{\Psi}' \vdash [V'/u]([\cdot; \mathrm{id}_\Psi \cup \varrho, u/u](\{B \Rightarrow A; \Xi \mid x \Rightarrow u; \hat{\Psi}, x : B_1'\}(V)))$
$\in [\![\mathcal{C}^*\ (B, A, B_2')]\!]$ by lemma 6.17 (1)

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash \lambda u{:}\mathcal{C}\ (B, A, B_1').\, [\cdot; \mathrm{id}_\Psi \cup \varrho, u/u](\{B \Rightarrow A; \Xi \mid x \Rightarrow u; \hat{\Psi}, x : B_1'\}(V))$
$\in |\mathcal{C}\ (B, A, B_1') \to \mathcal{C}^*\ (B, A, B_2')|$ by definition 7.4

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash \lambda u{:}\mathcal{C}\ (B, A, B_1').\, [\cdot; (\mathrm{id}_\Psi \cup \varrho), u/u](\{B \Rightarrow A; \Xi \mid x \Rightarrow u; \hat{\Psi}, x : B_1'\}(V))$
$\in |\mathcal{C}\ (B, A, B_1') \to \mathcal{C}^*\ (B, A, B_2')|$ by definition CuInd

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\lambda u{:}\mathcal{C}\ (B, A, B_1').\, \{B \Rightarrow A; \Xi \mid x \Rightarrow u; \hat{\Psi}, x : B_1'\}(V))$
$\in |\mathcal{C}\ (B, A, B_1') \to \mathcal{C}^*\ (B, A, B_2')|$ by definition SBLam

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(\lambda x{:}B_1'.\, V)) \in |\mathcal{C}\ (B, A, B_1') \to \mathcal{C}^*\ (B, A, B_2')|$
by definition SeLam

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(\lambda x{:}B_1'.\, V)) \in |\mathcal{C}^*\ (B, A, B_1' \to B_2')|$ by definition 5.12

$\Rightarrow \quad \Psi \cup \bar{\Psi} \vdash [\cdot; \mathrm{id}_\Psi \cup \varrho](\{B \Rightarrow A; \Xi; \hat{\Psi}\}(\lambda x{:}B_1'.\, V)) \in [\![\mathcal{C}^*\ (B, A, B_1' \to B_2')]\!]$ by lemma 7.21

□

Lemma 7.47 (Typing and logical relations) *Let* $\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$

1. *If* $\Delta; \Gamma \vdash M : A$ *then* $\Psi \vdash [\theta; \varrho](M) \in [\![A]\!]$
2. *If* $\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$ *then* $\Psi + \cdot \vdash [\theta; \varrho](\Xi) \in [\![\langle B \Rightarrow A\rangle(\Sigma'; \cdot)]\!]$
3. *If* $\Delta; \Gamma \vdash \Omega : \langle \omega\rangle(\Sigma')$ *then* $\Psi + \cdot \vdash [\theta; \varrho](\Omega) \in [\![\langle \omega\rangle(\Sigma'; \cdot)]\!]$

Proof: by induction over $\mathcal{D} :: \Delta; \Gamma \vdash M : A$

1. **Case:** $\mathcal{D} = \dfrac{\Gamma(x) = A}{\Delta; \Gamma \vdash x : A}$ TpVarReg

$\Gamma(x) = A$ by assumption
$\Rightarrow \quad \Gamma = \Gamma_1, x : A \cup \Gamma_2$ by definition 2.2
$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ by assumption
$\Rightarrow \quad \varrho(x) = M$ by lemma 7.34 (2)
$\Rightarrow \quad \Psi \vdash M \in [\![A]\!]$ by lemma 7.34 (2)
$\Rightarrow \quad [\theta; \varrho](x) = M$ by definition SBVar
$\Rightarrow \quad \Psi \vdash [\theta; \varrho](x) \in [\![A]\!]$ by definition

Case: $\mathcal{D} = \dfrac{\Delta(x) = A}{\Delta; \Gamma \vdash x : A}$ TpVarMod

$\Delta(x) = A$ by assumption
$\Rightarrow \quad \Delta = \Delta_1, x : A \cup \Delta_2$ by definition 2.2
$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ by assumption
$\Rightarrow \quad \theta(x) = M$ by lemma 7.34 (1)
$\Rightarrow \quad \cdot \vdash M \in [\![A]\!]$ by lemma 7.34 (1)

$\Rightarrow$ $[\theta;\varrho](x) = M$ by definition SBVar
$\Rightarrow$ $\cdot \vdash [\theta;\varrho](x) \in [\![A]\!]$ by definition
$\Rightarrow$ $\Psi \geq \cdot$ by lemma 6.2
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](x) \in [\![A]\!]$ by lemma 7.9 (1)

Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\Delta;\Gamma \vdash c : B}\,\mathsf{TpConst}$

$\Sigma(c) = B$ by assumption
$\Rightarrow$ $\Psi \vdash c \downarrow B$ by application of AtConst
$\Rightarrow$ $\Psi \vdash c \in |B|$ by lemma 7.17 (2)
$\Rightarrow$ $\Psi \vdash c \hookrightarrow c : B$ by application of EvConst
$\Rightarrow$ $\cdot;\Psi \vdash c : B$ by application of TpConst
$\Rightarrow$ $\Psi \vdash c \in [\![B]\!]$ by definition 7.4
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](c) \in [\![B]\!]$ by definition SBConst

Case: $\mathcal{D} = \dfrac{\Delta;\Gamma, x : A_1 \vdash M : A_2}{\Delta;\Gamma \vdash \lambda x\!:\!A_1.\, M : A_1 \to A_2}\,\mathsf{TpLam}$

Let $\Psi' \geq \Psi$ by assumption
Let $\Psi' \vdash V \in |A_1|$ by assumption
$\Rightarrow$ $\Psi' \vdash V \in [\![A_1]\!]$ by lemma 7.21
$\Psi \vdash \theta;\varrho \in [\Delta;\Gamma]$ by assumption
$\Rightarrow$ $\Psi' \vdash \theta;\varrho \in [\Delta;\Gamma]$ by lemma 7.26 (2)
$\Rightarrow$ $\Psi' \vdash \theta;\varrho, V/x \in [\Delta;\Gamma, x : A_1]$ by assumption
$\Rightarrow$ $\Psi' \vdash [\theta;\varrho, V/x](M) \in [\![A_2]\!]$ by ind. hyp.
$\Rightarrow$ $\Psi' \vdash [V/x]([\theta;\varrho, x/x](M)) \in [\![A_2]\!]$ by lemma 6.17 (1)
$\Rightarrow$ $\Psi \vdash \lambda x\!:\!A_1.\,[\theta;\varrho, x/x](M) \in |A_1 \to A_2|$ by definition 7.4
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](\lambda x\!:\!A_1.\, M) \in |A_1 \to A_2|$ by definition SBLam
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](\lambda x\!:\!A_1.\, M) \in [\![A_1 \to A_2]\!]$ by lemma 7.21

Case: $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Delta;\Gamma \vdash M_1 : A_2 \to A_1 & \Delta;\Gamma \vdash M_2 : A_2\end{array}}{\Delta;\Gamma \vdash M_1\ M_2 : A_1}\,\mathsf{TpApp}$

$\Rightarrow$ $\Psi \vdash [\theta;\varrho](M_1) \in [\![A_2 \to A_1]\!]$ by ind. hyp. on $\mathcal{D}_1$
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](M_2) \in [\![A_2]\!]$ by ind. hyp. on $\mathcal{D}_2$
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](M_1\ M_2) \in [\![A_1]\!]$ by lemma 7.22

Case: $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Delta;\Gamma \vdash M_1 : A_1 & \Delta;\Gamma \vdash M_2 : A_2\end{array}}{\Delta;\Gamma \vdash \langle M_1, M_2\rangle : A_1 \times A_2}\,\mathsf{TpPair}$

$\Rightarrow$ $\Psi \vdash [\theta;\varrho](M_1) \in [\![A_1]\!]$ by ind. hyp. on $\mathcal{D}_1$
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](M_2) \in [\![A_2]\!]$ by ind. hyp. on $\mathcal{D}_2$
$\Rightarrow$ $\Psi \vdash \langle [\theta;\varrho](M_1), [\theta;\varrho](M_2)\rangle \in |A_1 \times A_2|$ by definition 7.4
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](\langle M_1, M_2\rangle) \in |A_1 \times A_2|$ by definition SBPair
$\Rightarrow$ $\Psi \vdash [\theta;\varrho](\langle M_1, M_2\rangle) \in [\![A_1 \times A_2]\!]$ by lemma 7.21

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash M : A_1 \times A_2\end{array}}{\Delta; \Gamma \vdash \text{fst } M : A_1}\ \textsf{TpFst}$

$\Rightarrow\ \Psi \vdash [\theta; \varrho](M) \in [\![A_1 \times A_2]\!]$	by ind. hyp. on $\mathcal{D}$
$\Rightarrow\ \cdot; \Psi \vdash [\theta; \varrho](M) : A_1 \times A_2$	by definition 7.4
$\Rightarrow\ \Psi \vdash [\theta; \varrho](M) \hookrightarrow V' : A_1 \times A_2$	by definition 7.4
$\Rightarrow\ \Psi \vdash V' \in \lvert A_1 \times A_2 \rvert$	by definition 7.4
$\Rightarrow\ V' = \langle M_1, M_2 \rangle$	by definition 7.4
$\Rightarrow\ \Psi \vdash M_1 \in [\![A_1]\!]$	by definition 7.4
$\Rightarrow\ \Psi \vdash M_1 \hookrightarrow V : A_1$	by definition 7.4
$\Rightarrow\ \Psi \vdash V \in \lvert A_1 \rvert$	by definition 7.4
$\Rightarrow\ \cdot; \Psi \vdash \text{fst } [\theta; \varrho](M) : A_1$	by application of TpFst
$\Rightarrow\ \Psi \vdash \text{fst } [\theta; \varrho](M) \hookrightarrow V : A_1$	by application of EvFst
$\Rightarrow\ \Psi \vdash \text{fst } [\theta; \varrho](M) \in [\![A_1]\!]$	by definition 7.4
$\Rightarrow\ \Psi \vdash [\theta; \varrho](\text{fst } M) \in [\![A_1]\!]$	by definition SBFst

Case: $\mathcal{D} = \dfrac{\Delta; \Gamma \vdash M : A_1 \times A_2}{\Delta; \Gamma \vdash \text{snd } M : A_2}\ \textsf{TpSnd}$

analog

Case: $\mathcal{D} = \dfrac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \text{box } M : \Box A}\ \textsf{TpBox}$

$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$	by assumption
$\Rightarrow\ \cdot \vdash \theta; \cdot \in [\Delta; \cdot]$	by lemma 7.27
$\Rightarrow\ \cdot \vdash [\theta; \cdot](M) \in [\![A]\!]$	by ind. hyp.
$\Rightarrow\ \cdot; \cdot \vdash [\theta; \cdot](M) : A$	by definition 7.4
$\Rightarrow\ \cdot; \Psi \vdash \text{box } [\theta; \cdot](M) : \Box A$	by application of TpBox
$\Rightarrow\ \Psi \vdash \text{box } [\theta; \cdot](M) \hookrightarrow \text{box } [\theta; \cdot](M) : \Box A$	by application of EvBox
$\Rightarrow\ \Psi \vdash \text{box } [\theta; \cdot](M) \in \lvert \Box A \rvert$	by definition 7.4
$\Rightarrow\ \Psi \vdash \text{box } [\theta; \cdot](M) \in [\![\Box A]\!]$	by definition 7.4
$\Rightarrow\ \Psi \vdash [\theta; \varrho](\text{box } M) \in [\![\Box A]\!]$	by definition SBBox

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ \Delta; \Gamma \vdash M_1 : \Box A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Delta, x : A_1; \Gamma \vdash M_2 : A_2\end{array}}{\Delta; \Gamma \vdash \text{let box } x = M_1 \text{ in } M_2 : A_2}\ \textsf{TpLet}$

$\Psi \vdash [\theta; \varrho](M_1) \in [\![\Box A_1]\!]$	by ind. hyp. on $\mathcal{D}_1$
$\Rightarrow\ \Psi \vdash [\theta; \varrho](M_1) \hookrightarrow V : \Box A_1$	by definition 7.4
$\Rightarrow\ \cdot; \Psi \vdash [\theta; \varrho](M_1) : \Box A_1$	by definition 7.4
$\Rightarrow\ \Psi \vdash V \in \lvert \Box A_1 \rvert$	by definition 7.4
$\Rightarrow\ V = \text{box } M_1'$	by definition 7.4
$\Rightarrow\ \cdot \vdash M_1' \in [\![A_1]\!]$	by definition 7.4
$\Rightarrow\ \Psi \vdash \theta, M_1'/x; \varrho \in [\Delta, x : A_1; \Gamma]$	by definition 7.25
$\Rightarrow\ \Psi \vdash [\theta, M_1'/x; \varrho](M_2) \in [\![A_2]\!]$	by ind. hyp. on $\mathcal{D}_2$

$\Rightarrow\quad \Psi \vdash [\theta, M_1'/x; \varrho](M_2) \hookrightarrow V' : A_2$ by definition 7.4
$\Rightarrow\quad \Psi \vdash V' \in |A_2|$ by definition 7.4
$\Rightarrow\quad \Psi \vdash [M_1'/x]([\theta, x/x; \varrho](M_2)) \hookrightarrow V' : A_2$ by lemma 6.17 (2)
$\Rightarrow\quad \Psi \vdash \text{let box } x = [\theta; \varrho](M_1) \text{ in } [\theta, x/x; \varrho](M_2) \hookrightarrow V' : A_2$ by application of EvLet
$\Rightarrow\quad \Psi \vdash [\theta; \varrho](\text{let box } x = M_1 \text{ in } M_2) \hookrightarrow V' : A_2$ by definition SBLet
$\Psi \vdash \theta; \varrho \in [\Delta; \Gamma]$ by assumption
$\Rightarrow\quad \cdot; \Psi \vdash (\theta; \varrho) \ : \ (\Delta; \Gamma)$ by lemma 7.31
$\Rightarrow\quad \cdot; \Psi \vdash [\theta; \varrho](\text{let box } x = M_1 \text{ in } M_2) : A_2$ by lemma 6.21 on $\mathcal{D}$
$\Rightarrow\quad \Psi \vdash [\theta; \varrho](\text{let box } x = M_1 \text{ in } M_2) \in [\![A_2]\!]$ by definition 7.4

Case: $\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Delta; \Gamma \vdash M : \Box B & \Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma') \end{array}}{\Delta; \Gamma \vdash \text{case } \langle A \rangle \ M \ \langle \Xi \rangle : \mathcal{C}^* \ (B, A, B)}\ \mathsf{TpCase}$

$\Sigma' = \mathcal{S}(\Sigma; \tau(B))$ Side Condition
$\Psi \vdash [\theta; \varrho](M) \in [\![\Box B]\!]$ by ind. hyp. on $\mathcal{D}_1$
$\Rightarrow\quad \Psi \vdash [\theta; \varrho](M) \hookrightarrow \text{box } M' : \Box B$ by definition 7.4
$\Rightarrow\quad \cdot; \Psi \vdash [\theta; \varrho](M) : \Box B$ by definition 7.4
$\Rightarrow\quad \Psi \vdash \text{box } M' \in |\Box B|$ by definition 7.4
$\Rightarrow\quad \cdot \vdash M' \in [\![B]\!]$ by definition 7.4
$\Rightarrow\quad \cdot \vdash M' \Uparrow V' : B$ by lemma 7.17 (1)
$\Rightarrow\quad \cdot \vdash V' \Uparrow B$ by lemma 7.2 (1)
$\Delta; \Gamma \vdash \Xi : \langle B \Rightarrow A \rangle(\Sigma')$ by assumption
$\Rightarrow\quad \Psi + \cdot \vdash [\theta; \varrho](\Xi) \in [\![\langle B \Rightarrow A \rangle(\Sigma'; \cdot)]\!]$ by ind. hyp. (2)
$\Rightarrow\quad \cdot; \Psi \vdash [\theta; \varrho](\Xi) : \langle B \Rightarrow A \rangle(\Sigma')$ by lemma 7.8
$\Rightarrow\quad \Psi \vdash \cdot; \text{id}_\Psi \in [\cdot; \Psi]$ by definition 7.37
$\Rightarrow\quad \Psi \cup \cdot \vdash \cdot; \text{id}_\Psi \cup \cdot \in [\cdot; \Psi \cup \cdot]$ by definition CuBase
$\Pi\{\cdot\}.\, B = B$ by definition 5.9
$\Rightarrow\quad \Psi \cup \cdot \vdash [\cdot; \text{id}_\Psi \cup \cdot](\{B \Rightarrow A; \Xi; \cdot\}(V')) \in [\![\mathcal{C}^* \ (B, A, B)]\!]$ by lemma 7.46
$\Rightarrow\quad \Psi \vdash [\cdot; \text{id}_\Psi](\{B \Rightarrow A; \Xi; \cdot\}(V')) \in [\![\mathcal{C}^* \ (B, A, B)]\!]$ by definition CuBase
$\Rightarrow\quad \cdot; \Psi \vdash [\cdot; \text{id}_\Psi](\{B \Rightarrow A; \Xi; \cdot\}(V')) : \mathcal{C}^* \ (B, A, B)$ by definition 7.4
$\Rightarrow\quad \{B \Rightarrow A; \Xi; \cdot\}(V') = [\cdot; \text{id}_\Psi](\{B \Rightarrow A; \Xi; \cdot\}(V'))$
$\Rightarrow\quad \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \in [\![\mathcal{C}^* \ (B, A, B)]\!]$ by definition
$\Rightarrow\quad \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \hookrightarrow V : \mathcal{C}^* \ (B, A, B)$ by definition 7.4
$\Rightarrow\quad \Psi \vdash V \in |\mathcal{C}^* \ (B, A, B)|$ by definition 7.4
$\Rightarrow\quad \Psi \vdash \text{case } \langle A \rangle \ [\theta; \varrho](M) \ \langle [\theta; \varrho](\Xi) \rangle \hookrightarrow V : \mathcal{C}^* \ (B, A, B)$ by application of EvCase
$\Rightarrow\quad \cdot; \Psi \vdash \text{case } \langle A \rangle \ [\theta; \varrho](M) \ \langle [\theta; \varrho](\Xi) \rangle : \mathcal{C}^* \ (B, A, B)$ by application of TpCase
$\Rightarrow\quad \Psi \vdash \text{case } \langle A \rangle \ [\theta; \varrho](M) \ \langle [\theta; \varrho](\Xi) \rangle \in [\![\mathcal{C}^* \ (B, A, B)]\!]$ by definition 7.4
$\Rightarrow\quad \Psi \vdash [\theta; \varrho](\text{case } \langle A \rangle \ M \ \langle \Xi \rangle) \in [\![\mathcal{C}^* \ (B, A, B)]\!]$ by definition SBCase

Case: $\mathcal{D} = \dfrac{\Delta; \Gamma \vdash M : \Box B \qquad \vdash \omega : \alpha \qquad \Delta; \Gamma \vdash \Omega : \langle \omega \rangle(\Sigma')}{\Delta; \Gamma \vdash \text{it } \langle \omega \rangle \ M \ \langle \Omega \rangle : \langle \omega \rangle(B)}\ \mathsf{TpIt}$

$\Delta; \Gamma \vdash M : \Box B$ by assumption

$\Psi \vdash [\theta;\varrho](M) \in [\![\Box B]\!]$	by ind. hyp. (1)
$\Rightarrow\quad \Psi \vdash [\theta;\varrho](M) \hookrightarrow \text{box } M' : \Box B$	by definition 7.4
$\Rightarrow\quad \cdot;\Psi \vdash [\theta;\varrho](M) : \Box B$	by definition 7.4
$\Rightarrow\quad \Psi \vdash \text{box } M' \in \lvert\Box B\rvert$	by definition 7.4
$\Rightarrow\quad \cdot \vdash M' \in [\![B]\!]$	by definition 7.4
$\Rightarrow\quad \cdot \vdash M' \Uparrow V' : B$	by lemma 7.17 (1)
$\Rightarrow\quad \cdot \vdash V' \Uparrow B$	by lemma 7.2 (1)
$\Delta;\Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$	by assumption
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\Omega) \in [\![\langle\omega\rangle(\Sigma';\cdot)]\!]$	by ind. hyp. (3)
$\Rightarrow\quad \cdot;\Psi \vdash [\theta;\varrho](\Omega) : \langle\omega\rangle(\Sigma')$	by lemma 7.6
$\Rightarrow\quad \Psi \vdash \cdot;\mathrm{id}_\Psi \in [\cdot;\Psi]$	by definition 7.37
$\Rightarrow\quad \Psi \cup \cdot \vdash \cdot;\mathrm{id}_\Psi \cup \cdot \in [\cdot;\Psi \cup \cdot]$	by definition CuBase
$\Rightarrow\quad \Psi \cup \cdot \vdash [\cdot;\mathrm{id}_\Psi \cup \cdot](\langle\omega;\Omega\rangle(V')) \in [\![\langle\omega\rangle(B)]\!]$	by lemma 7.43
$\Rightarrow\quad \Psi \vdash [\cdot;\mathrm{id}_\Psi](\langle\omega;\Omega\rangle(V')) \in [\![\langle\omega\rangle(B)]\!]$	by definition CuBase
$\Rightarrow\quad \cdot;\Psi \vdash [\cdot;\mathrm{id}_\Psi](\langle\omega;\Omega\rangle(V')) : \langle\omega\rangle(B)$	by definition 7.4
$\Rightarrow\quad \langle\omega;\Omega\rangle(V') = [\cdot;\mathrm{id}_\Psi](\langle\omega;\Omega\rangle(V'))$	
$\Rightarrow\quad \Psi \vdash \langle\omega;\Omega\rangle(V') \in [\![\langle\omega\rangle(B)]\!]$	by definition
$\Rightarrow\quad \Psi \vdash \langle\omega;\Omega\rangle(V') \hookrightarrow V : \langle\omega\rangle(B)$	by definition 7.4
$\Rightarrow\quad \Psi \vdash V \in \lvert\langle\omega\rangle(B)\rvert$	by definition 7.4
$\Rightarrow\quad \Psi \vdash \text{it } \langle\omega\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Omega)\rangle \hookrightarrow V : \langle\omega\rangle(B)$	by application of EvIt
$\vdash \omega : \alpha$	by assumption
$\Rightarrow\quad \cdot;\Psi \vdash \text{it } \langle\omega\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Omega)\rangle : \langle\omega\rangle(B)$	by application of TpIt
$\Rightarrow\quad \Psi \vdash \text{it } \langle\omega\rangle\ [\theta;\varrho](M)\ \langle[\theta;\varrho](\Omega)\rangle \in [\![\langle\omega\rangle(B)]\!]$	by definition 7.4
$\Rightarrow\quad \Psi \vdash [\theta;\varrho](\text{it } \langle\omega\rangle\ M\ \langle\Omega\rangle) \in [\![\langle\omega\rangle(B)]\!]$	by definition SBIt

2. **Case:** $\mathcal{D} = \dfrac{}{\Delta;\Gamma \vdash \cdot : \langle B \Rightarrow A\rangle(\cdot)}\,\mathsf{TmBase}$

$\Psi + \cdot \vdash \cdot \in [\![\langle B \Rightarrow A\rangle(\cdot;\cdot)]\!]$	by definition 7.7
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\cdot) \in [\![\langle B \Rightarrow A\rangle(\cdot;\cdot)]\!]$	by application of SBXiEmpty

Case: $\mathcal{D} = \dfrac{\Delta;\Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma') \qquad \Delta;\Gamma \vdash M : \mathcal{C}\ (B, A, B')}{\Delta;\Gamma \vdash (\Xi \mid c \mapsto M) : \langle B \Rightarrow A\rangle(\Sigma', c : B')}\,\mathsf{TmInd}$

$\Delta;\Gamma \vdash \Xi : \langle B \Rightarrow A\rangle(\Sigma')$	by assumption
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\Xi) \in [\![\langle B \Rightarrow A\rangle(\Sigma';\cdot)]\!]$	by ind. hyp. (2)
$\Delta;\Gamma \vdash M : \mathcal{C}\ (B, A, B')$	by assumption
$\Rightarrow\quad \Psi \vdash [\theta;\varrho](M) \in [\![\mathcal{C}\ (B, A, B')]\!]$	by ind. hyp. (1)
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\Xi) \mid c \mapsto [\theta;\varrho](M) \in [\![\langle B \Rightarrow A\rangle(\Sigma', c : B';\cdot)]\!]$	by definition 7.7
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\Xi \mid c \mapsto M) \in [\![\langle B \Rightarrow A\rangle(\Sigma', c : B';\cdot)]\!]$	by application of SBXi

3. **Case:** $\mathcal{D} = \dfrac{}{\Delta;\Gamma \vdash \cdot : \langle\omega\rangle(\cdot)}\,\mathsf{TrBase}$

$\Psi + \cdot \vdash \cdot \in [\![\langle\omega\rangle(\cdot;\cdot)]\!]$	by definition 7.5
$\Rightarrow\quad \Psi + \cdot \vdash [\theta;\varrho](\cdot) \in [\![\langle\omega\rangle(\cdot;\cdot)]\!]$	by application of SBOmegaEmpty

Case: $\mathcal{D} = \dfrac{\Delta;\Gamma \vdash \Omega : \langle\omega\rangle(\Sigma') \qquad \Delta;\Gamma \vdash M : \langle\omega\rangle(B')}{\Delta;\Gamma \vdash (\Omega \mid c \mapsto M) : \langle\omega\rangle(\Sigma', c : B')}$ TrInd

$\Delta;\Gamma \vdash \Omega : \langle\omega\rangle(\Sigma')$ by assumption

$\Rightarrow \quad \Psi + \cdot \vdash [\theta;\varrho](\Omega) \in [\![\langle\omega\rangle(\Sigma';\cdot)]\!]$ by ind. hyp. (3)

$\Delta;\Gamma \vdash M : \langle\omega\rangle(B')$ by assumption

$\Rightarrow \quad \Psi \vdash [\theta;\varrho](M) \in [\![\langle\omega\rangle(B')]\!]$ by ind. hyp. (1)

$\Rightarrow \quad \Psi + \cdot \vdash [\theta;\varrho](\Omega) \mid c \mapsto [\theta;\varrho](M) \in [\![\langle\omega\rangle(\Sigma', c : B';\cdot)]\!]$ by definition 7.5

$\Rightarrow \quad \Psi + \cdot \vdash [\theta;\varrho](\Omega \mid c \mapsto M) \in [\![\langle\omega\rangle(\Sigma', c : B';\cdot)]\!]$ by application of SBOmega

□

Theorem 7.48 (Canonical form theorem)

If $\cdot;\Psi \vdash M : B$ *then* $\Psi \vdash M \Uparrow V : B$

Proof: $\cdot;\Psi \vdash M : B$ by assumption

$\Rightarrow \quad \Psi \vdash \cdot;\mathrm{id}_\Psi \in [\cdot;\Psi]$ by lemma 7.37

$\Rightarrow \quad \Psi \vdash [\cdot;\mathrm{id}_\Psi](M) \in [\![B]\!]$ by lemma 7.47

$\Rightarrow \quad \cdot;\Psi \vdash [\cdot;\mathrm{id}_\Psi](M) : B$ by definition 7.4

$\Rightarrow \quad \Psi \vdash M \in [\![B]\!]$ by lemma 6.23

$\Rightarrow \quad \Psi \vdash M \Uparrow V : B$ by lemma 7.17 (1)

□

D Type preservation theorem

Lemma 8.1 (Uniqueness of evaluation)

1. *If* $\Psi \vdash M \Uparrow V : A$ *and* $\Psi \vdash M \Uparrow V' : A$ *then* $V = V'$
2. *If* $\Psi \vdash M \hookrightarrow V : A$ *and* $\Psi \vdash M \hookrightarrow V' : A$ *then* $V = V'$

Proof: by induction over $\mathcal{D} :: \Psi \vdash M \Uparrow V : A$ and $\mathcal{E} :: \Psi M V A$

1. **Case:** $\mathcal{E} = \dfrac{\Psi \vdash M \hookrightarrow V : a}{\Psi \vdash M \Uparrow V : a}\,\mathsf{EcAtomic}$:

   $\Psi \vdash M \hookrightarrow V' : a$ — by inversion using $\mathsf{EcAtomic}$
   $\Rightarrow \quad V = V'$ — by ind. hyp. (2)

   Case: $\mathcal{E} = \dfrac{\Psi, x : B_1 \vdash M\ x \Uparrow V_1 : B_2}{\Psi \vdash M \Uparrow \lambda x\!:\!B_1.\, V_1 : B_1 \to B_2}\,\mathsf{EcArrow}$:

   $\Psi, x : B_1 \vdash M\ x \Uparrow V_1' : B_2$ — by inversion using $\mathsf{EcArrow}$
   $\Rightarrow \quad V_1 = V_1'$ — by ind. hyp. (2)
   $\Rightarrow \quad \lambda x\!:\!B_1.\, V_1 = \lambda x\!:\!B_1.\, V_1'$ — by definition

2. **Case:** $\mathcal{D} = \dfrac{\Psi(x) = A}{\Psi \vdash x \hookrightarrow x : A}\,\mathsf{EvVar}$:

   $x = x$ — by definition

   Case: $\mathcal{D} = \dfrac{\Sigma(c) = B}{\Psi \vdash c \hookrightarrow c : B}\,\mathsf{EvConst}$:

   $c = c$ — by definition

   Case: $\mathcal{D} = \dfrac{\cdot; \Psi, x : A_1 \vdash M : A_2}{\Psi \vdash \lambda x\!:\!A_1.\, M \hookrightarrow \lambda x\!:\!A_1.\, M : A_1 \to A_2}\,\mathsf{EvLam}$:

   $\lambda x\!:\!A_1.\, M = \lambda x\!:\!A_1.\, M$ — by definition

   Case: $\mathcal{D} = \dfrac{\Psi \vdash M_1 \hookrightarrow \lambda x\!:\!A_2.\, M_1' : A_2 \to A_1 \qquad \Psi \vdash M_2 \hookrightarrow V_2 : A_2 \qquad \Psi \vdash [V_2/x](M_1') \hookrightarrow V : A_1}{\Psi \vdash M_1\ M_2 \hookrightarrow V : A_1}\,\mathsf{EvApp}$:

   $\Psi \vdash M_1 \hookrightarrow \lambda x\!:\!A_2.\, M_1'' : A_2 \to A_1$ — by inversion using EvApp
   $\Rightarrow \quad \lambda x\!:\!A_2.\, M_1' = \lambda x\!:\!A_2.\, M_1''$ — by ind. hyp. (2)
   $\Psi \vdash M_2 \hookrightarrow V_2' : A_2$ — by inversion using EvApp
   $\Rightarrow \quad V_2 = V_2'$ — by ind. hyp. (2)
   $\Rightarrow \quad [V_2/x](M_1') = [V_2'/x](M_1'')$ — by definition
   $\Rightarrow \quad V = V'$ — by ind. hyp.

   Case: $\mathcal{D} = \dfrac{\Psi \vdash M_1 \hookrightarrow V_1 : B_2 \to B_1 \qquad \Psi \vdash V_1 \downarrow B_2 \to B_1 \qquad \Psi \vdash M_2 \Uparrow V_2 : B_2}{\Psi \vdash M_1\ M_2 \hookrightarrow V_1\ V_2 : B_1}\,\mathsf{EvAtomic}$:

   $\Psi \vdash M_1 \hookrightarrow V_1' : B_2 \to B_1$ — by inversion using $\mathsf{EvAtomic}$
   $\Rightarrow \quad V_1 = V_1'$ — by ind. hyp. (2)
   $\Psi \vdash M_2 \Uparrow V_2' : B_2$ — by inversion using $\mathsf{EvAtomic}$
   $\Rightarrow \quad V_2 = V_2'$ — by ind. hyp. (1)
   $\Rightarrow \quad V_1\ V_2 = V_1'\ V_2'$

Case: $\mathcal{D} = \dfrac{\cdot;\Psi \vdash M_1 : A_1 \qquad \cdot;\Psi \vdash M_2 : A_2}{\Psi \vdash \langle M_1, M_2\rangle \hookrightarrow \langle M_1, M_2\rangle : A_1 \times A_2}\,\mathsf{EvPair}$:

trivial

[Case:] $\mathcal{D} = \dfrac{\Psi \vdash M \hookrightarrow \langle M_1, M_2\rangle : A_1 \times A_2 \qquad \Psi \vdash M_1 \hookrightarrow V : A_1}{\Psi \vdash \text{fst } M \hookrightarrow V : A_1}\,\mathsf{EvFst}$:

trivial

Case: $\mathcal{D} = \dfrac{\Psi \vdash M \hookrightarrow \langle M_1, M_2\rangle : A_1 \times A_2 \qquad \Psi \vdash M_2 \hookrightarrow V : A_2}{\Psi \vdash \text{snd } M \hookrightarrow V : A_2}\,\mathsf{EvSnd}$:

trivial

Case: $\mathcal{D} = \dfrac{\cdot;\cdot \vdash M : A}{\Psi \vdash \text{box } M \hookrightarrow \text{box } M : \Box A}\,\mathsf{EvBox}$:

trivial

Case: $\mathcal{D} = \dfrac{\Psi \vdash M_1 \hookrightarrow \text{box } M_1' : \Box A \qquad \Psi \vdash [M_1'/x](M_2) \hookrightarrow V : A_2}{\Psi \vdash \text{let box } x = M_1 \text{ in } M_2 \hookrightarrow V : A_2}\,\mathsf{EvLet}$:

trivial

Case: $\mathcal{D} = \dfrac{\Psi \vdash M \hookrightarrow \text{box } M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \{B \Rightarrow A; \Xi; \cdot\}(V') \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}{\Psi \vdash \text{case } \langle A\rangle\ M\ \langle \Xi\rangle \hookrightarrow V : \mathcal{C}^*\ (B, A, B)}\,\mathsf{EvCase}$:

trivial

Case: $\mathcal{D} = \dfrac{\Psi \vdash M \hookrightarrow \text{box } M' : \Box B \qquad \cdot \vdash M' \Uparrow V' : B \qquad \Psi \vdash \langle \omega; \Omega\rangle(V') \hookrightarrow V : \langle\omega\rangle(B)}{\Psi \vdash \text{it } \langle\omega\rangle\ M\ \langle\Omega\rangle \hookrightarrow V : \langle\omega\rangle(B)}\,\mathsf{EvIt}$:

trivial

□

Theorem 8.2 (Type preservation)

If $\cdot;\Psi \vdash M : A$ *and* $\Psi \vdash M \hookrightarrow V : A$ *then* $\cdot;\Psi \vdash V : A$

Proof: $\cdot;\Psi \vdash M : A$	by assumption
$\Psi \vdash \cdot;\text{id}_\Psi \in [\cdot;\Psi]$	by lemma 7.37
$\Rightarrow\ \Psi \vdash [\cdot;\text{id}_\Psi](M) \in [\![A]\!]$	by lemma 7.47
$\Rightarrow\ \Psi \vdash [\cdot;\text{id}_\Psi](M) \in [\![A]\!]$	by lemma 7.47
$\Rightarrow\ \cdot;\Psi \vdash [\cdot;\text{id}_\Psi](M) : A$	by definition 7.4
$\Rightarrow\ [\cdot;\text{id}_\Psi](M) = M$	by lemma 6.23
$\Rightarrow\ \Psi \vdash M \hookrightarrow V' : A$	by definition 7.4
$\Rightarrow\ V = V'$	by lemma 8.1
$\Rightarrow\ \Psi \vdash V \in \lvert A\rvert$	by definition 7.4
$\Rightarrow\ \Psi \vdash V \in [\![A]\!]$	by lemma 7.21
$\Rightarrow\ \cdot;\Psi \vdash V : A$	by definition 7.4

□

E Conservative extension theorem

Lemma 9.1 (Typing extension)

If $\Psi \vdash M : B$ *then* $\cdot; \Psi \vdash M : B$

Proof: by induction over $\Psi \vdash M : B$

Case: $\dfrac{\Psi(x) = B}{\Psi \vdash x : B}\,\mathsf{StpVar}$

$\cdot; \Psi \vdash x : B$ — by application of $\mathsf{TpVarReg}$

Case: $\dfrac{\Sigma(c) = B}{\Psi \vdash c : B}\,\mathsf{StpConst}$

$\cdot; \Psi \vdash c : B$ — by application of $\mathsf{TpConst}$

Case: $\dfrac{\Psi, x : B_1 \vdash M : B_2}{\Psi \vdash \lambda x{:}B_1.\, M : B_1 \to B_2}\,\mathsf{StpLam}$

$\Psi, x : B_1 \vdash M : B_2$ — by assumption

$\Rightarrow\ \cdot; \Psi, x : B_1 \vdash M : B_2$ — by ind. hyp.

$\Rightarrow\ \cdot; \Psi \vdash \lambda x{:}B_1.\, M : B_1 \to B_2$ — by application of TpLam

Case: $\dfrac{\Psi \vdash M_1 : B_2 \to B_1 \qquad \Psi \vdash M_2 : B_2}{\Psi \vdash M_1\ M_2 : B_1}\,\mathsf{StpApp}$

$\Psi \vdash M_1 : B_2 \to B_1$ — by assumption

$\Rightarrow\ \cdot; \Psi \vdash M_1 : B_2 \to B_1$ — by ind. hyp.

$\Psi \vdash \Psi : M_2 B_2$ — by assumption

$\Rightarrow\ \cdot; \Psi \vdash \Psi : M_2 B_2$ — by ind. hyp.

$\Rightarrow\ \cdot; \Psi \vdash M_1\ M_2 : B_1$ — by application of TpApp

□

Theorem 9.2 (Conservative Extension)

If $\cdot; \Psi \vdash M : B$ *then* $\Psi \vdash M \Uparrow V : B$ *and* $\Psi \vdash V \Uparrow B$

Proof: $\cdot; \Psi \vdash M : B$ — by assumption

$\Rightarrow\ \Psi \vdash M \Uparrow V : B$ — by theorem 7.48 (1)

$\Rightarrow\ \Psi \vdash V \Uparrow B$ — by lemma 7.2 (1)

□

References

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[CNSvS94] Thierry Coquand, Bengt Nordström, Jan M. Smith, and Björn von Sydow. Type theory and programming. *Bulletin of the European Association for Theoretical Computer Science*, 52:203–228, February 1994.

[DFH95] Joëlle Despeyroux, Amy Felty, and André Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 124–138, Edinburgh, Scotland, April 1995. Springer-Verlag LNCS 902.

[DH94] Joëlle Despeyroux and André Hirschowitz. Higher-order abstract syntax with induction in Coq. In Frank Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, pages 159–173, Kiev, Ukraine, July 1994. Springer-Verlag LNAI 822.

[DP96] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In Jr. Guy Steele, editor, *Proceedings of the 23rd Annual Symposium on Principles of Programming Languages*, pages 258–270, St. Petersburg Beach, Florida, January 1996. ACM Press.

[FS96] Leonidas Fegaras and Tim Sheard. Revisiting catamorphisms over datatypes with embedded functions (or, programs from outer space). In *Proceedings of 23rd Annual Symposium on Principles of Programming Languages*, pages 284–294, St. Petersburg Beach, Florida, January 1996. ACM Press.

[Göd90] Kurt Gödel. On an extension of finitary mathematics which has not yet been used. In Solomon Feferman et al., editors, *Kurt Gödel, Collected Works, Volume II*, pages 271–280. Oxford University Press, 1990.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[JO91] Jean-Pierre Jouannaud and Mitsuhiro Okada. A computation model for executable higher-order algebraic specification languages. In Gilles Kahn, editor, *Proceedings of the 6th Annual Symposium on Logic in Computer Science*, pages 350–361, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[Mag95] Lena Magnusson. *The Implementation of ALF—A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution.* PhD thesis, Chalmers University of Technology and Göteborg University, January 1995.

[MH95] Erik Meijer and Graham Hutton. Bananas in space: Extending fold and unfold to exponential types. In *Proceedings of the 7th Conference on Functional Programming Languages and Computer Architecture*, La Jolla, California, June 1995.

[Mil90] Dale Miller. An extension to ML to handle bound variables in data structures: Preliminary report. In *Proceedings of the Logical Frameworks BRA Workshop*, Nice, France, May 1990.

[Mil91] Dale Miller. Unification of simply typed lambda-terms as logic programming. In Koichi Furukawa, editor, *Eighth International Logic Programming Conference*, pages 255–269, Paris, France, June 1991. MIT Press.

[Mil92] Dale Miller. Abstract syntax and logic programming. In *Proceedings of the First and Second Russian Conferences on Logic Programming*, pages 322–337, Irkutsk and St. Petersburg, Russia, 1992. Springer-Verlag LNAI 592.

[MM96] Raymond McDowell and Dale Miller. A logic for reasoning about logic specifications. Draft manuscript, July 1996.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*, volume 7 of *International Series of Monographs on Computer Science*. Oxford University Press, 1990.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664.

[PW95] Frank Pfenning and Hao-Chi Wong. On a modal λ-calculus for S4. In S. Brookes and M. Main, editors, *Proceedings of the Eleventh Conference on Mathematical Foundations of Programming Sematics*, New Orleans, Louisiana, March 1995. To appear in *Electronic Notes in Theoretical Computer Science*, Volume 1, Elsevier.