The Logic of Metaslang

Alessandro Coglio

February 16, 2006

DRAFT; PLEASE DO NOT DISTRIBUTE

1 Introduction

This document formally defines the logic of the Metaslang language [1], drawing ideas from [2], [3], and [4, Part II].

The rest of this section introduces the mathematical notation used in this document. §2 defines the core (abstract) syntax of the Metaslang logic while §3 defines its proof theory. §4 states properties of the syntax and proof theory. §5 defines a set-theoretic semantics of the Metaslang logic. §6 shows how various Metaslang constructs are captured in terms of the core constructs. Finally, §A collects all the (meta) proofs of the (meta) theorems stated in the other sections.

1.1 Notation

We define the Metaslang logic in the usual semi-formal notation consisting of naive set theory and natural language. However, it is possible to define the Metaslang logic in axiomatic set theory or any other sufficiently expressive formal language.[1]

The (meta) logical notations $=$, $\forall$, $\exists$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $.\!\!\not.$ (e.g. $\neq$) have the usual meaning.

The set-theoretic notations $\in$, $\emptyset$, $\{\ldots \mid \ldots\}$, $\{\ldots\}$, $\cup$, $\cap$, and $\subseteq$ have the usual meaning.

$\mathbf{N}$ is the set of natural numbers, i.e. $\{0, 1, 2, \ldots\}$.

If A and B are sets, $A - B$ is their difference, i.e. $\{x \in A \mid x \notin B\}$.

If A and B are sets, $A \times B$ is their cartesian product, i.e. $\{\langle a, b\rangle \mid a \in A \ \wedge \ b \in B\}$. This generalizes to $n > 2$ sets.

If A and B are sets, $A + B$ is their disjoint union, i.e. $\{\langle 0, a\rangle \mid a \in A\} \cup \{\langle 1, b\rangle \mid b \in B\}$. The "tags" 0 and 1 are always left implicit. This generalizes to $n > 2$ sets.

If A and B are sets: $A \xrightarrow{\text{p}} B$ is the set of all partial functions from A to B, i.e. $\{f \subseteq A \times B \mid \forall \langle a, b_1\rangle, \langle a, b_2\rangle \in f.\ \ b_1 = b_2\}$; $A \to B$ is the set of all total functions from A to B, i.e. $\{f \in A \xrightarrow{\text{p}} B \mid \forall a \in A.\ \ \exists b \in B.\ \ \langle a, b\rangle \in f\}$; and $A \hookrightarrow B$ is the set of all total injective functions from A to B, i.e. $\{f \in A \to B \mid \forall \langle a_1, b\rangle, \langle a_2, b\rangle \in f.\ \ a_1 = a_2\}$. We write $f : A \xrightarrow{\text{p}} B$, $f : A \to B$, and $f : A \hookrightarrow B$ for $f \in A \xrightarrow{\text{p}} B$, $f \in A \to B$, and $f \in A \hookrightarrow B$, respectively.

If f is a function from A to B: $\mathcal{D}(f)$ is the domain of f, i.e. $\{a \in A \mid \exists b \in B.\ \ \langle a, b\rangle \in f\}$; $\mathcal{R}(f)$ is the range of f, i.e. $\{b \in B \mid \exists a \in A.\ \ \langle a, b\rangle \in f\}$.

If f is a function and $a \in \mathcal{D}(f)$, $f(a)$ denotes the unique value such that $\langle a, f(a)\rangle \in f$.

If f is a function and A is a set: $f{\downarrow_A}$ denotes the restriction of f to A, i.e. $\{\langle a, f(a)\rangle \mid a \in \mathcal{D}(f) \cap A\}$ (we may have $A \subseteq \mathcal{D}(f)$ or not); $f{\uparrow_A}$ denotes the restriction of f to $\mathcal{D}(f) - A$, i.e. $f{\downarrow_{\mathcal{D}(f)-A}}$.

If A is a set, $\mathcal{P}_\omega(A)$ is the set of all finite subsets of A, i.e. $\{S \subseteq A \mid S \text{ finite}\}$.

If A is a set, A^* is the set of all finite sequences of elements of A, i.e. $\{x_1, \ldots, x_n \mid x_1 \in A \wedge \ldots \wedge x_n \in A\}$; A^+, $A^{(*)}$, and $A^{(+)}$ are the subsets of A^* of non-empty sequences, sequences without repeated

[1] For example, Peter Homeier has formalized the Metaslang logic in the logic of the HOL theorem prover (see Acknowledgments section for details).

elements, and non-empty sequences without repeated elements, respectively.[2] The empty sequence is written ϵ. A sequence $x_1, \dots, x_n$ is often written $\overline{x}$, leaving n implicit. The length of a sequence s is written $|s|$. When a sequence is written where a set is expected, it stands for the set of its elements.

2 Syntax

2.1 Names

We postulate the existence of an infinite set of names

$$\mathcal{N}$$

2.2 Types

We inductively define the set of types as

$$\begin{aligned} \mathit{Type} = {} & \{\mathsf{Bool}\} \\ & + \{\beta \mid \beta \in \mathcal{N}\} \\ & + \{\tau[\overline{T}] \mid \tau \in \mathcal{N} \ \wedge \ \overline{T} \in \mathit{Type}^*\} \\ & + \{T_1 \to T_2 \mid T_1, T_2 \in \mathit{Type}\} \\ & + \{f_1\ T_1 \times \cdots \times f_n\ T_n \mid \overline{f} \in \mathcal{N}^{(*)} \ \wedge \ \overline{T} \in \mathit{Type}^*\} \\ & + \{T|r \mid T \in \mathit{Type} \ \wedge \ r \in \mathit{Exp}\} \end{aligned}$$

where *Exp* is defined later.[3]

Explanation:

- There is a type Bool for boolean (i.e. truth) values.
- A name β is a type variable.
- A type instance $\tau[\overline{T}]$ is obtained by combining a type name τ with zero or more argument types $\overline{T}$. We may write $\tau[\epsilon]$ as just τ.
- An arrow type $T_1 \to T_2$ consists of a domain T_1 and a range T_2.
- Record types $\prod_i f_i\ T_i$ consist of typed fields f_i. There exists one record type that has no fields (i.e. $n = 0$), denoted $\prod \epsilon$, whose only inhabitant is the empty record. All the fields of a record type must be distinct names.
- Restriction types $T|r$ are obtained by combining types T with expressions r (meant to be suitable predicates). Restriction types create the dependency of types on expressions. Restriction types as defined above also capture comprehension types as defined in [1]: as mentioned in [1], a comprehension type can be turned into a restriction type by re-combining the pattern and expression into a lambda expression.

We introduce the abbreviation

$$\textstyle\prod_i T_i \quad \longrightarrow \quad \prod_i \pi_i\ T_i$$

where each π_i is a fixed but unspecified name in $\mathcal{N}$ such that $\pi_i \neq \pi_j$ if $i \neq j$. Thus, record types as defined above also capture product types as defined in [1]; the names π_i capture the natural literal fields used in [1].

[2]Strictly speaking, our notation $x_1, \dots, x_n$ for sequences may lead to ambiguities, e.g. if s_1 and s_2 are sequences, is s_1, s_2 the sequence of length 2 whose elements are s_1 and s_2, or is it the concatenation of s_1 and s_2? However, in this document, the intended meaning should be always clear from the symbols used and from mathematical context.

[3]Types depend on expressions, which depend on types. Thus, types and expressions are inductively defined together, not separately. Their definitions are presented separately only for readability.

2.3 Expressions

We inductively define the set of expressions as

$$\begin{aligned} \mathit{Exp} = {} & \{v \mid v \in \mathcal{N}\} \\ & + \{o[\overline{T}] \mid o \in \mathcal{N} \ \wedge \ \overline{T} \in \mathit{Type}^*\} \\ & + \{e_1\ e_2 \mid e_1, e_2 \in \mathit{Exp}\} \\ & + \{\lambda v\!:\!T.\ e \mid v \in \mathcal{N} \ \wedge \ T \in \mathit{Type} \ \wedge \ e \in \mathit{Exp}\} \\ & + \{e_1 \equiv e_2 \mid e_1, e_2 \in \mathit{Exp}\} \\ & + \{\mathsf{if}\ e_0\ e_1\ e_2 \mid e_0, e_1, e_2 \in \mathit{Exp}\} \\ & + \{\iota_T \mid T \in \mathit{Type}\} \\ & + \{\mathsf{proj}_{\prod_i f_i\ T_i}\ f_j \mid \textstyle\prod_i f_i\ T_i \in \mathit{Type}\} \end{aligned}$$

Explanation:

- A name v is a variable.
- An op(eration) instance $o[\overline{T}]$ consists of an op name o and zero or more types $\overline{T}$ that instantiate the (generally, polymorphic) type of the declared op. We may write $o[\epsilon]$ as just o.
- An application $e_1\ e_2$ consists of a function e_1 juxtaposed to an argument e_2.
- A (lambda) abstraction $\lambda v\!:\!T.\ e$ consists of an argument v with an explicit type T and a body e. Even though lambda expressions as defined in [1] may have branches with patterns, that does not increase expressivity: one can imagine to use a fresh variable as argument of the abstraction and a case expression (introduced later) on the fresh variable with the branches as the body.
- An equality $e_1 \equiv e_2$ consists of a left-hand side e_1 and a right-hand side e_2. A conditional $\mathsf{if}\ e_0\ e_1\ e_2$ consists of a condition e_0, a "then" branch e_1, and an "else" branch e_2.
- The description operator ι_T is tagged by a type. It operates on predicates over T (i.e. over values of type $T \to \mathsf{Bool}$) that are satisfied by a unique value of T, and its result is *the* value that satisfies the predicate.
- A projector $\mathsf{proj}_{\prod_i f_i\ T_i}\ f_j$ is tagged by a record type and by a field of that record type. We may write $\mathsf{proj}_{\prod_i f_i\ T_i}\ f_j$ as just $\mathsf{proj}\ f_j$ when the record type is inferrable or irrelevant.

We introduce the abbreviations (many of which are expressions defined in [1])

$$\begin{array}{rcl}
\mathsf{true} & \longrightarrow & \lambda\gamma\!:\!\mathsf{Bool}.\ \gamma \equiv \lambda\gamma\!:\!\mathsf{Bool}.\ \gamma \\
\mathsf{false} & \longrightarrow & \lambda\gamma\!:\!\mathsf{Bool}.\ \gamma \equiv \lambda\gamma\!:\!\mathsf{Bool}.\ \mathsf{true} \\
\neg & \longrightarrow & \lambda\gamma\!:\!\mathsf{Bool}.\ (\mathsf{if}\ \gamma\ \mathsf{false}\ \mathsf{true}) \\
e_1 \wedge e_2 & \longrightarrow & \mathsf{if}\ e_1\ e_2\ \mathsf{false} \\
e_1 \vee e_2 & \longrightarrow & \mathsf{if}\ e_1\ \mathsf{true}\ e_2 \\
e_1 \Rightarrow e_2 & \longrightarrow & \mathsf{if}\ e_1\ e_2\ \mathsf{true} \\
\Leftrightarrow & \longrightarrow & \lambda\gamma\!:\!\mathsf{Bool}.\ \lambda\gamma'\!:\!\mathsf{Bool}.\ (\gamma \equiv \gamma') \\
e_1 \Leftrightarrow e_2 & \longrightarrow & \Leftrightarrow\ e_1\ e_2 \\
e_1 \not\equiv e_2 & \longrightarrow & \neg\ (e_1 \equiv e_2) \\
\iota v\!:\!T.e & \longrightarrow & \iota_T\ (\lambda v\!:\!T.\ e) \\
\forall_T & \longrightarrow & \lambda\psi\!:\!T \to \mathsf{Bool}.\ (\psi \equiv \lambda\gamma\!:\!T.\ \mathsf{true}) \\
\forall v\!:\!T.\ e & \longrightarrow & \forall_T\ (\lambda v\!:\!T.\ e) \\
\forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e & \longrightarrow & \forall v_1\!:\!T_1.\ \ldots \forall v_n\!:\!T_n.\ e \\
\forall \overline{v}\!:\!\overline{T}.\ e & \longrightarrow & \forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e \\
\exists_T & \longrightarrow & \lambda\psi\!:\!T \to \mathsf{Bool}.\ \neg\ (\forall\gamma\!:\!T.\ \neg\ (\psi\ \gamma)) \\
\exists v\!:\!T.\ e & \longrightarrow & \exists_T\ (\lambda v\!:\!T.\ e) \\
\exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e & \longrightarrow & \exists v_1\!:\!T_1.\ \ldots \exists v_n\!:\!T_n.\ e \\
\exists \overline{v}\!:\!\overline{T}.\ e & \longrightarrow & \exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e \\
\exists!_T & \longrightarrow & \lambda\psi\!:\!T \to \mathsf{Bool}.\ (\exists\gamma\!:\!T.\ (\psi\ \gamma \wedge \forall\gamma'\!:\!T.\ (\psi\ \gamma' \Rightarrow \gamma' \equiv \gamma))) \\
\exists!\, v\!:\!T.\ e & \longrightarrow & \exists!_T\ (\lambda v\!:\!T.\ e) \\
e.f & \longrightarrow & \mathsf{proj}\ f\ e
\end{array}$$

where γ, γ', and ψ are fixed but unspecified names in $\mathcal{N}$ that are all distinct.

Explanation:

- The abbreviations true and false stand for logical truth and falsehood, respectively.
- The logical connectives for negation, conjunction, disjunction, and implication are defined in terms of conditionals.
- Coimplication is a synonym for equality, but only for booleans.
- Inequality is negation of equality.
- Stand-alone quantifiers are higher-order functions, but they can be written in binder form. Also the description operator can be written in binder form. Note that both $\forall\epsilon.\ e$ and $\exists\epsilon.\ e$ stand for e.
- A dotted projection $e.f$ abbreviates an applied projection whose record type is implicit; this is why no record type appears in $e.f$.

The function $\mathcal{FV} : \mathit{Exp} \to \mathcal{P}_\omega(\mathcal{N})$ returns the free variables of an expression

$$\begin{array}{ll}
\mathcal{FV}(v) & = \{v\} \\
\mathcal{FV}(o[\overline{T}]) & = \emptyset \\
\mathcal{FV}(e_1\ e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(\lambda v\!:\!T.\ e) & = \mathcal{FV}(e) - \{v\} \\
\mathcal{FV}(e_1 \equiv e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(\mathsf{if}\ e_0\ e_1\ e_2) & = \mathcal{FV}(e_0) \cup \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(\iota_T) & = \emptyset \\
\mathcal{FV}(\mathsf{proj}\ f) & = \emptyset
\end{array}$$

Note that we do not consider the free variables in expressions contained in types contained in expressions (e.g. we do not consider the free variables in r as part of the free variables of $o[T|r]$). The reason is that, as defined later, all the expressions contained in well-formed types have no free variables.

2.4 Contexts

We define the set of context elements as

$$\begin{array}{rl}
\mathit{CxElem} = & \{\mathsf{ty}\ \tau\!:\!n \mid \tau \in \mathcal{N}\ \wedge\ n \in \mathbf{N}\} \\
+ & \{\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T \mid o \in \mathcal{N}\ \wedge\ \overline{\beta} \in \mathcal{N}^{(*)}\ \wedge\ T \in \mathit{Type}\} \\
+ & \{\mathsf{def}\ \tau[\overline{\beta}] = T \mid \tau \in \mathcal{N}\ \wedge\ \overline{\beta} \in \mathcal{N}^{(*)}\ \wedge\ T \in \mathit{Type}\} \\
+ & \{\mathsf{ax}\ \{\overline{\beta}\}\ e \mid \overline{\beta} \in \mathcal{N}^{(*)}\ \wedge\ e \in \mathit{Exp}\} \\
+ & \{\mathsf{lem}\ \{\overline{\beta}\}\ e \mid \overline{\beta} \in \mathcal{N}^{(*)}\ \wedge\ e \in \mathit{Exp}\} \\
+ & \{\mathsf{tvar}\ \beta \mid \beta \in \mathcal{N}\} \\
+ & \{\mathsf{var}\ v\!:\!T \mid v \in \mathcal{N}\ \wedge\ T \in \mathit{Type}\}
\end{array}$$

Explanation:

- A type declaration $\mathsf{ty}\ \tau\!:\!n$ introduces a type name with an associated arity. The type variables of a type declaration as defined in [1] only serve to determine an arity and are otherwise irrelevant; thus, in the above definition we directly use the arity without type variables.
- An op(eration) declaration $\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T$ introduces an op name with an associated type, polymorphic in the explicit type variables.
- A type definition $\mathsf{def}\ \tau[\overline{\beta}] = T$ assigns a type to a maximally generic type instance of some type name (i.e. an instance with distinct type variables as arguments to the type name). A combined type declaration and definition as defined in [1] is captured by a type declaration as defined above immediately followed by a type definition as defined above.

- An axiom $\mathsf{ax}\ \{\overline{\beta}\}\ e$ introduces an expression (with type Bool, as defined later), polymorphic in the explicit type variables. We may write $\mathsf{ax}\ \{\epsilon\}\ e$ as just $\mathsf{ax}\ e$.
- A lemma $\mathsf{lem}\ \{\overline{\beta}\}\ e$ captures a theorem or conjecture as defined in [1]. We may write $\mathsf{lem}\ \{\epsilon\}\ e$ as just $\mathsf{lem}\ e$.
- A type variable declaration $\mathsf{tvar}\ \beta$ introduces a type variable. We may write $\mathsf{tvar}\ \beta_1, \ldots, \mathsf{tvar}\ \beta_n$ as just $\mathsf{tvar}\ \beta_1, \ldots, \beta_n$.
- A variable declaration $\mathsf{var}\ v{:}T$ introduces a variable with a type.

We define the set of contexts as

$$\mathit{Cx} = \mathit{CxElem}^*$$

In other words, a context is a finite sequence of context elements.

The function $\mathcal{TN} : \mathit{Cx} \to \mathcal{P}_\omega(\mathcal{N})$ returns the type names declared in a context

$$\begin{aligned} \mathcal{TN}(\epsilon) &= \emptyset \\ \mathcal{TN}(\mathit{cxel}, \mathit{cx}) &= \begin{cases} \mathcal{TN}(\mathit{cx}) \cup \{\tau\} & \text{if} \quad \mathit{cxel} = \mathsf{ty}\ \tau{:}n \\ \mathcal{TN}(\mathit{cx}) & \text{otherwise} \end{cases} \end{aligned}$$

The function $\mathcal{ON} : \mathit{Cx} \to \mathcal{P}_\omega(\mathcal{N})$ returns the op names declared in a context

$$\begin{aligned} \mathcal{ON}(\epsilon) &= \emptyset \\ \mathcal{ON}(\mathit{cxel}, \mathit{cx}) &= \begin{cases} \mathcal{ON}(\mathit{cx}) \cup \{o\} & \text{if} \quad \mathit{cxel} = \mathsf{op}\ o{:}\{\overline{\beta}\}\ T \\ \mathcal{ON}(\mathit{cx}) & \text{otherwise} \end{cases} \end{aligned}$$

The function $\mathcal{TV} : \mathit{Cx} \to \mathcal{P}_\omega(\mathcal{N})$ returns the type variables declared in a context

$$\begin{aligned} \mathcal{TV}(\epsilon) &= \emptyset \\ \mathcal{TV}(\mathit{cxel}, \mathit{cx}) &= \begin{cases} \mathcal{TV}(\mathit{cx}) \cup \{\beta\} & \text{if} \quad \mathit{cxel} = \mathsf{tvar}\ \beta \\ \mathcal{TV}(\mathit{cx}) & \text{otherwise} \end{cases} \end{aligned}$$

The function $\mathcal{V} : \mathit{Cx} \to \mathcal{P}_\omega(\mathcal{N})$ returns the variables declared in a context

$$\begin{aligned} \mathcal{V}(\epsilon) &= \emptyset \\ \mathcal{V}(\mathit{cxel}, \mathit{cx}) &= \begin{cases} \mathcal{V}(\mathit{cx}) \cup \{v\} & \text{if} \quad \mathit{cxel} = \mathsf{var}\ v{:}T \\ \mathcal{V}(\mathit{cx}) & \text{otherwise} \end{cases} \end{aligned}$$

2.5 Specs

We define the set of spec(ification)s as

$$\mathit{Sp} = \{\mathit{cx} \in \mathit{Cx} \mid \mathcal{TV}(\mathit{cx}) = \mathcal{V}(\mathit{cx}) = \emptyset\}$$

In other words, a spec is a context without type variable declarations and variable declarations.

2.6 Substitutions

2.6.1 Type substitutions

The function $\_[\_] : (\mathit{Type} + \mathit{Exp}) \times (\mathcal{N} \xrightarrow{\mathrm{p}} \mathit{Type}) \to \mathit{Type} + \mathit{Exp}$ substitutes each type variable $\beta \in \mathcal{D}(\sigma)$, where $\sigma : \mathcal{N} \xrightarrow{\mathrm{p}} \mathit{Type}$, with the type $\sigma(\beta)$ in a type or expression x (written $x[\sigma]$)

$$\begin{aligned} \mathsf{Bool}[\sigma] &= \mathsf{Bool} \\ \beta[\sigma] &= \begin{cases} \sigma(\beta) & \text{if} \quad \beta \in \mathcal{D}(\sigma) \\ \beta & \text{otherwise} \end{cases} \\ \tau[\overline{T}][\sigma] &= \tau[\overline{T[\sigma]}] \\ (T_1 \to T_2)[\sigma] &= T_1[\sigma] \to T_2[\sigma] \\ (\textstyle\prod_i f_i\ T_i)[\sigma] &= \textstyle\prod_i f_i\ T_i[\sigma] \\ (T|r)[\sigma] &= T[\sigma]|r[\sigma] \end{aligned}$$

$$
\begin{aligned}
v[\sigma] &= v \\
o[\overline{T}][\sigma] &= o[\overline{T}[\sigma]] \\
(e_1\ e_2)[\sigma] &= e_1[\sigma]\ e_2[\sigma] \\
(\lambda v\!:\!T.\ e)[\sigma] &= \lambda v\!:\!T[\sigma].\ e[\sigma] \\
(e_1 \equiv e_2)[\sigma] &= e_1[\sigma] \equiv e_2[\sigma] \\
(\mathsf{if}\ e_0\ e_1\ e_2)[\sigma] &= \mathsf{if}\ e_0[\sigma]\ e_1[\sigma]\ e_2[\sigma] \\
\iota_T[\sigma] &= \iota_{T[\sigma]} \\
\left(\mathsf{proj}_{\prod_i f_i\ T_i}\ f_j\right)[\sigma] &= \mathsf{proj}_{(\prod_i f_i\ T_i)[\sigma]}\ f_j
\end{aligned}
$$

where of course $(T_1, \ldots, T_n)[\sigma] = T_1[\sigma], \ldots, T_n[\sigma]$. Given $\overline{\beta} \in \mathcal{N}^{(*)}$ and $\overline{T} \in \mathit{Type}^*$ such that $|\overline{\beta}| = |\overline{T}|$, we may write $T[\{\langle \beta_i, T_i\rangle \mid 1 \le i \le n\}]$ as just $T[\overline{\beta}/\overline{T}]$.

2.6.2 Expression substitutions

The function $\_[\_/\_] : \mathit{Exp} \times \mathcal{N} \times \mathit{Exp} \to \mathit{Exp}$ substitutes a variable u with an expression d in an expression e (written $e[u/d]$)

$$
\begin{aligned}
v[u/d] &= \begin{cases} d & \text{if} \quad u = v \\ v & \text{otherwise} \end{cases} \\
o[\overline{T}][u/d] &= o[\overline{T}] \\
(e_1\ e_2)[u/d] &= e_1[u/d]\ e_2[u/d] \\
(\lambda v\!:\!T.\ e)[u/d] &= \begin{cases} \lambda v\!:\!T.\ e & \text{if} \quad u = v \\ \lambda v\!:\!T.\ e[u/d] & \text{otherwise} \end{cases} \\
(e_1 \equiv e_2)[u/d] &= e_1[u/d] \equiv e_2[u/d] \\
(\mathsf{if}\ e_0\ e_1\ e_2)[u/d] &= \mathsf{if}\ e_0[u/d]\ e_1[u/d]\ e_2[u/d] \\
\iota_T[u/d] &= \iota_T \\
(\mathsf{proj}\ f)[u/d] &= \mathsf{proj}\ f
\end{aligned}
$$

No substitution is performed in the expressions contained in types contained in expressions because, as already mentioned, such inner expressions have no free variables in well-formed types.

The function $\mathcal{CV} : \mathit{Exp} \times \mathcal{N} \to \mathcal{P}_\omega(\mathcal{N})$ returns the variables that would be captured if a variable u were substituted with those variables in an expression e (i.e. all the variables bound in e at the free occurrences of u in e)

$$
\begin{aligned}
&\mathcal{CV}(v, u) &&= \emptyset \\
&\mathcal{CV}(o[\overline{T}], u) &&= \emptyset \\
&\mathcal{CV}(e_1\ e_2, u) &&= \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
&\mathcal{CV}(\lambda v\!:\!T.\ e, u) &&= \begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \text{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \text{otherwise} \end{cases} \\
&\mathcal{CV}(e_1 \equiv e_2, u) &&= \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
&\mathcal{CV}(\mathsf{if}\ e_0\ e_1\ e_2, u) &&= \mathcal{CV}(e_0, u) \cup \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
&\mathcal{CV}(\iota_T, u) &&= \emptyset \\
&\mathcal{CV}(\mathsf{proj}\ f, u) &&= \emptyset
\end{aligned}
$$

The relation $\mathit{OKsbs} \subseteq \mathit{Exp} \times \mathcal{N} \times \mathit{Exp}$ captures the condition that the substitution $e[u/d]$ causes no free variables in d to be captured

$$
\mathit{OKsbs}(e, u, d) \ \Leftrightarrow\ \mathcal{FV}(d) \cap \mathcal{CV}(e, u) = \emptyset
$$

3 Proof theory

The proof theory of the Metaslang logic includes not only rules to derive formulas (theorems), but also rules to derive typing judgements, type equivalences, and other judgements. The rules to derive such judgements are mutually recursive; even though they are presented separately in the following subsections, the rules are inductively defined all together.

3.1 Well-formed contexts

We define a unary relation $\vdash \_ : \textsc{context} \subseteq Cx$ to capture well-formed contexts as

$$\frac{}{\vdash \epsilon : \textsc{context}} \quad (\textsc{cxMT})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ \tau \notin \mathcal{TN}(cx)\end{array}}{\vdash cx, \mathsf{ty}\ \tau{:}n : \textsc{context}} \quad (\textsc{cxTdec})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ o \notin \mathcal{ON}(cx) \\ cx, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type}\end{array}}{\vdash cx, \mathsf{op}\ o{:}\{\overline{\beta}\}\ T : \textsc{context}} \quad (\textsc{cxOdec})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ \mathsf{ty}\ \tau{:}n \in cx \\ \mathsf{def}\ \tau \ldots \notin cx \\ cx, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type} \\ |\overline{\beta}| = n\end{array}}{\vdash cx, \mathsf{def}\ \tau[\overline{\beta}] = T : \textsc{context}} \quad (\textsc{cxTdef})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ cx, \mathsf{tvar}\ \overline{\beta} \vdash e : \mathsf{Bool}\end{array}}{\vdash cx, \mathsf{ax}\ \{\overline{\beta}\}\ e : \textsc{context}} \quad (\textsc{cxAx})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ cx, \mathsf{tvar}\ \overline{\beta} \vdash e\end{array}}{\vdash cx, \mathsf{lem}\ \{\overline{\beta}\}\ e : \textsc{context}} \quad (\textsc{cxLem})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ \beta \notin \mathcal{TV}(cx)\end{array}}{\vdash cx, \mathsf{tvar}\ \beta : \textsc{context}} \quad (\textsc{cxTVdec})$$

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ v \notin \mathcal{V}(cx) \\ cx \vdash T : \textsc{type}\end{array}}{\vdash cx, \mathsf{var}\ v{:}T : \textsc{context}} \quad (\textsc{cxVdec})$$

Eplanation:

- The empty context ϵ is well-formed. All other rules add context elements to well-formed contexts. Thus, a well-formed context is constructed incrementally starting with the empty one and adding suitable elements.
- A type declaration $\mathsf{ty}\ \tau{:}n$ can be added to cx if τ is not already declared in cx.
- An op declaration $\mathsf{op}\ o{:}\{\overline{\beta}\}\ T$ can be added to cx if o is not already declared in cx. The op's type T must be well-formed (defined later) in cx extended with the type variables $\overline{\beta}$, which must be distinct. Note that we do not require that all type variables in T are among $\overline{\beta}$, because there is no need for that restriction. However, since a well-formed spec has no type variable declarations, an op declaration in a well-formed spec automatically satisfies the restriction.

- A type definition $\mathsf{def}\ \tau[\overline{\beta}] = T$ can be added to cx if τ is already declared but not already defined in cx. The defining type T must be well-formed in cx extended with the type variables $\overline{\beta}$, which must be distinct and whose number must match the arity of τ. Similarly to op declarations, we do not require that all type variables in T are among $\overline{\beta}$, but such a restriction is automatically satisfied in a well-formed spec. Note that we allow vacuous type definitions such as $\mathsf{def}\ \tau[\epsilon] = \tau$ or $\mathsf{def}\ \tau[\epsilon] = \tau', \mathsf{def}\ \tau'[\epsilon] = \tau$, as well as other (mutually) recursive definitions that do not uniquely pin down the defined type such as the usual definition of lists; uniqueness can be enforced by suitable axioms (e.g. induction on lists). Unlike [1], there is no implicit assumption of (mutually) recursively defined types having least fixpoint semantics because there is no general way to generate implicit axioms expressing least fixpoint semantics in the Metaslang type system.

- A formula e can be added to cx as an axiom if e has type Bool (defined later). In general, the axiom may be polymorphic in (distinct) type variables $\overline{\beta}$. As in other cases, all type variables in e are automatically in $\overline{\beta}$ in well-formed specs, but that is not required in well-formed contexts in general.

- A formula e can be added to cx as a lemma if it is a theorem (defined later). In general, the lemma may be polymorphic in (distinct) type variables $\overline{\beta}$. As in other cases, all type variables in e are automatically in $\overline{\beta}$ in well-formed specs, but that is not required in well-formed contexts in general.

- A type variable declaration $\mathsf{tvar}\ \beta$ can be added to cx if β is not already declared in cx.

- A variable declaration $\mathsf{var}\ v\!:\!T$ can be added to cx if v is not already declared in cx and T is well-formed type in cx.

3.2 Well-formed specs

We define a unary relation $\vdash \_ : \text{SPEC} \subseteq \mathit{Sp}$ to capture well-formed specs as

$$\frac{\vdash sp : \text{CONTEXT}}{\vdash sp : \text{SPEC}} \quad (\text{SPEC})$$

Note that the unary relation $\vdash \_ : \text{SPEC}$ is defined on set Sp, which, as previously defined, consists of all the contexts without type variable and variable declarations. Thus, there is no need to include, as part of this rule, the condition that sp does not contain any type variable and variable declaration. The rule just says that a spec (which has no type variable or variable declarations by definition) is well-formed as a spec if it is well-formed as a context.

3.3 Well-formed types

We define a binary relation $\_ \vdash \_ : \text{TYPE} \subseteq \mathit{Cx} \times \mathit{Type}$ to capture well-formed types as

$$\frac{\vdash cx : \text{CONTEXT}}{cx \vdash \mathsf{Bool} : \text{TYPE}} \quad (\text{TYBOOL})$$

$$\frac{\begin{array}{c}\vdash cx : \text{CONTEXT} \\ \beta \in \mathcal{TV}(cx)\end{array}}{cx \vdash \beta : \text{TYPE}} \quad (\text{TYVAR})$$

$$\frac{\begin{array}{c}\vdash cx : \text{CONTEXT} \\ \mathsf{ty}\ \tau\!:\!n \in cx \\ |\overline{T}| = n \\ \forall i.\ \ cx \vdash T_i : \text{TYPE}\end{array}}{cx \vdash \tau[\overline{T}] : \text{TYPE}} \quad (\text{TYINST})$$

$$\frac{cx \vdash T_1 : \textsc{type} \quad cx \vdash T_2 : \textsc{type}}{cx \vdash T_1 \to T_2 : \textsc{type}} \quad (\textsc{tyArr})$$

$$\frac{\vdash cx : \textsc{context} \quad \forall i.\ \ cx \vdash T_i : \textsc{type}}{cx \vdash \prod_i f_i\ T_i : \textsc{type}} \quad (\textsc{tyRec})$$

$$\frac{cx \vdash r : T \to \mathsf{Bool} \quad \mathcal{FV}(r) = \emptyset}{cx \vdash T|r : \textsc{type}} \quad (\textsc{tyRestr})$$

Explanation:

- The type Bool is well-formed in any well-formed context.
- A type variable is a well-formed type in any well-formed context that declares it.
- A type instance $\tau[\overline{T}]$ is well-formed in any well-formed context cx that declares τ if the argument types are well-formed in cx and their number matches the arity of τ.
- The rules for arrow and record types are straightforward. Note that in tyArr we do not explicitly require cx to be well-formed because the fact that a type is well-formed in a context implies that the context is well-formed (as proved later). However, the condition is explicit in tyRec because a record type can have zero components.
- For restriction types, we require the predicate to have type $T \to \mathsf{Bool}$, which implies that T is a well-formed type (as proved later). We also require that r has no free variables.

3.4 Type equivalence

We define a ternary relation $\_ \vdash \_ \approx \_ \subseteq \mathit{Cx} \times \mathit{Type} \times \mathit{Type}$ to capture type equivalence as

$$\frac{\vdash cx : \textsc{context} \quad \mathsf{def}\ \tau[\overline{\beta}] = T \in cx \quad \forall i.\ \ cx \vdash T_i : \textsc{type}}{cx \vdash \tau[\overline{T}] \approx T[\overline{\beta}/\overline{T}]} \quad (\textsc{teDef})$$

$$\frac{cx \vdash T : \textsc{type}}{cx \vdash T \approx T} \quad (\textsc{teRefl})$$

$$\frac{cx \vdash T_1 \approx T_2}{cx \vdash T_2 \approx T_1} \quad (\textsc{teSymm})$$

$$\frac{cx \vdash T_1 \approx T_2 \quad cx \vdash T_2 \approx T_3}{cx \vdash T_1 \approx T_3} \quad (\textsc{teTrans})$$

$$\frac{cx \vdash \tau[\overline{T}] : \textsc{type} \quad \forall i.\ \ cx \vdash T_i \approx T_i'}{cx \vdash \tau[\overline{T}] \approx \tau[\overline{T}']} \quad (\textsc{teInst})$$

$$\frac{\begin{array}{c} cx \vdash T_1 \approx T_1' \\ cx \vdash T_2 \approx T_2' \end{array}}{cx \vdash T_1 \to T_2 \approx T_1' \to T_2'} \quad (\textsc{teArr})$$

$$\frac{\begin{array}{c} \vdash cx : \textsc{context} \\ \forall i.\;\; cx \vdash T_i \approx T_i' \end{array}}{cx \vdash \prod_i f_i\ T_i \approx \prod_i f_i\ T_i'} \quad (\textsc{teRec})$$

$$\frac{\begin{array}{c} cx \vdash T|r : \textsc{type} \\ cx \vdash T'|r' : \textsc{type} \\ cx \vdash T \approx T' \\ cx \vdash r \equiv r' \end{array}}{cx \vdash T|r \approx T'|r'} \quad (\textsc{teRestr})$$

$$\frac{\begin{array}{c} cx \vdash \prod_i f_i\ T_i : \textsc{type} \\ P : \{1,\ldots,n\} \hookrightarrow \{1,\ldots,n\} \end{array}}{cx \vdash \prod_i f_i\ T_i \approx \prod_i f_{P(i)}\ T_{P(i)}} \quad (\textsc{teRecOrd})$$

Explanation:

- A type definition introduces type equivalences, one for each instance of the defining equation.
- Type equivalence is indeed an equivalence, i.e. reflexive, symmetric, and transitive.
- Type equivalence is a congruence with respect to syntactic (meta-)operations to construct types in the Metaslang type system, namely type instantiations, arrow types, record types, and restriction types. In addition, equal restriction predicates give rise to equivalent restriction types. The premise $cx \vdash T'|r' : \textsc{type}$ in $\textsc{teRestr}$ may be redundant, i.e. derivable from the others, but this fact has not been established yet and so for now we have it as an explicit premise, which is needed to prove a theorem in §4.
- The order of the components of a record type is unimportant: any permutation of the components yields equivalent types. In the rule, the permutation is captured by a bijective function P on the field indices $\{1,\ldots,n\}$ (the rule explicitly says that P is injective only, but since domain and codomain are finite and equal, it follows that P is also surjective, hence bijective).

3.5 Subtyping

We define a quaternary relation $\_ \vdash \_ \prec_\_ \_ \subseteq \mathit{Cx} \times \mathit{Type} \times \mathit{Exp} \times \mathit{Type}$ to capture subtyping as

$$\frac{cx \vdash T|r : \textsc{type}}{cx \vdash T|r \prec_r T} \quad (\textsc{stRestr})$$

$$\frac{\begin{array}{c} cx \vdash T : \textsc{type} \\ r = \lambda v\!:\!T.\ \mathsf{true} \end{array}}{cx \vdash T \prec_r T} \quad (\textsc{stRefl})$$

$$\frac{\begin{array}{c} cx \vdash T : \textsc{type} \\ cx \vdash T_1 \prec_r T_2 \\ v \neq v' \\ r' = \lambda v\!:\!T \to T_2.\ \forall v'\!:\!T.\ r\ (v\ v') \end{array}}{cx \vdash T \to T_1 \prec_{r'} T \to T_2} \quad (\textsc{stArr})$$

$$\frac{\begin{array}{c} cx \vdash \prod_i f_i\ T_i : \textsc{type} \\ \forall i.\ \ cx \vdash T_i \prec_{r_i} T'_i \\ r = \lambda v\!:\!\prod_i f_i\ T'_i.\ \bigwedge_i r_i\ v.f_i \end{array}}{cx \vdash \prod_i f_i\ T_i \prec_r \prod_i f_i\ T'_i} \quad (\textsc{stRec})$$

$$\frac{\begin{array}{c} cx \vdash T_1 \prec_r T_2 \\ cx \vdash T_1 \approx T'_1 \\ cx \vdash T_2 \approx T'_2 \end{array}}{cx \vdash T'_1 \prec_r T'_2} \quad (\textsc{stTE})$$

Explanation:

- Unsurprisingly, a restriction type $T|r$ is a subtype of T, with r being the predicate over the supertype T that identifies the values that are also in the subtype $T|r$.
- Subtyping is reflexive, i.e. a (well-formed) type T is a subtype of itself and the associated subtype predicate is always true.
- Arrow types are monotone in their range types with respect to subtyping. The associated predicate holds when all the values of the function satisfy the predicate associated to the range subtype. Note that the domain must be the same; while domain contravariance is used in some type systems with subtypes, it would violate extensionality (see explanation in [3]).
- Record types are monotonic in their component types with respect to subtyping. The record subtype predicate holds when all the component subtype predicates hold on the record components.
- Rule stTE states the substitutivity of equivalent types in subtype judgements.

We do not need a transitivity rule for subtyping. As defined later, subtyping judgements are only used to assign types to expressions, e.g. to assign a supertype to an expression of a subtype. So, instead of using transitivity of subtyping, rules for well-typed expressions can be applied multiple times, achieving the same effect.

3.6 Well-typed expressions

We define a ternary relation $\_ \vdash \_ : \_ \subseteq Cx \times Exp \times Type$ to capture well-typed expressions as

$$\frac{\begin{array}{c} \vdash cx : \textsc{context} \\ \mathsf{var}\ v\!:\!T \in cx \end{array}}{cx \vdash v : T} \quad (\textsc{exVar})$$

$$\frac{\begin{array}{c} \vdash cx : \textsc{context} \\ \mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T \in cx \\ \forall i.\ \ cx \vdash T_i : \textsc{type} \end{array}}{cx \vdash o[\overline{T}] : T[\overline{\beta}/\overline{T}]} \quad (\textsc{exOp})$$

$$\frac{\begin{array}{c} cx \vdash e_1 : T_1 \to T_2 \\ cx \vdash e_2 : T_1 \end{array}}{cx \vdash e_1\ e_2 : T_2} \quad (\textsc{exApp})$$

$$\frac{\begin{array}{c} cx, \mathsf{var}\ v\!:\!T \vdash e : T' \\ cx \vdash T' : \textsc{type} \end{array}}{cx \vdash \lambda v\!:\!T.\ e : T \to T'} \quad (\textsc{exAbs})$$

$$\frac{\begin{array}{c} cx \vdash e_1 : T \\ cx \vdash e_2 : T \end{array}}{cx \vdash e_1 \equiv e_2 : \mathsf{Bool}} \quad (\textsc{exEq})$$

$$\frac{\begin{array}{c} cx \vdash e_0 : \mathsf{Bool} \\ cx, \mathsf{ax}\ e_0 \vdash e_1 : T \\ cx, \mathsf{ax}\ \neg\ e_0 \vdash e_2 : T \\ cx \vdash T : \textsc{type} \end{array}}{cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 : T} \quad (\textsc{exIf})$$

$$\frac{\begin{array}{c} cx \vdash e : \mathsf{Bool} \\ cx \vdash e_1 : T \\ cx \vdash e_2 : T \end{array}}{cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 : T} \quad (\textsc{exIf0})$$

$$\frac{cx \vdash T : \textsc{type}}{cx \vdash \iota_T : (T \to \mathsf{Bool})|\exists!_T \to T} \quad (\textsc{exThe})$$

$$\frac{cx \vdash \prod_i f_i\ T_i : \textsc{type}}{cx \vdash \mathsf{proj}\ f_j : \prod_i f_i\ T_i \to T_j} \quad (\textsc{exProj})$$

$$\frac{\begin{array}{c} cx \vdash e : T \\ cx \vdash T \prec_r T' \end{array}}{cx \vdash e : T'} \quad (\textsc{exSuper})$$

$$\frac{\begin{array}{c} cx \vdash e : T' \\ cx \vdash T \prec_r T' \\ cx \vdash r\ e \end{array}}{cx \vdash e : T} \quad (\textsc{exSub})$$

$$\frac{\begin{array}{c} cx \vdash \lambda v{:}T.\ e : T' \\ v' \notin \mathcal{FV}(e) \cup \mathcal{CV}(e, v) \end{array}}{cx \vdash \lambda v'{:}T.\ e[v/v'] : T'} \quad (\textsc{exAbsAlpha})$$

Explanation:

- A variable v declared in a well-formed context is a well-typed expression with the type T given in its declaration.
- An op o declared in a well-formed context can be instantiated via well-formed types $\overline{T}$ whose number matches the number of type variables $\overline{\beta}$. The result is a well-formed expression whose type is obtained by substituting $\overline{\beta}$ with $\overline{T}$ in the declared type T of o.
- An application is well-typed if the function has an arrow type and the argument has the domain type of the arrow type. The type of the application is the range type of the arrow type.
- An abstraction is well-typed if the body is well-typed in the context extended with the declaration of the variable bound by the abstraction. The type of the abstraction is the arrow type that has the type of the variable as domain and the type of the body as range. The premise $cx \vdash T' : \textsc{type}$ of exAbs is needed to prove a theorem in §4.

- An equality is well-typed with type Bool if the left- and right-hand sides are both well-typed with a common type T.
- In the rule exIf for conditionals, the two branches must be well-typed, with a common type, in the context where the condition holds and does not hold, respectively. The additional assumption about the condition holding or not is realized by adding an axiom to the context. This rule makes conditionals non-strict. There is also a stronger rule exIf0 that does not add any assumption about the condition to the context. The reason for rule exIf0 and for why it does not seem derivable from exIf, is given in §4. The premise $cx \vdash T : \textsc{type}$ in exIf may be redundant, i.e. derivable from the others, but this fact has not been established yet and so for now we have it as an explicit premise, which is needed to prove a theorem in §4.
- The description operator for a well-formed type is well-typed and denotes a function from the predicates over the type that are satisfied by a unique value to the type itself.
- A projector for a well-formed record type is well-typed and denotes a function from the record type to the corresponding component type.
- Rules exSuper and exSub link the notion of well-typed expressions to the notion of subtyping. If an expression e has a subtype T, it also has any supertype T'. If an expression e has a supertype T', it also has any subtype T such that the associated predicate holds on e. Note that rule exSuper, in conjunction with rule stRefl, can be used to show that if an expression e has type T, it also has any type T' equivalent to T.
- The last rule amounts to treating expressions up to alpha equivalence, allowing to rename bound variables maintaining well-typedness. Without this rule, the Metaslang logic would be non-monotone, because extending a context with variable declarations may invalidate conclusions about the well-typedness of expressions that bind those variables (e.g. if $cx \vdash \lambda v\!:\!T.\ e : T'$ were provable then $cx, \mathsf{var}\ v\!:\!T \vdash \lambda v\!:\!T.\ e : T'$ would not be provable). This rule also allows variable hiding as described in [1].

3.7 Theorems

We define a binary relation $\_ \vdash \_ \subseteq Cx \times Exp$ to capture theorems as

$$\frac{\begin{array}{c}\vdash cx : \textsc{context} \\ \mathsf{ax}\ \{\overline{\beta}\}\ e \in cx \\ \forall i.\ \ cx \vdash T_i : \textsc{type}\end{array}}{cx \vdash e[\overline{\beta}/\overline{T}]} \quad (\textsc{thAx})$$

$$\frac{cx \vdash e : \ldots}{cx \vdash e \equiv e} \quad (\textsc{thRefl})$$

$$\frac{cx \vdash e_1 \equiv e_2}{cx \vdash e_2 \equiv e_1} \quad (\textsc{thSymm})$$

$$\frac{\begin{array}{c}cx \vdash e_1 \equiv e_2 \\ cx \vdash e_2 \equiv e_3\end{array}}{cx \vdash e_1 \equiv e_3} \quad (\textsc{thTrans})$$

$$\frac{\begin{array}{c}cx \vdash o[\overline{T}] : \ldots \\ \forall i.\ \ cx \vdash T_i \approx T'_i\end{array}}{cx \vdash o[\overline{T}] \equiv o[\overline{T}']} \quad (\textsc{thOpSubst})$$

$$\frac{\begin{array}{c} cx \vdash e_1\ e_2 : \ldots \\ cx \vdash e_1 \equiv e'_1 \\ cx \vdash e_2 \equiv e'_2 \end{array}}{cx \vdash e_1\ e_2 \equiv e'_1\ e'_2} \quad (\textsc{thAppSubst})$$

$$\frac{\begin{array}{c} cx \vdash \lambda v{:}T.\ e : \ldots \\ cx \vdash T \approx T' \end{array}}{cx \vdash \lambda v{:}T.\ e \equiv \lambda v{:}T'.\ e} \quad (\textsc{thAbsSubst})$$

$$\frac{\begin{array}{c} cx \vdash e_1 \equiv e_2 : \ldots \\ cx \vdash e_1 \equiv e'_1 \\ cx \vdash e_2 \equiv e'_2 \end{array}}{cx \vdash (e_1 \equiv e_2) \equiv (e'_1 \equiv e'_2)} \quad (\textsc{thEqSubst})$$

$$\frac{\begin{array}{c} cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 : \ldots \\ cx \vdash e_0 \equiv e'_0 \\ cx, \mathsf{ax}\ e_0 \vdash e_1 \equiv e'_1 \\ cx, \mathsf{ax}\ \neg\ e_0 \vdash e_2 \equiv e'_2 \end{array}}{cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 \equiv \mathsf{if}\ e'_0\ e'_1\ e'_2} \quad (\textsc{thIfSubst})$$

$$\frac{\begin{array}{c} cx \vdash \iota_T : \ldots \\ cx \vdash T \approx T' \end{array}}{cx \vdash \iota_T \equiv \iota_{T'}} \quad (\textsc{thTheSubst})$$

$$\frac{\begin{array}{c} cx \vdash \mathsf{proj}_{\prod_i f_i\ T_i}\ f : \ldots \\ cx \vdash \prod_i f_i\ T_i \approx \prod_i f'_i\ T'_i \end{array}}{cx \vdash \mathsf{proj}_{\prod_i f_i\ T_i}\ f \equiv \mathsf{proj}_{\prod_i f'_i\ T'_i}\ f} \quad (\textsc{thProjSubst})$$

$$\frac{\begin{array}{c} cx \vdash e \\ cx \vdash e \equiv e' \end{array}}{cx \vdash e'} \quad (\textsc{thSubst})$$

$$\frac{\begin{array}{c} \vdash cx : \textsc{context} \\ v \neq v' \end{array}}{cx \vdash \forall v{:}\mathsf{Bool} \rightarrow \mathsf{Bool}.\ (v\ \mathsf{true} \wedge v\ \mathsf{false} \Leftrightarrow (\forall v'{:}\mathsf{Bool}.\ v\ v'))} \quad (\textsc{thBool})$$

$$\frac{\begin{array}{c} cx \vdash T \rightarrow T' : \textsc{type} \\ v \neq v' \ \wedge\ v' \neq v'' \ \wedge\ v'' \neq v \end{array}}{cx \vdash \forall v{:}T \rightarrow T', v'{:}T \rightarrow T'.\ (v \equiv v' \Leftrightarrow (\forall v''{:}T.\ v\ v'' \equiv v'\ v''))} \quad (\textsc{thExt})$$

$$\frac{\begin{array}{c} cx \vdash (\lambda v{:}T.\ e)\ e' : \ldots \\ OKsbs(e, v, e') \end{array}}{cx \vdash (\lambda v{:}T.\ e)\ e' \equiv e[v/e']} \quad (\textsc{thAbs})$$

$$\frac{\begin{array}{c} cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 : \ldots \\ cx, \mathsf{ax}\ e_0 \vdash e_1 \equiv e \\ cx, \mathsf{ax}\ \neg\ e_0 \vdash e_2 \equiv e \end{array}}{cx \vdash \mathsf{if}\ e_0\ e_1\ e_2 \equiv e} \quad (\textsc{thIf})$$

$$\frac{cx \vdash \iota_T\ e : T}{cx \vdash e\ (\iota_T\ e)} \quad (\textsc{thThe})$$

$$\frac{\begin{array}{c} cx \vdash \prod_i f_i\ T_i : \textsc{type} \\ v \neq v' \end{array}}{cx \vdash \forall v : \prod_i f_i\ T_i, v' : \prod_i f_i\ T_i.\ ((\bigwedge_i v.f_i \equiv v'.f_i) \Rightarrow v \equiv v')} \quad (\textsc{thRec})$$

$$\frac{cx \vdash \prod_i f_i\ T_i \prec_r \prod_i f_i\ T_i'}{cx \vdash \forall v : \prod_i f_i\ T_i.\ \mathsf{proj}_{\prod_i f_i\ T_i}\ f_j\ v \equiv \mathsf{proj}_{\prod_i f_i\ T_i'}\ f_j\ v} \quad (\textsc{thProjSub})$$

$$\frac{\begin{array}{c} cx \vdash T \prec_r T' \\ cx \vdash e : T \end{array}}{cx \vdash r\ e} \quad (\textsc{thSub})$$

Explanation:

- Axioms in a well-formed context are readily instantiated into theorems, via rule thAx. More precisely, the type variables over which the axiom is polymorphic are replaced with well-formed types.
- Rules thRefl, thSymm, and thTrans say that equality is an equivalence, i.e. reflexive, symmetric, and transitive.
- Rules thOpSubst to thProjSubst state the substitutivity of equalities and type equivalences in expressions. Note that in rule thIfSubst the context is extended with an axiom saying that the condition is true or false.
- Rule thSubst says that anything equal to a theorem is itself a theorem.
- Rule thBool asserts that true and false are the only values of type Bool.
- Rule thExt says that a function is characterized by its values over all the values of its domain, i.e. by extensionality.
- Rule thAbs defines the semantics of lambda abstraction: the bound variable v is replaced with the argument e' in the body e. The premise of the rule says that the application is well-typed.
- Rule thIf defines the semantics of conditionals: a conditional equals an expression if both branches do, in the contexts extended with the assumption that the condition is true and false, respectively. Note the premise that requires the conditional to be well-typed.
- Rule thThe says that a description satisfies the predicate associated to the description.
- Rule thRec says that a record is characterized by the values of its components.
- Rule thProjSub says that projectors for record subtypes agree with projectors for record supertypes.
- Rule thSub says that every value of a subtype satisfies the predicate that characterizes the subtype with respect to a supertype.

3.8 Proofs

The previous subsections have defined judgements of the forms

$$\begin{array}{l} \vdash cx : \textsc{context} \\ \vdash sp : \textsc{spec} \\ cx \vdash T : \textsc{type} \\ cx \vdash T_1 \approx T_2 \\ cx \vdash T_1 \prec_r T_2 \\ cx \vdash e : T \\ cx \vdash e \end{array}$$

by means of a set of inductive rules.

A proof of a judgement is a finite sequence of judgements that ends with the proved judgement and where each judgement in the sequence is derived from preceding judgements using some rule.

[[[TO DO: Make sure that the rules for theorems are "sufficient", i.e. all truths "of interest" are indeed theorems derivable from the rules. Even though higher-order logic is notoriously incomplete, in practice theorem provers like PVS and HOL are sufficient to prove desired properties of formalized concepts without running into theoretical limitations. Perhaps the requirement boils down to prove completeness with respect to so-called "general models" (cf. [2]).]]]

4 Properties

4.1 Additional notions

In order to prove certain properties of the syntax and proof theory formalized in §2 and §3, we introduce some additional notions. Those notions have not been introduced earlier because they are not needed to define the Metaslang logic, but only to express and prove properties about it.

We extend expression substitution to context elements and contexts

$$\begin{aligned} (\mathsf{ty}\ \tau : n)[u/d] &= \mathsf{ty}\ \tau : n \\ (\mathsf{op}\ o : \{\overline{\beta}\}\ T)[u/d] &= \mathsf{op}\ o : \{\overline{\beta}\}\ T \\ (\mathsf{def}\ \tau[\overline{\beta}] = T)[u/d] &= \mathsf{def}\ \tau[\overline{\beta}] = T \\ (\mathsf{ax}\ \{\overline{\beta}\}\ e)[u/d] &= \mathsf{ax}\ \{\overline{\beta}\}\ e[u/d] \\ (\mathsf{lem}\ \{\overline{\beta}\}\ e)[u/d] &= \mathsf{lem}\ \{\overline{\beta}\}\ e[u/d] \\ (\mathsf{tvar}\ \beta)[u/d] &= \mathsf{tvar}\ \beta \\ (\mathsf{var}\ v : T)[u/d] &= \mathsf{var}\ v : T \end{aligned}$$

$$\begin{aligned} \epsilon[u/d] &= \epsilon \\ (cx_1, cx_2)[u/d] &= cx_1[u/d], cx_2[u/d] \end{aligned}$$

Note that $(\mathsf{var}\ v : T)[u/d] = \mathsf{var}\ v : T$ even if $u = v$, because a variable declaration "works" more or less like a binder (and the v in $\lambda v : T.\ e$ is unchanged under substitution).

The function $\mathcal{FTV} : \mathit{Type} + \mathit{Exp} + \mathit{CxElem} + \mathit{Cx} \to \mathcal{P}_\omega(\mathcal{N})$ returns the free type variables of a type, expression, or context (element)

$$\begin{aligned} &\mathcal{FTV}(\mathsf{Bool}) &&= \emptyset \\ &\mathcal{FTV}(\beta) &&= \{\beta\} \\ &\mathcal{FTV}(\tau[\overline{T}]) &&= \textstyle\bigcup_i \mathcal{FTV}(T_i) \\ &\mathcal{FTV}(T_1 \to T_2) &&= \mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2) \\ &\mathcal{FTV}(\textstyle\prod_i f_i\ T_i) &&= \textstyle\bigcup_i \mathcal{FTV}(T_i) \\ &\mathcal{FTV}(T|r) &&= \mathcal{FTV}(T) \cup \mathcal{FTV}(r) \end{aligned}$$

$$\begin{array}{ll}
\mathcal{FTV}(v) & = \emptyset \\
\mathcal{FTV}(o[\overline{T}]) & = \bigcup_i \mathcal{FTV}(T_i) \\
\mathcal{FTV}(e_1\ e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(\lambda v{:}T.\ e) & = \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \\
\mathcal{FTV}(e_1 \equiv e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(\mathsf{if}\ e_0\ e_1\ e_2) & = \mathcal{FTV}(e_0) \cup \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(\iota_T) & = \mathcal{FTV}(T) \\
\mathcal{FTV}(\mathsf{proj}_{\prod_i f_i\ T_i}\ f_j) & = \mathcal{FTV}(\prod_i f_i\ T_i)
\end{array}$$

$$\begin{array}{ll}
\mathcal{FTV}(\mathsf{ty}\ \tau{:}n) & = \emptyset \\
\mathcal{FTV}(\mathsf{op}\ o{:}\{\overline{\beta}\}\ T) & = \mathcal{FTV}(T) - \overline{\beta} \\
\mathcal{FTV}(\mathsf{def}\ \tau[\overline{\beta}] = T) & = \mathcal{FTV}(T) - \overline{\beta} \\
\mathcal{FTV}(\mathsf{ax}\ \{\overline{\beta}\}\ e) & = \mathcal{FTV}(e) - \overline{\beta} \\
\mathcal{FTV}(\mathsf{lem}\ \{\overline{\beta}\}\ e) & = \mathcal{FTV}(e) - \overline{\beta} \\
\mathcal{FTV}(\mathsf{tvar}\ \beta) & = \emptyset \\
\mathcal{FTV}(\mathsf{var}\ v{:}T) & = \mathcal{FTV}(T)
\end{array}$$

$$\begin{array}{ll}
\mathcal{FTV}(\epsilon) & = \emptyset \\
\mathcal{FTV}(cx_1, cx_2) & = \mathcal{FTV}(cx_1) \cup \mathcal{FTV}(cx_2)
\end{array}$$

Note that the type variables that occur in a type or expression are always free; the only bound type variables are the $\overline{\beta}$ in context elements.

We extend type substitution to context elements, contexts, and type substitution themselves

$$\begin{array}{rl}
(\mathsf{ty}\ \tau{:}n)[\sigma] & = \mathsf{ty}\ \tau{:}n \\
(\mathsf{op}\ o{:}\{\overline{\beta}\}\ T)[\sigma] & = \mathsf{op}\ o{:}\{\overline{\beta}\}\ T[\sigma{\uparrow}_{\overline{\beta}}] \\
(\mathsf{def}\ \tau[\overline{\beta}] = T)[\sigma] & = \mathsf{def}\ \tau[\overline{\beta}] = T[\sigma{\uparrow}_{\overline{\beta}}] \\
(\mathsf{ax}\ \{\overline{\beta}\}\ e)[\sigma] & = \mathsf{ax}\ \{\overline{\beta}\}\ e[\sigma{\uparrow}_{\overline{\beta}}] \\
(\mathsf{lem}\ \{\overline{\beta}\}\ e)[\sigma] & = \mathsf{lem}\ \{\overline{\beta}\}\ e[\sigma{\uparrow}_{\overline{\beta}}] \\
(\mathsf{tvar}\ \beta)[\sigma] & = \mathsf{tvar}\ \beta \\
(\mathsf{var}\ v{:}T)[\sigma] & = \mathsf{var}\ v{:}T[\sigma]
\end{array}$$

$$\begin{array}{rl}
\epsilon[\sigma] & = \epsilon \\
(cx_1, cx_2)[\sigma] & = cx_1[\sigma], cx_2[\sigma]
\end{array}$$

$$\sigma[\sigma'] = \{\langle \beta, \sigma(\beta)[\sigma'] \rangle \mid \beta \in \mathcal{D}(\sigma)\}$$

The function $\mathcal{CTV} : (\mathit{CxElem} + \mathit{Cx}) \times \mathcal{N} \to \mathcal{P}_\omega(\mathcal{N})$ returns the type variables that would be captured if a type variable α were substituted with those type variables in a context (element), i.e. all the variables bound in the (context) element at the free occurrences of α in the context (element)

$$\begin{array}{ll}
\mathcal{CTV}(\mathsf{ty}\ \tau{:}n, \alpha) & = \emptyset \\
\mathcal{CTV}(\mathsf{op}\ o{:}\{\overline{\beta}\}\ T, \alpha) & = \begin{cases} \overline{\beta} & \text{if} \quad \alpha \in \mathcal{FTV}(T) - \overline{\beta} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{CTV}(\mathsf{def}\ \tau[\overline{\beta}] = T, \alpha) & = \begin{cases} \overline{\beta} & \text{if} \quad \alpha \in \mathcal{FTV}(T) - \overline{\beta} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{CTV}(\mathsf{ax}\ \{\overline{\beta}\}\ e, \alpha) & = \begin{cases} \overline{\beta} & \text{if} \quad \alpha \in \mathcal{FTV}(e) - \overline{\beta} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{CTV}(\mathsf{lem}\ \{\overline{\beta}\}\ e, \alpha) & = \begin{cases} \overline{\beta} & \text{if} \quad \alpha \in \mathcal{FTV}(e) - \overline{\beta} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathcal{CTV}(\mathsf{tvar}\ \beta, \alpha) & = \emptyset \\
\mathcal{CTV}(\mathsf{var}\ v{:}T, \alpha) & = \emptyset
\end{array}$$

$$\begin{array}{ll}
\mathcal{CTV}(\epsilon, \alpha) & = \emptyset \\
\mathcal{CTV}((cx_1, cx_2), \alpha) & = \mathcal{CTV}(cx_1, \alpha) \cup \mathcal{CTV}(cx_2, \alpha)
\end{array}$$

The relation $\mathit{OKsbs} \subseteq \mathit{Cx} \times (\mathcal{N} \xrightarrow{\mathrm{p}} \mathit{Type})$ captures the condition that the substitution $cx[\sigma]$ causes no free type variables in cx to be captured and does not substitute any type variable declared in cx

$$\mathit{OKsbs}(cx, \sigma) \;\Leftrightarrow\; (\forall \beta \in \mathcal{D}(\sigma).\;\; \mathcal{FTV}(\sigma(\beta)) \cap \mathcal{CTV}(cx, \beta) = \emptyset) \;\wedge\; \mathcal{D}(\sigma) \cap \mathcal{TV}(cx) = \emptyset$$

Given $\overline{\beta} \in \mathcal{N}^{(*)}$ and $\overline{T} \in \mathit{Type}^*$ such that $|\overline{\beta}| = |\overline{T}|$, we may write $\mathit{OKsbs}(cx, \{\langle \beta_i, T_i \rangle \mid 1 \leq i \leq n\})$ as just $\mathit{OKsbs}(cx, \overline{\beta}, \overline{T})$.

4.2 Syntactic properties

Expression substitution only takes place at the free occurrences of the variable that is substituted. Thus, if the variable is not free in the expression, substitution causes no change and no variable capture:

Theorem 4.1

$$u \notin \mathcal{FV}(e) \;\Rightarrow\; \mathcal{CV}(e, u) = \emptyset \;\wedge\; \mathit{OKsbs}(e, u, d) \;\wedge\; e[u/d] = e$$

Since expression substitution only substitutes the free occurrences of the variable, the variable itself is not captured at those occurrences:

Theorem 4.2

$$u \notin \mathcal{CV}(e, u)$$

It is always possible to substitute a variable with itself and the substitution causes no change:

Theorem 4.3

$$\mathit{OKsbs}(e, u, u) \;\wedge\; e[u/u] = e$$

The following four theorems say that if a substitution causes no capture in a compound expression, it does not cause capture in the component expressions either:

Theorem 4.4

$$\mathit{OKsbs}(e_1\ e_2, u, d) \;\Rightarrow\; \mathit{OKsbs}(e_1, u, d) \;\wedge\; \mathit{OKsbs}(e_2, u, d)$$

Theorem 4.5

$$\mathit{OKsbs}(e_1 \equiv e_2, u, d) \;\Rightarrow\; \mathit{OKsbs}(e_1, u, d) \;\wedge\; \mathit{OKsbs}(e_2, u, d)$$

Theorem 4.6

$$\mathit{OKsbs}(\mathsf{if}\ e_0\ e_1\ e_2, u, d) \;\Rightarrow\; \mathit{OKsbs}(e_0, u, d) \;\wedge\; \mathit{OKsbs}(e_1, u, d) \;\wedge\; \mathit{OKsbs}(e_2, u, d)$$

Theorem 4.7

$$\mathit{OKsbs}(\lambda v\!:\!T.\ e, u, d) \;\wedge\; u \neq v \;\Rightarrow\; \mathit{OKsbs}(e, u, d)$$

In the last theorem, the condition $u \neq v$ is used in the proof and is in fact necessary, because in general $\mathit{OKsbs}(\lambda v\!:\!T.\ e, u, d) \;\not\Rightarrow\; \mathit{OKsbs}(e, u, d)$, as shown by the counter-example $u = v$, $e = \lambda w\!:\!T.\ v$, and $d = w$.

If we substitute u with d in e, we remove u from the free variables of e and add the free variables of d. This is actually true only if u is free in e (otherwise the substitution causes no change) and no variable is captured (otherwise the captured variables of d would not contribute to the free variables of the result of substitution):

Theorem 4.8

$$u \in \mathcal{FV}(e) \;\wedge\; \mathit{OKsbs}(e, u, d) \;\Rightarrow\; \mathcal{FV}(e[u/d]) = (\mathcal{FV}(e) - \{u\}) \cup \mathcal{FV}(d)$$

If we substitute u with an expression d that does not have u among its free variables, the resulting expression does not have u among its free variables (provided that the substitution captured no variables):

Theorem 4.9

$$OKsbs(e, u, d) \ \wedge \ u \notin \mathcal{FV}(d) \ \Rightarrow \ u \notin \mathcal{FV}(e[u/d])$$

Two expression substitutions commute if their variables do not "interact":

Theorem 4.10

$$u \neq u' \ \wedge \ u \notin \mathcal{FV}(d') \ \wedge \ u' \notin \mathcal{FV}(d) \ \Rightarrow \ e[u/d][u'/d'] = e[u'/d'][u/d]$$

If we substitute first u with d and then u with d', we obtain the same result as directly substituting u with $d[u/d']$:

Theorem 4.11

$$e[u/d][u/d'] = e[u/d[u/d']]$$

Variable renaming is idempotent:

Theorem 4.12

$$e[u/u'][u/u'] = e[u/u']$$

Substituting w with d (such that no capture takes place) and then renaming u to u' is equivalent to substituting w with the renaming applied to d (which renames the free occurrences of u in d) and then applying the renaming to the result (which renames the free occurrences of u in the original expression e):

Theorem 4.13

$$OKsbs(e, w, d) \ \Rightarrow \ e[w/d][u/u'] = e[w/d[u/u']][u/u']$$

In an expression e, renaming u to u' and then back u' to u leaves e unchanged, provided that u' is not captured (otherwise the captured occurrences would not be renamed back to u) and does not already occur free in e (otherwise we do not obtain e at the end, because all free occurrences of u' disappear when we rename u' to u):

Theorem 4.14

$$OKsbs(e, u, u') \ \wedge \ u' \notin \mathcal{FV}(e) \ \Rightarrow \ e[u/u'][u'/u] = e$$

If we rename a variable u to u' in an expression then the variables captured at the free occurrences of u in the original expression coincide with those captured at the free occurrences of u' in the transformed expression (provided that u' is not captured in the renaming and that u' was not already free in the original expression):

Theorem 4.15

$$OKsbs(e, u, u') \ \wedge \ u' \notin \mathcal{FV}(e) \ \Rightarrow \ \mathcal{CV}(e, u) = \mathcal{CV}(e[u/u'], u')$$

If we rename a variable u to u' in an expression then the variables captured at the free occurrences of a third variable w in the original expression coincide with those captured at the free occurrences of w in the transformed expression (provided that u' is not captured in the renaming):

Theorem 4.16

$$OKsbs(e, u, u') \ \wedge \ w \neq u \ \wedge \ w \neq u' \ \Rightarrow \ \mathcal{CV}(e, w) = \mathcal{CV}(e[u/u'], w)$$

Renaming bound variables does not change free variables, provided that the renaming causes no capture (not only the substitution $e[v/v']$ must cause no capture, but also v' must not occur free in e, otherwise it would be captured by the top-level $\lambda v' \!:\! T.\ \ldots$):

Theorem 4.17

$$v' \notin \mathcal{FV}(e) \cup \mathcal{CV}(e, v) \ \Rightarrow\ \mathcal{FV}(\lambda v \!:\! T.\ e) = \mathcal{FV}(\lambda v' \!:\! T.\ e[v/v'])$$

Type substitution does not change free and captured variables:

Theorem 4.18

$$\mathcal{FV}(e[\sigma]) = \mathcal{FV}(e)$$

Theorem 4.19

$$\mathcal{CV}(e[\sigma], u) = \mathcal{CV}(e, u)$$

If we substitute u with d in an e, we add the free type variables of d to those of e if u is free in e (otherwise we add no type variables):

Theorem 4.20

$$\mathcal{FTV}(e[u/d]) = \mathcal{FTV}(e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e) \\ \emptyset & \text{otherwise} \end{cases}$$

Variable renaming does not change free type variables:

Theorem 4.21

$$\mathcal{FTV}(e[u/u']) = \mathcal{FTV}(e)$$

Applying a type substitution σ to the result of a variable substitution $e[u/d]$ is like applying σ to e and d and then performing the variable substitution:

Theorem 4.22

$$e[u/d][\sigma] = e[\sigma][u/d[\sigma]]$$

Variable renaming commutes with type substitution:

Theorem 4.23

$$e[u/u'][\sigma] = e[\sigma][u/u']$$

Expression substitution does not change the type, op, and (type) variable declarations in a context, does not change the type definitions in a context, and applies to the axioms and lemmas in a context:

Theorem 4.24

$$\begin{array}{rcl}
\mathcal{TN}(cx[u/d]) & = & \mathcal{TN}(cx) \\
\mathcal{ON}(cx[u/d]) & = & \mathcal{ON}(cx) \\
\mathcal{TV}(cx[u/d]) & = & \mathcal{TV}(cx) \\
\mathcal{V}(cx[u/d]) & = & \mathcal{V}(cx) \\
\mathsf{ty}\ \tau \!:\! n \in cx & \Leftrightarrow & \mathsf{ty}\ \tau \!:\! n \in cx[u/d] \\
\mathsf{op}\ o \!:\! \{\overline{\beta}\}\ T \in cx & \Leftrightarrow & \mathsf{op}\ o \!:\! \{\overline{\beta}\}\ T \in cx[u/d] \\
\mathsf{def}\ \tau[\overline{\beta}] = T \in cx & \Leftrightarrow & \mathsf{def}\ \tau[\overline{\beta}] = T \in cx[u/d] \\
\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx & \Rightarrow & \mathsf{ax}\ \{\overline{\beta}\}\ e[u/d] \in cx[u/d] \\
\mathsf{lem}\ \{\overline{\beta}\}\ e \in cx & \Rightarrow & \mathsf{lem}\ \{\overline{\beta}\}\ e[u/d] \in cx[u/d] \\
\mathsf{tvar}\ \beta \in cx & \Leftrightarrow & \mathsf{tvar}\ \beta \in cx[u/d] \\
\mathsf{var}\ v \!:\! T \in cx & \Leftrightarrow & \mathsf{var}\ v \!:\! T \in cx[u/d]
\end{array}$$

An empty type substitution causes no change:

Theorem 4.25

$$x \in \mathit{Type} + \mathit{Exp} + \mathit{CxElem} + \mathit{Cx} \quad \Rightarrow \quad x[\emptyset] = x$$

In a type substitution, only the type variables that are free in the type, expression, or context (element) matter (i.e. we can eliminate the other type variables from the domain of the substitution):

Theorem 4.26

$$x \in \mathit{Type} + \mathit{Exp} + \mathit{CxElem} + \mathit{Cx} \quad \Rightarrow \quad x[\sigma] = x[\sigma\downarrow_{\mathcal{FTV}(x)}]$$

If none of the type variables substituted by a type substitution appears in the type, expression, or context (element) to which the type substitution is applied, then the type, expression, or context (element) is unchanged:

Theorem 4.27

$$x \in \mathit{Type} + \mathit{Exp} + \mathit{CxElem} + \mathit{Cx} \quad \wedge \quad \mathcal{D}(\sigma) \cap \mathcal{FTV}(x) = \emptyset \quad \Rightarrow \quad x[\sigma] = x$$

If we apply a type substitution σ to a type or expression x, the free type variables that are substituted are replaced by the free type variables of the types that replace those type variables:

Theorem 4.28

$$x \in \mathit{Type} + \mathit{Exp} \quad \Rightarrow \quad \mathcal{FTV}(x[\sigma]) \quad = \quad (\mathcal{FTV}(x) - \mathcal{D}(\sigma)) \quad \cup \bigcup_{\alpha \in \mathcal{FTV}(x) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$$

Applying two type substitutions σ and σ' one after the other to a type or expression is like applying $\sigma[\sigma']$, provided that σ' does not substitute type variables that are free in the type or expression but are not in the domain of σ:

Theorem 4.29

$$x \in \mathit{Type} + \mathit{Exp} \quad \wedge \quad (\mathcal{FTV}(x) - \mathcal{D}(\sigma)) \cap \mathcal{D}(\sigma') = \emptyset \quad \Rightarrow \quad x[\sigma][\sigma'] = x[\sigma[\sigma']]$$

Applying a type substitutions σ followed by another type substitution σ' is like applying σ' followed by $\sigma[\sigma']$, under certain conditions:

Theorem 4.30

$$\begin{array}{l}
x \in \mathit{Type} + \mathit{Exp} \\
\mathcal{D}(\sigma) \cap \mathcal{D}(\sigma') = \emptyset \\
(\forall \alpha \in \mathcal{FTV}(x) \cap \mathcal{D}(\sigma').\ \ \mathcal{FTV}(\sigma'(\alpha)) \cap \mathcal{D}(\sigma) = \emptyset) \\
\quad \Rightarrow \\
x[\sigma][\sigma'] = x[\sigma'][\sigma[\sigma']]
\end{array}$$

4.3 Properties of abbreviations

The following five theorems show that $\mathcal{FV}$, $\_[\_/\_]$, $\mathcal{CV}$, $\_[\_]$, and $\mathcal{FTV}$ “extend” to the abbreviations defined in §2 as expected.

Theorem 4.31

$$\begin{array}{ll}
\mathcal{FV}(\mathsf{true}) & = \emptyset \\
\mathcal{FV}(\mathsf{false}) & = \emptyset \\
\mathcal{FV}(\neg) & = \emptyset \\
\mathcal{FV}(e_1 \wedge e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(e_1 \vee e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(e_1 \Rightarrow e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(\Leftrightarrow) & = \emptyset \\
\mathcal{FV}(e_1 \Leftrightarrow e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(e_1 \not\equiv e_2) & = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \\
\mathcal{FV}(\iota v\!:\!T.e) & = \mathcal{FV}(e) - \{v\} \\
\mathcal{FV}(\forall_T) & = \emptyset \\
\mathcal{FV}(\forall v\!:\!T.\ e) & = \mathcal{FV}(e) - \{v\} \\
\mathcal{FV}(\forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e) & = \mathcal{FV}(e) - \overline{v} \\
\mathcal{FV}(\forall \overline{v}\!:\!\overline{T}.\ e) & = \mathcal{FV}(e) - \overline{v} \\
\mathcal{FV}(\exists_T) & = \emptyset \\
\mathcal{FV}(\exists v\!:\!T.\ e) & = \mathcal{FV}(e) - \{v\} \\
\mathcal{FV}(\exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e) & = \mathcal{FV}(e) - \overline{v} \\
\mathcal{FV}(\exists \overline{v}\!:\!\overline{T}.\ e) & = \mathcal{FV}(e) - \overline{v} \\
\mathcal{FV}(\exists!_T) & = \emptyset \\
\mathcal{FV}(\exists!\, v\!:\!T.\ e) & = \mathcal{FV}(e) - \{v\} \\
\mathcal{FV}(e.f) & = \mathcal{FV}(e)
\end{array}$$

Theorem 4.32

$$\begin{array}{rcl}
\mathsf{true}[u/d] & = & \mathsf{true} \\
\mathsf{false}[u/d] & = & \mathsf{false} \\
\neg[u/d] & = & \neg \\
(e_1 \wedge e_2)[u/d] & = & e_1[u/d] \wedge e_2[u/d] \\
(e_1 \vee e_2)[u/d] & = & e_1[u/d] \vee e_2[u/d] \\
(e_1 \Rightarrow e_2)[u/d] & = & e_1[u/d] \Rightarrow e_2[u/d] \\
\Leftrightarrow [u/d] & = & \Leftrightarrow \\
(e_1 \Leftrightarrow e_2)[u/d] & = & e_1[u/d] \Leftrightarrow e_2[u/d] \\
(e_1 \not\equiv e_2)[u/d] & = & e_1[u/d] \not\equiv e_2[u/d] \\
(\iota v\!:\!T.e)[u/d] & = & \begin{cases} \iota v\!:\!T.e & \mathit{if} \quad u = v \\ \iota v\!:\!T.e[u/d] & \mathit{otherwise} \end{cases} \\
\forall_T[u/d] & = & \forall_T \\
(\forall v\!:\!T.\ e)[u/d] & = & \begin{cases} \forall v\!:\!T.\ e & \mathit{if} \quad u = v \\ \forall v\!:\!T.\ e[u/d] & \mathit{otherwise} \end{cases} \\
(\forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e)[u/d] & = & \begin{cases} \forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e & \mathit{if} \quad u \in \overline{v} \\ \forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e[u/d] & \mathit{otherwise} \end{cases} \\
(\forall \overline{v}\!:\!\overline{T}.\ e)[u/d] & = & \begin{cases} \forall \overline{v}\!:\!\overline{T}.\ e & \mathit{if} \quad u \in \overline{v} \\ \forall \overline{v}\!:\!\overline{T}.\ e[u/d] & \mathit{otherwise} \end{cases} \\
\exists_T[u/d] & = & \exists_T \\
(\exists v\!:\!T.\ e)[u/d] & = & \begin{cases} \exists v\!:\!T.\ e & \mathit{if} \quad u = v \\ \exists v\!:\!T.\ e[u/d] & \mathit{otherwise} \end{cases} \\
(\exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e)[u/d] & = & \begin{cases} \exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e & \mathit{if} \quad u \in \overline{v} \\ \exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e[u/d] & \mathit{otherwise} \end{cases} \\
(\exists \overline{v}\!:\!\overline{T}.\ e)[u/d] & = & \begin{cases} \exists \overline{v}\!:\!\overline{T}.\ e & \mathit{if} \quad u \in \overline{v} \\ \exists \overline{v}\!:\!\overline{T}.\ e[u/d] & \mathit{otherwise} \end{cases} \\
\exists!_T[u/d] & = & \exists!_T \\
(\exists!\, v\!:\!T.\ e)[u/d] & = & \begin{cases} \exists!\, v\!:\!T.\ e & \mathit{if} \quad u = v \\ \exists!\, v\!:\!T.\ e[u/d] & \mathit{otherwise} \end{cases} \\
(e.f)[u/d] & = & e[u/d].f
\end{array}$$

Theorem 4.33

$$
\begin{array}{ll}
\mathcal{CV}(\mathsf{true}, u) & = \emptyset \\
\mathcal{CV}(\mathsf{false}, u) & = \emptyset \\
\mathcal{CV}(\neg, u) & = \emptyset \\
\mathcal{CV}(e_1 \wedge e_2, u) & = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\mathcal{CV}(e_1 \vee e_2, u) & = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\mathcal{CV}(e_1 \Rightarrow e_2, u) & = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\mathcal{CV}(\Leftrightarrow, u) & = \emptyset \\
\mathcal{CV}(e_1 \Leftrightarrow e_2, u) & = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\mathcal{CV}(e_1 \not\equiv e_2, u) & = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\mathcal{CV}(\iota v : T.e, u) & = \begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\forall_T, u) & = \emptyset \\
\mathcal{CV}(\forall v : T.\ e, u) & = \begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\forall v_1 : T_1, \ldots, v_n : T_n.\ e, u) & = \begin{cases} \overline{v} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \overline{v} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\forall \overline{v} : \overline{T}.\ e, u) & = \begin{cases} \overline{v} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \overline{v} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\exists_T, u) & = \emptyset \\
\mathcal{CV}(\exists v : T.\ e, u) & = \begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\exists v_1 : T_1, \ldots, v_n : T_n.\ e, u) & = \begin{cases} \overline{v} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \overline{v} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\exists \overline{v} : \overline{T}.\ e, u) & = \begin{cases} \overline{v} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \overline{v} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(\exists!_T, u) & = \emptyset \\
\mathcal{CV}(\exists!\, v : T.\ e, u) & = \begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \mathit{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \mathit{otherwise} \end{cases} \\
\mathcal{CV}(e.f, u) & = \mathcal{CV}(e, u)
\end{array}
$$

Theorem 4.34

$$
\begin{aligned}
\mathsf{true}[\sigma] &= \mathsf{true} \\
\mathsf{false}[\sigma] &= \mathsf{false} \\
\neg[\sigma] &= \neg \\
(e_1 \wedge e_2)[\sigma] &= e_1[\sigma] \wedge e_2[\sigma] \\
(e_1 \vee e_2)[\sigma] &= e_1[\sigma] \vee e_2[\sigma] \\
(e_1 \Rightarrow e_2)[\sigma] &= e_1[\sigma] \Rightarrow e_2[\sigma] \\
\Leftrightarrow [\sigma] &= \emptyset \\
(e_1 \Leftrightarrow e_2)[\sigma] &= e_1[\sigma] \Leftrightarrow e_2[\sigma] \\
(e_1 \not\equiv e_2)[\sigma] &= e_1[\sigma] \not\equiv e_2[\sigma] \\
(\iota v : T.e)[\sigma] &= \iota v : T[\sigma].e[\sigma] \\
\forall_T[\sigma] &= \forall_{T[\sigma]} \\
(\forall v : T.\ e)[\sigma] &= \forall v : T[\sigma].\ e[\sigma] \\
(\forall v_1 : T_1, \ldots, v_n : T_n.\ e)[\sigma] &= \forall v_1 : T_1[\sigma], \ldots, v_n : T_n[\sigma].\ e[\sigma] \\
(\forall \overline{v} : \overline{T}.\ e)[\sigma] &= \forall \overline{v} : \overline{T}[\sigma].\ e[\sigma] \\
\exists_T[\sigma] &= \exists_{T[\sigma]} \\
(\exists v : T.\ e)[\sigma] &= \exists v : T[\sigma].\ e[\sigma] \\
(\exists v_1 : T_1, \ldots, v_n : T_n.\ e)[\sigma] &= \exists v_1 : T_1[\sigma], \ldots, v_n : T_n[\sigma].\ e[\sigma] \\
(\exists \overline{v} : \overline{T}.\ e)[\sigma] &= \exists \overline{v} : \overline{T}[\sigma].\ e[\sigma] \\
\exists!_T[\sigma] &= \exists!_{T[\sigma]} \\
(\exists!\, v : T.\ e)[\sigma] &= \exists!\, v : T[\sigma].\ e[\sigma] \\
(e.f)[\sigma] &= e[\sigma].f
\end{aligned}
$$

Theorem 4.35

$$\begin{array}{ll}
\mathcal{FTV}(\mathsf{true}) & = \emptyset \\
\mathcal{FTV}(\mathsf{false}) & = \emptyset \\
\mathcal{FTV}(\neg) & = \emptyset \\
\mathcal{FTV}(e_1 \wedge e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(e_1 \vee e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(e_1 \Rightarrow e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(\Leftrightarrow) & = \emptyset \\
\mathcal{FTV}(e_1 \Leftrightarrow e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(e_1 \not\equiv e_2) & = \mathcal{FTV}(e_1) \cup \mathcal{FTV}(e_2) \\
\mathcal{FTV}(\iota v\!:\!T.e) & = \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \\
\mathcal{FTV}(\forall_T) & = \mathcal{FTV}(T) \\
\mathcal{FTV}(\forall v\!:\!T.\ e) & = \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \\
\mathcal{FTV}(\forall v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e) & = \mathcal{FTV}(e) \cup \bigcup_i \mathcal{FTV}(T_i) \\
\mathcal{FTV}(\forall \overline{v}\!:\!\overline{T}.\ e) & = \mathcal{FTV}(e) \cup \bigcup_i \mathcal{FTV}(T_i) \\
\mathcal{FTV}(\exists_T) & = \mathcal{FTV}(T) \\
\mathcal{FTV}(\exists v\!:\!T.\ e) & = \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \\
\mathcal{FTV}(\exists v_1\!:\!T_1, \ldots, v_n\!:\!T_n.\ e) & = \mathcal{FTV}(e) \cup \bigcup_i \mathcal{FTV}(T_i) \\
\mathcal{FTV}(\exists \overline{v}\!:\!\overline{T}.\ e) & = \mathcal{FTV}(e) \cup \bigcup_i \mathcal{FTV}(T_i) \\
\mathcal{FTV}(\exists!_T) & = \mathcal{FTV}(T) \\
\mathcal{FTV}(\exists!\, v\!:\!T.\ e) & = \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \\
\mathcal{FTV}(e.f) & = \mathcal{FTV}(e) \cup \bigcup_i \mathcal{FTV}(T_i)
\end{array}$$

4.4 Proof-theoretical properties

Any prefix of a well-formed context is itself well-formed:

Theorem 4.36

$$\vdash cx_1, cx_2 : \textsc{context} \quad \Rightarrow \quad \vdash cx_1 : \textsc{context}$$

If we derive a judgement with some context, that context is well-formed:

Theorem 4.37

$$cx \vdash \ldots \quad \Rightarrow \quad \vdash cx : \textsc{context}$$

The following four theorems say that well-formed contexts have no duplicate types, ops, and (type) variables:

Theorem 4.38

$$\vdash cx_1, \mathsf{ty}\ \tau\!:\!n, cx_2 : \textsc{context} \ \Rightarrow\ \tau \notin \mathcal{TN}(cx_1, cx_2)$$

Theorem 4.39

$$\vdash cx_1, \mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T, cx_2 : \textsc{context} \ \Rightarrow\ o \notin \mathcal{ON}(cx_1, cx_2)$$

Theorem 4.40

$$\vdash cx_1, \mathsf{tvar}\ \beta, cx_2 : \textsc{context} \ \Rightarrow\ \beta \notin \mathcal{TV}(cx_1, cx_2)$$

Theorem 4.41

$$\vdash cx_1, \mathsf{var}\ v\!:\!T, cx_2 : \textsc{context} \ \Rightarrow\ v \notin \mathcal{V}(cx_1, cx_2)$$

The type of an op declared in a well-formed context is well-formed in the prefix of the context that precedes the op, extended with the type variables over which the op is polymorphic:

Theorem 4.42

$$\vdash cx_1, \mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T, cx_2 : \textsc{context} \quad\Rightarrow\quad cx_1, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type}$$

A type definition in a well-formed context defines a type previously declared in the context with the right arity, and the defining type is well-formed in the prefix of the context that precedes the type definition, extended with the type variable arguments of the defined type:

Theorem 4.43

$$\vdash cx_1, \mathsf{def}\ \tau[\overline{\beta}] = T, cx_2 : \textsc{context} \quad\Rightarrow\quad \mathsf{ty}\ \tau\!:\!|\overline{\beta}| \in cx_1 \quad\wedge\quad cx_1, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type}$$

An axiom declared in a well-formed context is well-typed (with type Bool) in the prefix of the context that precedes the axiom, extended with the type variables over which the axiom is polymorphic:

Theorem 4.44

$$\vdash cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e, cx_2 : \textsc{context} \quad\Rightarrow\quad cx_1, \mathsf{tvar}\ \overline{\beta} \vdash e : \mathsf{Bool}$$

A lemma declared in a well-formed context is a theorem in the prefix of the context that precedes the lemma, extended with the type variables over which the lemma is polymorphic:

Theorem 4.45

$$\vdash cx_1, \mathsf{lem}\ \{\overline{\beta}\}\ e, cx_2 : \textsc{context} \quad\Rightarrow\quad cx_1, \mathsf{tvar}\ \overline{\beta} \vdash e$$

The type of a variable declared in a well-formed context is well-formed in the prefix of the context that precedes the variable declaration:

Theorem 4.46

$$\vdash cx_1, \mathsf{var}\ v\!:\!T, cx_2 : \textsc{context} \quad\Rightarrow\quad cx_1 \vdash T : \textsc{type}$$

A well-formed type variable is declared in the context:

Theorem 4.47

$$cx \vdash \beta : \textsc{type} \quad\Rightarrow\quad \beta \in \mathcal{TV}(cx)$$

The type name of a well-formed type instance is declared in the context, with an arity that matches the type arguments, and each type argument is well-formed:

Theorem 4.48

$$cx \vdash \tau[\overline{T}] : \textsc{type} \quad\Rightarrow\quad \mathsf{ty}\ \tau\!:\!|\overline{T}| \in cx \quad\wedge\quad (\forall i.\ \ cx \vdash T_i : \textsc{type})$$

The domain and range types of a well-formed arrow type are well-formed:

Theorem 4.49

$$cx \vdash T_1 \to T_2 : \textsc{type} \quad\Rightarrow\quad cx \vdash T_1 : \textsc{type} \quad\wedge\quad cx \vdash T_2 : \textsc{type}$$

The component types of a well-formed record type are well-formed:

Theorem 4.50

$$cx \vdash \prod_i f_i\ T_i : \textsc{type} \quad\Rightarrow\quad (\forall i.\ \ cx \vdash T_i : \textsc{type})$$

The predicate of a restriction type is well-typed, with the expected type:

Theorem 4.51

$$cx \vdash T|r : \text{TYPE} \quad \Rightarrow \quad cx \vdash r : T \to \mathsf{Bool}$$

The predicate of a well-formed restriction type has no free variables:

Theorem 4.52

$$cx \vdash T|r : \text{TYPE} \quad \Rightarrow \quad \mathcal{FV}(r) = \emptyset$$

The subtype predicates that appear in (provable) subtype judgements have no free variables:

Theorem 4.53

$$cx \vdash T_1 \prec_r T_2 \;\Rightarrow\; \mathcal{FV}(r) = \emptyset$$

All the free variables in a well-typed expression or a theorem are declared in the context, and all the free variables in an axiom or lemma of a well-formed context are declared before the axiom or lemma:

Theorem 4.54

$$\begin{array}{lll} cx \vdash e : T & \Rightarrow & \mathcal{FV}(e) \subseteq \mathcal{V}(cx) \\ cx \vdash e & \Rightarrow & \mathcal{FV}(e) \subseteq \mathcal{V}(cx) \\ \vdash cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e, cx_2 : \text{CONTEXT} & \Rightarrow & \mathcal{FV}(e) \subseteq \mathcal{V}(cx_1) \\ \vdash cx_1, \mathsf{lem}\ \{\overline{\beta}\}\ e, cx_2 : \text{CONTEXT} & \Rightarrow & \mathcal{FV}(e) \subseteq \mathcal{V}(cx_1) \end{array}$$

All the free type variables in the types and expressions to the right of a provable judgement $cx \vdash \ldots$ are declared in cx, and all the free type variables in a context element of a well-formed context are declared before the context element:

Theorem 4.55

$$\begin{array}{lll} cx \vdash T : \text{TYPE} & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(T) \\ cx \vdash T_1 \approx T_2 & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2) \\ cx \vdash T_1 \prec_r T_2 & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(T_1) \cup \mathcal{FTV}(r) \cup \mathcal{FTV}(T_2) \\ cx \vdash e : T & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(e) \cup \mathcal{FTV}(T) \\ cx \vdash e & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(e) \\ \vdash cx_1, cxel, cx_2 : \text{CONTEXT} & \Rightarrow & \mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cxel) \end{array}$$

The derivation of a judgement does not depend on the choice of the names for the variables in the derivation. So, if we replace a variable declared in the context of the judgement with a fresh one[4] and perform the substitution in the rest of the judgement accordingly, the judgement is still provable:

Theorem 4.56

$$\begin{array}{lllll} u' \ \mathit{fresh} & \wedge & \vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \text{CONTEXT} & \Rightarrow & \vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] : \text{CONTEXT} \\ u' \ \mathit{fresh} & \wedge & cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T : \text{TYPE} & \Rightarrow & cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T : \text{TYPE} \\ u' \ \mathit{fresh} & \wedge & cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T_1 \approx T_2 & \Rightarrow & cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T_1 \approx T_2 \\ u' \ \mathit{fresh} & \wedge & cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T_1 \prec_r T_2 & \Rightarrow & cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T_1 \prec_r T_2 \\ u' \ \mathit{fresh} & \wedge & cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash e : T & \Rightarrow & cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[u/u'] : T \\ u' \ \mathit{fresh} & \wedge & cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash e & \Rightarrow & cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[u/u'] \end{array}$$

The Metaslang logic is monotonic, in the sense that anything can be derived in a bigger context that can be derived in a smaller context:

Theorem 4.57

$$\vdash cx' : \text{CONTEXT} \quad \wedge \quad cx \subseteq cx' \quad \Rightarrow \quad \left(\begin{array}{lll} cx \vdash T : \text{TYPE} & \Rightarrow & cx' \vdash T : \text{TYPE} \\ cx \vdash T_1 \approx T_2 & \Rightarrow & cx' \vdash T_1 \approx T_2 \\ cx \vdash T_1 \prec_r T_2 & \Rightarrow & cx' \vdash T_1 \prec_r T_2 \\ cx \vdash e : T & \Rightarrow & cx' \vdash e : T \\ cx \vdash e & \Rightarrow & cx' \vdash e \end{array} \right)$$

[4]The adjective "fresh" means that the variable does not occur anywhere in the (judgements that comprise the) derivation that is under discussion. This notion could be made more formal, but it should be sufficiently clear as stated.

Note that the monotonicity theorem above also applies to the case in which cx' is a (well-formed) permutation of cx (it is still the case that $cx \subseteq cx'$). In other words, anything that can be derived in a context can be also derived in any well-formed permutation of that context.

The following theorem says that if we derive a judgement $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash \ldots$, then we can substitute the $\overline{\alpha}$ with well-formed types $\overline{S}$, under certain conditions:

Theorem 4.58

$$\begin{array}{l}
\forall i.\ \ cx_1, cx_2 \vdash S_i : \textsc{type} \\
OKsbs(cx_3, \overline{\alpha}, \overline{S}) \\
\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \textsc{context} \\
\quad \Rightarrow \\
\left(\begin{array}{lcl}
cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T : \textsc{type} & \Rightarrow & cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash T[\overline{\alpha}/\overline{S}] : \textsc{type} \\
cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T_1 \approx T_2 & \Rightarrow & cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash T_1[\overline{\alpha}/\overline{S}] \approx T_2[\overline{\alpha}/\overline{S}] \\
cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T_1 \prec_r T_2 & \Rightarrow & cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash T_1[\overline{\alpha}/\overline{S}] \prec_{r[\overline{\alpha}/\overline{S}]} T_2[\overline{\alpha}/\overline{S}] \\
cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash e : T & \Rightarrow & cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash e[\overline{\alpha}/\overline{S}] : T[\overline{\alpha}/\overline{S}] \\
cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash e & \Rightarrow & cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash e[\overline{\alpha}/\overline{S}]
\end{array}\right)
\end{array}$$

The following theorems prove derived well-typedness rules for the abbreviations defined in §2:

Theorem 4.59

$$\frac{\vdash cx : \textsc{context} \qquad v \notin \mathcal{V}(cx)}{\vdash cx, \mathsf{var}\ v\!:\!\mathsf{Bool} : \textsc{context}} \quad (\textsc{cxVdecBool})$$

Theorem 4.60

$$\frac{cx \vdash e : \mathsf{Bool}}{\vdash cx, \mathsf{ax}\ e : \textsc{context}} \quad (\textsc{cxAx0})$$

Theorem 4.61

$$\frac{\vdash cx : \textsc{context}}{cx \vdash \lambda v\!:\!\mathsf{Bool}.\ v : \mathsf{Bool} \to \mathsf{Bool}} \quad (\textsc{exIdBool})$$

Theorem 4.62

$$\frac{\vdash cx : \textsc{context}}{cx \vdash \mathsf{true} : \mathsf{Bool}} \quad (\textsc{exTrue})$$

Theorem 4.63

$$\frac{cx \vdash T : \textsc{type}}{cx \vdash \lambda v\!:\!T.\ \mathsf{true} : T \to \mathsf{Bool}} \quad (\textsc{exConstTrue})$$

Theorem 4.64

$$\frac{\vdash cx : \textsc{context}}{cx \vdash \mathsf{false} : \mathsf{Bool}} \quad (\textsc{exFalse})$$

Theorem 4.65

$$\frac{\vdash cx : \textsc{context}}{cx \vdash \neg : \mathsf{Bool} \to \mathsf{Bool}} \quad (\textsc{exNot})$$

In the proof of the previous theorem, we use exIf0, not exIf. An attempt to use exIf causes a circularity, where in order to prove that $\neg$ is well-typed one has to first prove that $\neg$ is well-typed. The reader can try a backward derivation of $cx, \mathsf{var}\ \gamma\!:\!\mathsf{Bool} \vdash \mathsf{if}\ \gamma\ \mathsf{false}\ \mathsf{true} : \mathsf{Bool}$ where the first rule applied backwards is exIf: while the first two premises (i.e. $cx, \mathsf{var}\ \gamma\!:\!\mathsf{Bool} \vdash \gamma : \mathsf{Bool}$ and $cx, \mathsf{var}\ \gamma : \mathsf{Bool}, \mathsf{ax}\ \gamma \vdash \mathsf{false} : \mathsf{Bool}$) can be derived with ease, the third premise (i.e. $cx, \mathsf{var}\ \gamma : \mathsf{Bool}, \mathsf{ax}\ \neg\ \gamma \vdash \mathsf{true} : \mathsf{Bool}$) requires proving $\vdash cx, \mathsf{var}\ \gamma\!:\!\mathsf{Bool}, \mathsf{ax}\ \neg\ \gamma : \textsc{context}$ (so that we can use exTrue), which requires proving $cx, \mathsf{var}\ \gamma : \mathsf{Bool} \vdash \neg\ \gamma : \mathsf{Bool}$ (so that we can use cxAx), which requires proving $cx, \mathsf{var}\ \gamma : \mathsf{Bool} \vdash \neg : \mathsf{Bool} \to \mathsf{Bool}$ (so that we can use exApp), which causes a circularity. The arguments just given do not constitute a formal proof of the necessity of rule exIf0, but they do suggest te existence of such a proof.

Theorem 4.66

$$\frac{cx \vdash e : \mathsf{Bool}}{cx \vdash \neg\, e : \mathsf{Bool}} \quad (\text{exNeg})$$

Theorem 4.67

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx, \mathsf{ax}\ e_1 \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \wedge e_2 : \mathsf{Bool}} \quad (\text{exConj})$$

Theorem 4.68

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \wedge e_2 : \mathsf{Bool}} \quad (\text{exConj0})$$

Theorem 4.69

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx, \mathsf{ax}\ \neg\, e_1 \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \vee e_2 : \mathsf{Bool}} \quad (\text{exDisj})$$

Theorem 4.70

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \vee e_2 : \mathsf{Bool}} \quad (\text{exDisj0})$$

Theorem 4.71

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx, \mathsf{ax}\ e_1 \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \Rightarrow e_2 : \mathsf{Bool}} \quad (\text{exImpl})$$

Theorem 4.72

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \Rightarrow e_2 : \mathsf{Bool}} \quad (\text{exImpl0})$$

Theorem 4.73

$$\frac{\vdash cx : \text{CONTEXT}}{cx \vdash \Leftrightarrow : \mathsf{Bool} \rightarrow \mathsf{Bool} \rightarrow \mathsf{Bool}} \quad (\text{exIff})$$

Theorem 4.74

$$\frac{\begin{array}{c} cx \vdash e_1 : \mathsf{Bool} \\ cx \vdash e_2 : \mathsf{Bool} \end{array}}{cx \vdash e_1 \Leftrightarrow e_2 : \mathsf{Bool}} \quad (\text{exCoImpl})$$

Theorem 4.75

$$\frac{\begin{array}{c} cx \vdash e_1 : T \\ cx \vdash e_2 : T \end{array}}{cx \vdash e_1 \not\equiv e_2 : \mathsf{Bool}} \quad (\text{exNeq})$$

Theorem 4.76

$$\frac{cx \vdash T : \text{TYPE}}{cx \vdash \forall_T : (T \rightarrow \mathsf{Bool}) \rightarrow \mathsf{Bool}} \quad (\text{exFA})$$

Theorem 4.77

$$\frac{cx, \mathsf{var}\ v\!:\!T \vdash e : \mathsf{Bool}}{cx \vdash \forall v\!:\!T.\ e : \mathsf{Bool}} \quad (\textsc{exForAll})$$

Theorem 4.78

$$\frac{cx \vdash T : \textsc{type}}{cx \vdash \exists_T : (T \rightarrow \mathsf{Bool}) \rightarrow \mathsf{Bool}} \quad (\textsc{exEX})$$

Theorem 4.79

$$\frac{cx, \mathsf{var}\ v\!:\!T \vdash e : \mathsf{Bool}}{cx \vdash \exists v\!:\!T.\ e : \mathsf{Bool}} \quad (\textsc{exExists})$$

Theorem 4.80

$$\frac{cx \vdash T : \textsc{type}}{cx \vdash \exists!_T : (T \rightarrow \mathsf{Bool}) \rightarrow \mathsf{Bool}} \quad (\textsc{exEX1})$$

Theorem 4.81

$$\frac{cx, \mathsf{var}\ v\!:\!T \vdash e : \mathsf{Bool}}{cx \vdash \exists!\, v\!:\!T.\ e : \mathsf{Bool}} \quad (\textsc{exExists1})$$

Theorem 4.82

$$\frac{\begin{array}{c} cx \vdash \prod_i f_i\ T_i : \textsc{type} \\ cx \vdash e : \prod_i f_i\ T_i \end{array}}{cx \vdash e.f_j : T_j} \quad (\textsc{exDotProj0})$$

5 Models

[[[TO DO]]]

This section defines the notion of model of a context (recall that specs are contexts without variable and type variable declarations).

A model of a context is a mapping from names declared in the context to suitable set-theoretic entities. For instance, a type name τ of arity n is mapped to an n-ary function over sets (if $n = 0$, the model maps the type name simply to a set). The mapping is extended to all well-formed types, which are mapped to sets, and to all well-typed expressions, which are mapped to elements of the sets that their types map to. The model must satisfy all the type definitions, op definitions, and axioms of the context.

It should be possible to prove the soundness of the rules to derive judgements with respect to models.

Since higher-order logic is notoriously incomplete, it is not possible to prove completeness of the rules to derive assertions. However, it should be possible to prove completeness with respect to general (a.k.a. Henkin) models. A general model is one in which the type $T_1 \rightarrow T_2$ is a subset of all functions from T_1 to T_2, and not necessarily the set of all such functions (as in standard models). Since there are more general models than standard models (a standard model is also a general model but not all general models are standard models), fewer formulas are true in all general models than in all standard models.

Perhaps this section should also contain a proof of the consistency of the Metaslang logic, analogously to the proof of the consistency of the higher-order logic defined in [2].

6 Other Metaslang constructs

6.1 Records and tuples

A record constructor has the form

$$\mathsf{rec}_{\prod_i f_i\ T_i}$$

where $\prod_i f_i\ T_i \in \mathit{Type}$. We introduce the abbreviation

$$\mathsf{rec}_{\prod_i f_i\ T_i} \quad \longrightarrow \quad \lambda\gamma_1 : T_1.\ \ldots \lambda\gamma_n : T_n.\ \iota\gamma : \textstyle\prod_i f_i\ T_i.\ \bigwedge_i(\gamma.f_i \equiv \gamma_i)$$

where the γ_i are fixed but unspecified names in $\mathcal{N}$ that are all distinct from each other and from γ. The record constructor $\mathsf{rec}_{\prod_i f_i\ T_i}$ maps n arguments of the component types of the record type that tags the record constructor to the record with those arguments as components via the description operator (records are characterized by the values of their components). We may write $\mathsf{rec}_{\prod_i f_i\ T_i}$ as just rec when the record type is inferrable or irrelevant. Note that $\mathsf{rec}_{\prod \epsilon}$ denotes the empty record (a conjunction $\bigwedge_i e_i$ consisting of no conjuncts, i.e. such that $n = 0$, stands for true).

We introduce the abbreviations

$$\begin{aligned} \langle f_1 \overset{T_1}{\leftarrow} e_1\ \ldots\ f_n \overset{T_n}{\leftarrow} e_n \rangle \quad &\longrightarrow \quad \mathsf{rec}_{\prod_i f_i\ T_i}\ e_1\ \ldots\ e_n \\ \langle e_1, \ldots, e_n \rangle \quad &\longrightarrow \quad \langle \pi_1 \leftarrow e_1\ \ldots\ \pi_n \leftarrow e_n \rangle \end{aligned}$$

The notation $\langle f_i \overset{T_i}{\leftarrow} e_i \rangle_i$ is a more readable version of an applied record constructor. We may write $\langle f_i \overset{T_i}{\leftarrow} e_i \rangle_i$ as just $\langle f_i \leftarrow e_i \rangle_i$ when the component types are inferrable or irrelevant. A tuple $\langle \overline{e} \rangle$ captures tuple displays as defined in [1]: it abbreviates a record construction of a product type whose component types are implicit; this is why no component types appear in $\langle \overline{e} \rangle$.

Theorem 6.1

$$\begin{aligned} \mathcal{FV}(\mathsf{rec}) &= \emptyset \\ \mathcal{FV}(\langle f_i \leftarrow e_i \rangle_i) &= \textstyle\bigcup_i \mathcal{FV}(e_i) \\ \mathcal{FV}(\langle \overline{e} \rangle) &= \textstyle\bigcup_i \mathcal{FV}(e_i) \end{aligned}$$

$$\begin{aligned} \mathsf{rec}[u/d] &= \mathsf{rec} \\ \langle f_i \leftarrow e_i \rangle_i[u/d] &= \langle f_i \leftarrow e_i[u/d] \rangle_i \\ \langle \overline{e} \rangle[u/d] &= \langle e_1[u/d], \ldots, e_n[u/d] \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{CV}(\mathsf{rec}, u) &= \emptyset \\ \mathcal{CV}(\langle f_i \leftarrow e_i \rangle_i, u) &= \textstyle\bigcup_i \mathcal{CV}(e_i, u) \\ \mathcal{CV}(\langle \overline{e} \rangle, u) &= \textstyle\bigcup_i \mathcal{CV}(e_i, u) \end{aligned}$$

$$\begin{aligned} \mathsf{rec}_{\prod_i f_i\ T_i}[\sigma] &= \mathsf{rec}_{\prod_i f_i\ T_i[\sigma]} \\ \langle f_i \overset{T_i}{\leftarrow} e_i \rangle_i[\sigma] &= \langle f_i \overset{T_i[\sigma]}{\leftarrow} e_i[\sigma] \rangle_i \\ \langle \overline{e} \rangle[\sigma] &= \langle e_1[\sigma], \ldots, e_n[\sigma] \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{FTV}(\mathsf{rec}_{\prod_i f_i\ T_i}) &= \textstyle\bigcup_i \mathcal{FTV}(T_i) \\ \mathcal{FTV}(\langle f_i \overset{T_i}{\leftarrow} e_i \rangle_i) &= \textstyle\bigcup_i (\mathcal{FTV}(T_i) \cup \mathcal{FTV}(e_i)) \\ \mathcal{FTV}(\langle \overline{e} \rangle) &= \textstyle\bigcup_i (\mathcal{FTV}(T_i) \cup \mathcal{FTV}(e_i)) \end{aligned}$$

6.2 Record updates

A record update as defined in [1] is just an abbreviation for an explicit record construction whose fields are assigned projections from the two expressions, as appropriate.

A record updater has the form

$$\ll_{\prod_i f_i\ T_i, \prod_i f'_i\ T'_i, \prod_i f''_i\ T''_i}$$

where $\prod_i f_i\ T_i, \prod_i f'_i\ T'_i, \prod_i f''_i\ T''_i \in \mathit{Type}$ and $\overline{f}' \cap \overline{f}'' = \emptyset$. Its type is

$$(\textstyle\prod_i f_i\ T_i \times \prod_i f'_i\ T'_i) \rightarrow (\prod_i f_i\ T_i \times \prod_i f''_i\ T''_i) \rightarrow (\prod_i f_i\ T_i \times \prod_i f'_i\ T'_i \times \prod_i f''_i\ T''_i)$$

i.e. it operates on two records whose common fields have the same types and returns a record with the union of all the fields. The resulting record is obtained by putting together the second record with the fields of the first one that do not appear in the second record. In other words, we start with the first record, overwrite the common fields with those from the second record, and add the extra fields from the second record. This is defined by the abbreviation

$$\ll \quad \longrightarrow \quad \lambda\gamma{:}T'.\ \lambda\gamma'{:}T''.\ \langle \overline{f} \leftarrow (\gamma'.\overline{f})\ \overline{f}' \leftarrow (\gamma.\overline{f}')\ \overline{f}'' \leftarrow (\gamma'.\overline{f}'')\rangle$$

where we leave the record types that tag $\ll$ implicit, where T' and T'' respectively stand for $\prod_i f_i\ T_i \times \prod_i f'_i\ T'_i$ and $\prod_i f_i\ T_i \times \prod_i f''_i\ T''_i$, and where the notation $\overline{f} \leftarrow e.\overline{f}$ stands for $f_1 \leftarrow e.f_1\ \ldots\ f_n \leftarrow e.f_n$.

The infix form is defined by the abbreviation

$$e_1 {\ll} e_2 \quad \longrightarrow \quad \ll\ e_1\ e_2$$

where again we leave the tagging record types implicit.

Theorem 6.2

$$\begin{aligned} \mathcal{FV}(\ll) &= \emptyset \\ \mathcal{FV}(e_1 {\ll} e_2) &= \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \end{aligned}$$

$$\begin{aligned} {\ll}_{\prod_i f_i\ T_i, \prod_i f'_i\ T'_i, \prod_i f''_i\ T''_i}[\sigma] &= {\ll}_{\prod_i f_i T_i[\sigma], \prod_i f'_i T'_i[\sigma], \prod_i f''_i T''_i[\sigma]} \\ (e_1 {\ll} e_2)[\sigma] &= e_1[\sigma] {\ll} e_2[\sigma] \end{aligned}$$

$$\begin{aligned} {\ll}_{\prod_i f_i\ T_i, \prod_i f'_i\ T'_i, \prod_i f''_i\ T''_i}[u/d] &= {\ll}_{\prod_i f_i\ T_i, \prod_i f'_i\ T'_i, \prod_i f''_i\ T''_i} \\ (e_1 {\ll} e_2)[u/d] &= e_1[u/d] {\ll} e_2[u/d] \end{aligned}$$

$$\begin{aligned} \mathcal{CV}(\ll, u) &= \emptyset \\ \mathcal{CV}(e_1 {\ll} e_2, u) &= \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \end{aligned}$$

6.3 Binding conditionals

A binding conditional, currently absent from [1], has the form

$$\begin{aligned} \mathsf{cond}_T\ \langle & v_{1,1}{:}T_{1,1}, \ldots, v_{m_1,1}{:}T_{m_1,1}.\ b_1 \rightarrow e_1 \\ & \ldots \\ & v_{1,n}{:}T_{1,n}, \ldots, v_{m_n,n}{:}T_{m_n,n}.\ b_n \rightarrow e_n\rangle \end{aligned}$$

where $T \in \mathit{Type}$, $\overline{\overline{v}} \in (\mathcal{N}^{(*)})^+$, $\overline{\overline{T}} \in (\mathit{Type}^*)^+$, and $\overline{b}, \overline{e} \in \mathit{Exp}^+$.

Its intuitive meaning is the following. Each b_i is a boolean expression: if b_1 holds, the result of the conditional is e_1; otherwise, if b_2 holds, the result is e_2; and so on. At least one b_i must hold. Each branch i binds zero or more variables $\overline{v}_i$, whose scope is not only the condition b_i, but also the result expression e_i. Each branch i reads as: if there exist $\overline{v}_i$ with respective types $\overline{T}_i$ such that b_i holds, then the result is e_i, which can refer to the bound variables. The value of e_i must be the same for all values assigned to $\overline{v}_i$ that make b_i true. All the e_i must have type T.

We introduce the abbreviations

$$\begin{aligned} \mathsf{cond}_T\ \langle \overline{v}{:}\overline{T}.\ b \rightarrow e\rangle \quad &\longrightarrow \quad \iota\gamma_{\overline{v},b,e}{:}T.\exists\overline{v}{:}\overline{T}.\ (b \wedge \gamma_{\overline{v},b,e} \equiv e) \\ \mathsf{cond}_T\ \langle \overline{v}_1{:}\overline{T}_1.\ b_1 \rightarrow e_1 \quad &\longrightarrow \quad \mathsf{if}\ (\exists\overline{v}_1{:}\overline{T}_1.\ b_1) \\ \ldots \qquad\qquad & \qquad\qquad (\iota\gamma_{\overline{v}_1,b_1,e_1}{:}T.\exists\overline{v}_1{:}\overline{T}_1.\ (b_1 \wedge \gamma_{\overline{v}_1,b_1,e_1} \equiv e_1)) \\ \overline{v}_n{:}\overline{T}_n.\ b_n \rightarrow e_n\rangle \quad & \qquad\qquad (\mathsf{cond}_T\ \langle \overline{v}_2{:}\overline{T}_2.\ b_2 \rightarrow e_2\ \ldots\ \overline{v}_n{:}\overline{T}_n.\ b_n \rightarrow e_n\rangle) \end{aligned}$$

where the second abbreviation only applies when $n > 1$ and where $\gamma_{\overline{v},b,e}$ is, for each triple $\langle \overline{v}, b, e\rangle \in \mathcal{N}^* \times \mathit{Exp} \times \mathit{Exp}$, a fixed but unspecified name in $\mathcal{N}$ such that $\gamma_{\overline{v},b,e} \notin \overline{v} \cup \mathcal{FV}(b) \cup \mathcal{FV}(e)$.

6.4 Pattern matching

A pattern matching expression, currently present in [1] in a more limited form than defined here, has the form

$$\begin{aligned}\mathsf{case}_{T,T'}\ e\ \langle & v_{1,1}:T_{1,1},\ldots,v_{m_1,1}:T_{m_1,1}.\ p_1\rightarrow e_1\\ & \ldots\\ & v_{1,n}:T_{1,n},\ldots,v_{m_n,n}:T_{m_n,n}.\ p_n\rightarrow e_n\rangle\end{aligned}$$

where $T,T'\in \mathit{Type}$, $e\in \mathit{Exp}$, $\overline{\overline{v}}\in(\mathcal{N}^{(*)})^+$, $\overline{\overline{T}}\in(\mathit{Type}^*)^+$, and $\overline{p},\overline{e}\in \mathit{Exp}^+$.

Its intuitive meaning is the following. Each p_i is a pattern expression of type T against which e, which must also have type T, is compared: if e matches p_1, the result of the case expression is e_1; otherwise, if e matches p_2 holds, the result is e_2; and so on. The target expression e must match at least one p_i. Each branch i binds zero or more variables $\overline{v}_i$, whose scope is not only the pattern p_i, but also the result expression e_i. Here, "e matches p_i" means that $e\equiv p_i$ for some values of $\overline{v}_i$ of types $\overline{T}_i$. Every branch i reads as: if there exist $\overline{v}_i$ with respective types $\overline{T}_i$ such that $e\equiv b_i$ holds, then the result is e_i, which can refer to the bound variables. The value of e_i must be the same for all values assigned to $\overline{v}_i$ that make $e\equiv b_i$ true. All the e_i must have type T'.

We introduce the abbreviation

$$\mathsf{case}_{T,T'}\ e\ \langle\overline{v}_i:\overline{T}_i.\ p_i\rightarrow e_i\rangle_i\quad\longrightarrow\quad\begin{cases}\mathsf{cond}_{T'}\ \langle\overline{v}_i:\overline{T}_i.\ (e\equiv p_i)\rightarrow e_i\rangle_i\\ \qquad\text{if}\quad \mathcal{FV}(e)\cap\bigcup_i\overline{v}_i=\emptyset\\ \\ \mathsf{case}_{T,T'}\ e\ \langle\gamma_{\overline{\overline{v}},e}:T.\ \gamma_{\overline{\overline{v}},e}\rightarrow\mathsf{case}_{T,T'}\ \gamma_{\overline{\overline{v}},e}\ \langle\overline{v}_i:\overline{T}_i.\ p_i\rightarrow e_i\rangle_i\rangle\\ \qquad\text{otherwise}\end{cases}$$

where $\gamma_{\overline{\overline{v}},e}$ is, for each pair $\langle\overline{\overline{v}},e\rangle\in(\mathcal{N}^{(*)})^+\times \mathit{Exp}$, a fixed but unspecified name in $\mathcal{N}$ such that $\gamma_{\overline{\overline{v}},e}\notin\bigcup_i\overline{v}_i\cup\mathcal{FV}(e)$. The nested case expressions $\mathsf{case}_{T,T'}\ e\ \langle\gamma_{\overline{\overline{v}},e}:T.\ \gamma_{\overline{\overline{v}},e}\rightarrow\mathsf{case}\ldots\rangle$ coincide with $\mathsf{let}_{T'}\ \gamma_{\overline{\overline{v}},e}:T\leftarrow e\ \mathsf{in}\ \mathsf{case}\ldots$ (see let expressions, introduced later), which should be more intuitive. The reason for introducing the extra $\gamma_{\overline{\overline{v}},e}$ variable via a let expression when $\mathcal{FV}(e)\cap\bigcup_i\overline{v}_i\neq\emptyset$ is to prevent the bindings of the branches to capture free variables in the target expression e. Note that there is no circularity: both nested case expressions readily expand into binding conditionals, because of the hypothesis that $\gamma_{\overline{\overline{v}},e}\notin\bigcup_i\overline{v}_i\cup\mathcal{FV}(e)$.

Here, the patterns p_i can be any expressions. In [1], patterns are a separate syntactic category, which can be regarded as a subset of expressions.

Aliased patterns as defined in [1] can be easily captured as follows. Given an aliased pattern $(v:T\ \mathsf{as}\ p)$ in a branch, first drop v and all other alias variables that appear in any subpatterns of p, obtaining an alias-free pattern p'. Then, use the abbreviation expansion given above, obtaining a binding conditional where the branch condition is $e\equiv p'$. Finally, conjoin that branch condition with equations that equate the alias variables with the patterns, e.g. $v\equiv p'$, at the same time adding those variables, e.g. $v:T$, to the variables $\overline{v}:\overline{T}$ bound by the branch. An alternative to adding equations for the alias variables to the branch condition is to add simple let expressions for those variables in the result expression of the branch, e.g. $\mathsf{let}_{\ldots}\ v:T\leftarrow p'\ \mathsf{in}\ \ldots$

If the aliased patterns defined in [1] were generalized to allow an arbitrary pattern at the left of as, they could be captured as follows. First, recursively expand $p\ \mathsf{as}\ p'$ into a sequence of alias-free patterns $p_1,\ldots,p_n$ that are all equal to p and p'. Then, use the conjunction $\bigwedge_i e\equiv p_i$ as the condition of the branch of the binding conditional. The potentially exponential expansion of $p\ \mathsf{as}\ p'$ into $p_1,\ldots,p_n$ can be avoided by first introducing fresh variables for each left-hand side of each as in $p\ \mathsf{as}\ p'$, then using the method described earlier for aliased patterns of the form $v:T\ \mathsf{as}\ p$, and finally conjoining the branch condition with equations that equate the fresh variables to the corresponding left-hand sides.

6.5 Let expressions

6.5.1 Non-recursive

As in [1], a non-recursive let expression is defined as a case expression with one branch.

A non-recursive let expression has the form

$$\mathsf{let}_{T,T'}\ \overline{v}\!:\!\overline{T}.\ p\!\leftarrow\! e\ \mathsf{in}\ e'$$

where $T, T' \in \mathit{Type}$, $\overline{v} \in \mathcal{N}^{(*)}$, $\overline{T} \in \mathit{Type}^*$, and $p, e, e' \in \mathit{Exp}$.

We introduce the abbreviation

$$\mathsf{let}_{T,T'}\ \overline{v}\!:\!\overline{T}.\ p\!\leftarrow\! e\ \mathsf{in}\ e' \quad\longrightarrow\quad \mathsf{case}_{T,T'}\ e\ \langle \overline{v}\!:\!\overline{T}.\ p\!\rightarrow\! e'\rangle$$

which captures a generalization of non-recursive let expressions as defined in [1], in the same way as the pattern matching defined here generalizes the pattern matching defined in [1].

6.5.2 Simple

A simple let expression has the form

$$\mathsf{let}_{T'}\ v\!:\!T\!\leftarrow\! e\ \mathsf{in}\ e'$$

where $v \in \mathcal{N}$, $T, \mathit{typ}' \in \mathit{Type}$, and $e, e' \in \mathit{Exp}$.

We introduce the abbreviation

$$\mathsf{let}_{T'}\ v\!:\!T\!\leftarrow\! e\ \mathsf{in}\ e' \quad\longrightarrow\quad \mathsf{let}_{T,T'}\ v\!:\!T.\ v\!\leftarrow\! e\ \mathsf{in}\ e'$$

which captures a common kind of non-recursive let expression.

6.5.3 Recursive

Recursive let expressions are defined in terms of non-recursive let expressions and binding conditionals.

A recursive let expression has the form

$$\mathsf{let}_{T}\ \langle v_1\!:\!T_1\!\leftarrow\! e_1 \ldots v_n\!:\!T_n\!\leftarrow\! e_n\rangle\ \mathsf{in}\ e$$

where $T \in \mathit{Type}$, $\overline{v} \in \mathcal{N}^{(+)}$, $\overline{T} \in \mathit{Type}^+$, $\overline{e} \in \mathit{Exp}^+$, and $e \in \mathit{Exp}$.

We introduce the abbreviation

$$\mathsf{let}_{T}\ \langle v_i\!:\!T_i\!\leftarrow\! e_i\rangle_i\ \mathsf{in}\ e \quad\longrightarrow\quad \mathsf{let}_{\prod_i T_i,T}\ \overline{v}\!:\!\overline{T}.\ \langle\overline{v}\rangle\!\leftarrow\!\mathsf{cond}_{\prod_i T_i}\ \langle\overline{v}\!:\!\overline{T}.\ \langle\overline{v}\rangle \equiv \langle\overline{e}\rangle \!\rightarrow\! \langle\overline{v}\rangle\rangle\ \mathsf{in}\ e$$

which captures recursive let expressions as defined in [1].

6.6 Op definitions

An op definition as defined in [1] has the form

$$\mathsf{def}\ \{\overline{\beta}'\}\ (o\!:\!\{\overline{\beta}\}\ T) = e$$

where $o \in \mathcal{N}$, $\overline{\beta}, \overline{\beta}' \in \mathcal{N}^{(*)}$, $T \in \mathit{Type}$, $e \in \mathit{Exp}$, $|\overline{\beta}| = |\overline{\beta}'|$, and every $o[\overline{T}]$ occurring in e is such that $\overline{T} = \overline{\beta}'$. We introduce the abbreviation

$$\begin{aligned}\mathsf{def}\ \{\overline{\beta}'\}\ (o\!:\!\{\overline{\beta}\}\ T) = e \quad\longrightarrow\quad & \mathsf{lem}\ \{\overline{\beta}'\}\ \exists!\,\gamma_e\!:\!T[\overline{\beta}/\overline{\beta}'].\ \gamma_e \equiv e',\\ & \mathsf{ax}\ \{\overline{\beta}'\}\ o[\overline{\beta}'] \equiv e\end{aligned}$$

where γ_e is, for each expression $e \in \mathit{Exp}$, a fixed but unspecified name in $\mathcal{N}$ such that $\gamma_e \notin \mathcal{FV}(e)$ and where e' is the result of replacing every occurrence of $o[\overline{\beta}']$ with γ_e in e. An op definition $\mathsf{def}\ \{\overline{\beta}'\}\ (o\!:\!\{\overline{\beta}\}\ T) = e$ should reference an op $\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T$ previously declared in the context in which the op definition appears (if it does not, the addition of the axiom would not yield a well-formed context). The defining body e may reference o (in which case the op definition is recursive), but all the instances of o must be the same, namely $o[\overline{\beta}']$. We may have $\overline{\beta}' = \overline{\beta}$, but in general it is allowed to "rename" $\overline{\beta}$ to $\overline{\beta}'$ when defining the op. The op definition is logically just an axiom (the one that the abbreviation expands into), with the proof obligation expressed by the lemma, namely that there is a unique value for $o[\overline{\beta}']$ that satisfies the definition (in order to express this obligation, we need to replace all occurrences of $o[\overline{\beta}']$ with a variable γ_e, obtaining e'; we cannot quantify over an op, only over a variable).

The above notion of op definition is not fully satisfactory for two reasons. One is that an op declaration immediately followed by an op definition does not necessarily constitute a definitional extension (in the logical sense) of a context: given a consistent context that declares a type and constrains it to be empty (via a suitable axiom), a declaration and definition of an op of that type yields an extended context that is inconsistent. The other reason is that the notion allows vacuous definitions such as declaring two ops f and g of the same type and then defining f to be equal to g and g to be equal to f.

Those two shortcomings are overcome by the following notion of op definition, currently absent from [1] but more common in the literature. The form is

$$\mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1, \dots, o_n : \{\overline{\beta}_n\}\ T_n = e_1, \dots, e_n$$

where $\overline{o} \in \mathcal{N}^{(+)}$, $\overline{\overline{\beta}} \in (\mathcal{N}^{(*)})^+$, $\overline{T} \in \mathit{Type}^+$, $\overline{e} \in \mathit{Exp}^+$, and, for every i, every $o_i[\overline{S}]$ occurring in $\overline{e}$ is such that $\overline{S} = \overline{\beta}_i$. We introduce the abbreviation

$$\mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1, \dots, o_n : \{\overline{\beta}_n\}\ T_n = e_1, \dots, e_n \quad \longrightarrow \quad \begin{array}{l} \mathsf{lem}\ \{\overline{\beta}\}\ \exists!\, \gamma_{\overline{e}} : \prod_i T_i.\ \gamma_{\overline{e}} \equiv \langle \overline{e}' \rangle \\ \mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1 \\ \vdots \\ \mathsf{op}\ o_n : \{\overline{\beta}_n\}\ T_n \\ \mathsf{ax}\ \{\overline{\beta}_1\}\ o_1[\overline{\beta}_1] \equiv e_1 \\ \vdots \\ \mathsf{ax}\ \{\overline{\beta}_n\}\ o_n[\overline{\beta}_n] \equiv e_n \end{array}$$

where $\gamma_{\overline{e}}$ is, for every $\overline{e} \in \mathit{Exp}^+$, a fixed but unspecified name in $\mathcal{N}$ such that $\gamma_{\overline{e}} \notin \bigcup_i \mathcal{FV}(e_i)$, where e'_i is the result of replacing each $o_j[\overline{\beta}_j]$ with $\gamma_{\overline{e}}.\pi_j$ in e_i, and where $\overline{\beta} \in \mathcal{N}^{(*)}$ is such that $\overline{\beta} = \bigcup_i \overline{\beta}_i$.

The last abbreviation introduces $n \geq 1$ ops via definitional extension. The proof obligation expressed by the lemma says that there are unique values of $o_1[\overline{\beta}_1], \dots, o_n[\overline{\beta}_n]$ that satisfy the (in general, mutually recursive) defining equations.

6.7 Op constraints

An op constraint, currently absent from [1] (but present e.g. in [4]), has the form

$$\mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1, \dots, o_n : \{\overline{\beta}_n\}\ T_n :: e$$

where $\overline{o} \in \mathcal{N}^{(+)}$, $\overline{\overline{\beta}} \in (\mathcal{N}^{(*)})^+$, $\overline{T} \in \mathit{Type}^+$, $e \in \mathit{Exp}$, and, for every i, every $o_i[\overline{S}]$ occurring in e is such that $\overline{S} = \overline{\beta}_i$. We introduce the abbreviation

$$\mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1, \dots, o_n : \{\overline{\beta}_n\}\ T_n :: e \quad \longrightarrow \quad \begin{array}{l} \mathsf{lem}\ \{\overline{\beta}\}\ \exists \gamma_e : \prod_i T_i.\ e' \\ \mathsf{op}\ o_1 : \{\overline{\beta}_1\}\ T_1 \\ \vdots \\ \mathsf{op}\ o_n : \{\overline{\beta}_n\}\ T_n \\ \mathsf{ax}\ \{\overline{\beta}\}\ e \end{array}$$

where γ_e is, for every $e \in \mathit{Exp}$, a fixed but unspecified name in $\mathcal{N}$ such that $\gamma_e \notin \mathcal{FV}(e)$, where e' is the result of replacing each $o_i[\overline{\beta}_i]$ with $\gamma_e.\pi_i$ in e, and where $\overline{\beta} \in \mathcal{N}^{(*)}$ is such that $\overline{\beta} = \bigcup_i \overline{\beta}_i$.

This abbreviation introduces $n \geq 1$ ops via conservative extension. The proof obligation expressed by the lemma says that there exists at least one solution to the axiom (there may be more than one). The op declarations introduce the ops and the axiom expresses constraints on them.

6.8 Sum types

A sum type definition has the form

$$\mathsf{def}\ \tau[\overline{\beta}] = c_1\ T_1^\circ : g_1 + \cdots + c_n\ T_n^\circ : g_n$$

where $\tau \in \mathcal{N}$, $\overline{\beta} \in \mathcal{N}^{(*)}$, $\overline{c} \in \mathcal{N}^{(+)}$, $\overline{g} \in \mathcal{N}^{(+)}$, $\overline{c} \cap \overline{g} = \emptyset$, and $\overline{T^\circ} \in (\mathit{Type} + \{\mathsf{noty}\})^+$. We introduce the abbreviation

$$\mathsf{def}\ \tau[\overline{\beta}] = \textstyle\sum_i c_i\ T_i^\circ : g_i$$

$$\longrightarrow$$

$$\vdots$$

$$\mathsf{op}\ c_i : \{\overline{\beta}\} \begin{cases} T_i^\circ \to \tau[\overline{\beta}] & \text{if} \quad T_i^\circ \neq \mathsf{noty} \\ \tau[\overline{\beta}] & \text{otherwise} \end{cases}$$

$$\vdots$$

$$\begin{aligned}
&\mathsf{ax}\ \{\overline{\beta}\}\ \textstyle\bigwedge_{T_i^\circ \neq \mathsf{noty}} (\forall \gamma : T_i^\circ, \gamma' : T_i^\circ.\ c_i\ \gamma \equiv c_i\ \gamma' \Rightarrow \gamma \equiv \gamma') \\
&\mathsf{ax}\ \{\overline{\beta}\}\ \forall \gamma : \tau[\overline{\beta}].\ \left(\textstyle\bigvee_{T_i^\circ = \mathsf{noty}} \gamma \equiv c_i\right) \vee \left(\textstyle\bigvee_{T_i^\circ \neq \mathsf{noty}} \exists \gamma' : T_i^\circ.\ \gamma \equiv c_i\ \gamma'\right) \\
&\mathsf{ax}\ \{\overline{\beta}\}\ \left(\textstyle\bigwedge_{i<j, T_i^\circ = T_j^\circ = \mathsf{noty}} c_i \not\equiv c_j\right) \\
&\qquad \wedge \left(\textstyle\bigwedge_{i<j, T_i^\circ \neq \mathsf{noty}, T_j^\circ \neq \mathsf{noty}} \forall \gamma : T_i^\circ, \gamma' : T_j^\circ.\ c_i\ \gamma \not\equiv c_j\ \gamma'\right) \\
&\qquad \wedge \left(\textstyle\bigwedge_{i<j, T_i^\circ = \mathsf{noty}, T_j^\circ \neq \mathsf{noty}} \forall \gamma : T_j^\circ.\ c_i \not\equiv c_j\ \gamma\right) \\
&\qquad \wedge \left(\textstyle\bigwedge_{i<j, T_i^\circ \neq \mathsf{noty}, T_j^\circ = \mathsf{noty}} \forall \gamma : T_i^\circ.\ c_i\ \gamma \not\equiv c_j\right)
\end{aligned}$$

$$\vdots$$

$$\mathsf{op}\ g_i : \{\overline{\beta}\}\ \tau[\overline{\beta}] \to \mathsf{Bool} = \lambda \gamma : \tau[\overline{\beta}].\ \begin{cases} \exists \gamma' : T_i^\circ.\ \gamma \equiv c_i\ \gamma' & \text{if} \quad T_i^\circ \neq \mathsf{noty} \\ \gamma \equiv c_i & \text{otherwise} \end{cases}$$

$$\vdots$$

Each summand of the sum type $\tau[\overline{\beta}]$ consists of a constructor c_i, an optional type T_i°, and a recognizer g_i. Each constructor c_i is an op that returns a value of the sum type; if the optional type is absent (i.e. $T_i^\circ = \mathsf{noty}$), the constructor is a constant of the sum type; otherwise, it is a function from T_i° to the sum type. The first axiom says that each constructor that has an associated type is an injective function (if it has no associated type, it is not a function but a constant). The second axiom says that every value of the sum type is either equal to a constructor without type (constant) or is in the image of a constructor with type (function). The third axiom says that different constructors yield different values of the sum type; it involves all the pairs of distinct constructors c_i and c_j once (via the condition $i < j$) and contains four kinds of conjuncts, depending on whether c_i and c_j have types or not. Recognizers are defined ops that decide whether a value of the sum type has been constructed with the corresponding constructors.

Note that in order for a sum type definition to maintain the well-formedness of the context to which it is appended, it is necessary that the type name τ is declared in the context with arity $|\overline{\beta}|$.

This treatment of sum types is somewhat different from [1], but [1] will soon change to have the same treatment given here.

6.9 Quotient types

We first introduce the abbreviation

$$\begin{aligned}
\mathsf{equiv}_T \quad \longrightarrow \quad \lambda \psi : T \times T \to \mathsf{Bool}.\ & (\forall \gamma : T.\ \psi\ \langle \gamma, \gamma \rangle) \\
& \wedge (\forall \gamma : T, \gamma' : T.\ \psi\ \langle \gamma, \gamma' \rangle \Rightarrow \psi\ \langle \gamma', \gamma \rangle) \\
& \wedge (\forall \gamma : T, \gamma' : T, \gamma'' : T.\ \psi\ \langle \gamma, \gamma' \rangle \wedge \psi\ \langle \gamma', \gamma'' \rangle \Rightarrow \psi\ \langle \gamma, \gamma'' \rangle)
\end{aligned}$$

where γ'' is a fixed but unspecified in name in $\mathcal{N}$ that is distinct from γ, γ', and ψ. For every $T \in \mathit{Type}$, equiv_T is a predicate that says whether a binary predicate over T is an equivalence, i.e. it is reflexive, symmetric, and transitive.

A quotient type definition has the form

$$\mathsf{def}\ \tau[\overline{\beta}] = T/\mathit{eq} : \mathit{quo} : \mathit{ch}$$

where $\tau \in \mathcal{N}$, $\overline{\beta} \in \mathcal{N}^{(*)}$, $T \in \mathit{Type}$, $\mathit{eq} \in \mathit{Exp}$, $\mathcal{FV}(\mathit{eq}) = \emptyset$, $\mathit{quo} \in \mathcal{N}$, $\mathit{ch} \in \mathcal{N}$, and $\mathit{quo} \neq \mathit{ch}$. We introduce the abbreviation

$$\begin{array}{l}
\mathsf{def}\ \tau[\overline{\beta}] = T/\mathit{eq} : \mathit{quo} : \mathit{ch} \\
\qquad \longrightarrow \\
\mathsf{lem}\ \{\overline{\beta}\}\ \mathsf{equiv}_T\ \mathit{eq} \\
\mathsf{op}\ \mathit{quo} : \{\overline{\beta}\}\ T \to \tau[\overline{\beta}] \\
\mathsf{ax}\ \{\overline{\beta}\}\ \forall \gamma : \tau[\overline{\beta}].\ \exists \gamma' : T.\ \mathit{quo}\ \gamma' \equiv \gamma \\
\mathsf{ax}\ \{\overline{\beta}\}\ \forall \gamma : T, \gamma' : T.\ \mathit{eq}\ \langle \gamma, \gamma' \rangle \Leftrightarrow \mathit{quo}\ \gamma \equiv \mathit{quo}\ \gamma' \\
\mathsf{op}\ \mathit{ch} : \{\overline{\beta}, \beta\}\ T' \to \tau[\overline{\beta}] \to \beta = \lambda \phi : T'.\ \lambda \gamma : \tau[\overline{\beta}].\ \mathsf{let}_{\tau[\overline{\beta}],\beta}\ \gamma' : T.\ \mathit{quo}\ \gamma' \leftarrow \gamma\ \mathsf{in}\ \phi\ \gamma'
\end{array}$$

where ϕ is a fixed but unspecified name in $\mathcal{N}$ that is distinct from γ and γ', where $\beta \notin \overline{\beta}$, and where $T' = (T \to \beta) | \lambda \phi : T \to \beta.\ (\forall \gamma : T, \gamma' : T.\ \mathit{eq}\ \langle \gamma, \gamma' \rangle \Rightarrow \phi\ \gamma \equiv \phi\ \gamma')$.

The expression eq must denote an equivalence relation over the type T, as required by the lemma. The quotienter quo maps each value of T to its equivalence class in $\tau[\overline{\beta}]$. This interpretation is enforced by the two axioms. One requires that quo is surjective, i.e. that every value in the quotient type $\tau[\overline{\beta}]$ is the image of some value in the base type T. The other axiom says that quo maps equivalent values to the same value (their equivalence class), and non-equivalent values to different equivalence classes.

The chooser ch lifts a function over the base type to a function over the quotient type. In order for the lifting to make sense, the function over the base type must be invariant under eq, i.e. it must map equivalent values to the same value. This constraint is expressed by the restriction type T'. The value of the lifted function over an equivalence class is then the value of the base function over any element of the equivalent class (they all give rise to the same value). Besides $\overline{\beta}$, op ch is polymorphic over the range β of the function to be lifted.

A Proofs

Proof of Theorem 4.1

Since

$$\begin{array}{l}
\mathcal{CV}(e,u) = \emptyset \\
\qquad \Rightarrow \\
\mathcal{FV}(d) \cap \mathcal{CV}(e,u) = \emptyset \\
\qquad \Rightarrow \quad [\text{definition of } \mathit{OKsbs}] \\
\mathit{OKsbs}(e,u,d)
\end{array}$$

we are left to prove

$$u \notin \mathcal{FV}(e) \ \Rightarrow\ \mathcal{CV}(e,u) = \emptyset\ \wedge\ e[u/d] = e$$

which we do by induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

$$\mathcal{CV}(e,u) = \emptyset \qquad [\text{definition of } \mathcal{CV}]$$

$$e[u/d] = e \qquad [\text{definition of } \_[\_/\_]]$$

v)

1. $\mathcal{CV}(v,u) = \emptyset$ [definition of $\mathcal{CV}$]

2. $u \notin \mathcal{FV}(v)$ [hypothesis]
3. $u \neq v$ [2, $\mathcal{FV}(v) = \{v\}$]
4. $v[u/d] = v$ [definition of $\_[\_/\_]$, 3]

$e_1\ e_2$)

1. $u \notin \mathcal{FV}(e_1\ e_2)$ [hypothesis]
2. $u \notin \mathcal{FV}(e_1)\ \wedge\ u \notin \mathcal{FV}(e_2)$ [1, $\mathcal{FV}(e_1\ e_2) = \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2)$]
3. $\mathcal{CV}(e_1\ e_2, u)$
   $=$ [definition of $\mathcal{CV}$]
   $\mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u)$
   $=$ [induction hypothesis, 2]
   $\emptyset$
4. $(e_1\ e_2)[u/d]$
   $=$ [definition of $\_[\_/\_]$]
   $e_1[u/d]\ e_2[u/d]$
   $=$ [induction hypothesis, 2]
   $e_1\ e_2$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)

1. $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$ [hypothesis]
2. $u \notin \mathcal{FV}(e) - \{v\}$ [1, $\mathcal{FV}(\lambda v\!:\!T.\ e) = \mathcal{FV}(e) - \{v\}$]
3. $\mathcal{CV}(\lambda v\!:\!T.\ e, u) = \emptyset$ [definition of $\mathcal{CV}$, 2]

If $u = v$:

4. $(\lambda v\!:\!T.\ e)[u/d] = \lambda v\!:\!T.\ e$ [definition of $\_[\_/\_]$]

If $u \neq v$:

4. $u \notin \mathcal{FV}(e)$ [2]
5. $(\lambda v\!:\!T.\ e)[u/d]$
   $=$ [definition of $\_[\_/\_]$]
   $\lambda v\!:\!T.\ e[u/d]$
   $=$ [induction hypothesis, 4]
   $\lambda v\!:\!T.\ e$

Proof of Theorem 4.2

By induction on e:

v, $o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

1. $\mathcal{CV}(e, u) = \emptyset$ [definition of $\mathcal{CV}$]
2. $u \notin \mathcal{CV}(e, u)$ [1]

$e_1\ e_2$)

1. $\mathcal{CV}(e_1\ e_2, u) = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u)$ [definition of $\mathcal{CV}$]
2. $u \notin \mathcal{CV}(e_1, u)\ \wedge\ u \notin \mathcal{CV}(e_2, u)$ [induction hypothesis]
3. $u \notin \mathcal{CV}(e_1\ e_2, u)$ [1, 2]

$e_1 \equiv e_2$, if e_0 e_1 e_2)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u \notin \mathcal{FV}(e) - \{v\}$:

1. $\mathcal{CV}(\lambda v\!:\!T.\ e, u) = \emptyset$ [definition of $\mathcal{CV}$]
2. $u \notin \mathcal{CV}(\lambda v\!:\!T.\ e, u)$ [1]

If $u \in \mathcal{FV}(e) - \{v\}$:

1. $\mathcal{CV}(\lambda v\!:\!T.\ e, u) = \{v\} \cup \mathcal{CV}(e, u)$ [definition of $\mathcal{CV}$]
2. $u \notin \mathcal{CV}(e, u)$ [induction hypothesis]
3. $u \neq v$ [$u \in \mathcal{FV}(e) - \{v\}$]
4. $u \notin \mathcal{CV}(\lambda v\!:\!T.\ e, u)$ [1, 2, 3]

Proof of Theorem 4.3

We prove $OKsbs(e, u, u)$ as follows:

1. $u \notin \mathcal{CV}(e, u)$ [Theorem 4.2]
2. $\{u\} \cap \mathcal{CV}(e, u) = \emptyset$ [1]
3. $OKsbs(e, u, u)$ [definition of $OKsbs$, 2]

We prove $e[u/u] = e$ by induction on e:

$o[\overline{T}]$, ι_T, proj f)

$e[u/u] = e$ [definition of $\_[\_/\_]$]

v)
If $v \neq u$:

$v[u/u] = v$ [definition of $\_[\_/\_]$]

If $v = u$:

$v[u/u]$
$\quad = \ $ [definition of $\_[\_/\_]$]
u
$\quad = \ $ [$v = u$]
v

$e_1\ e_2$)

$(e_1\ e_2)[u/u]$
$\quad = \ $ [definition of $\_[\_/\_]$]
$e_1[u/u]\ e_2[u/u]$
$\quad = \ $ [induction hypothesis]
$e_1\ e_2$

$e_1 \equiv e_2$, if e_0 e_1 e_2)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u = v$:

$(\lambda v\!:\!T.\ e)[u/u] = \lambda v\!:\!T.\ e$ [definition of $\_[\_/\_]$]

If $u \neq v$:

$(\lambda v{:}T.\ e)[u/u]$
$\quad = \ $ [definition of $\_[\_/\_]$]
$\lambda v{:}T.\ e[u/u]$
$\quad = \ $ [induction hypothesis]
$\lambda v{:}T.\ e$

Proof of Theorem 4.4

1. $\mathit{OKsbs}(e_1\ e_2, u, d)$ [hypothesis]
2. $\mathcal{FV}(d) \cap \mathcal{CV}(e_1\ e_2, u) = \emptyset$ [1, definition of OKsbs]
3. $\mathcal{FV}(d) \cap \mathcal{CV}(e_1, u) = \emptyset\ \wedge\ \mathcal{FV}(d) \cap \mathcal{CV}(e_2, u) = \emptyset$ [2, $\mathcal{CV}(e_1\ e_2, u) = \mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u)$]
4. $\mathit{OKsbs}(e_1, u, d)\ \wedge\ \mathit{OKsbs}(e_2, u, d)$ [3, definition of OKsbs]

Proof of Theorem 4.5

Analogous to Theorem 4.4.

Proof of Theorem 4.6

Analogous to Theorem 4.4.

Proof of Theorem 4.7

If $u \notin \mathcal{FV}(e)$:

$\mathit{OKsbs}(e, u, d)$ [Theorem 4.1]

If $u \in \mathcal{FV}(e)$:

1. $\mathit{OKsbs}(\lambda v{:}T.\ e, u, d)$ [hypothesis]
2. $\mathcal{FV}(d) \cap \mathcal{CV}(\lambda v{:}T.\ e, u) = \emptyset$ [1, definition of OKsbs]
3. $u \in \mathcal{FV}(e) - \{v\}$ [$u \neq v$, $u \in \mathcal{FV}(e)$]
4. $\mathcal{FV}(d) \cap (\{v\} \cup \mathcal{CV}(e, u)) = \emptyset$ [definition of $\mathcal{CV}$, 3, 2]
5. $\mathcal{FV}(e) \cap \mathcal{CV}(e, u) = \emptyset$ [4]
6. $\mathit{OKsbs}(e, u, d)$ [definition of OKsbs, 5]

Proof of Theorem 4.8

By induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)
$\quad \mathcal{FV}(e) = \emptyset\ \Rightarrow\ u \notin \mathcal{FV}(e)$

v)

$\mathcal{FV}(v[u/d])$
$\quad = \;$ [definition of $\_[\_/\_]$, $u \in \mathcal{FV}(v) \;\Rightarrow\; u = v$]
$\mathcal{FV}(d)$
$\quad =$
$\emptyset \cup \mathcal{FV}(d)$
$\quad =$
$(\{u\} - \{u\}) \cup \mathcal{FV}(d)$
$\quad = \;$ [$u \in \mathcal{FV}(v) \;\Rightarrow\; \mathcal{FV}(v) = \{u\}$]
$(\mathcal{FV}(v) - \{u\}) \cup \mathcal{FV}(d)$

$e_1\ e_2$)

If $u \in \mathcal{FV}(e_1) \;\wedge\; u \in \mathcal{FV}(e_2)$:

$\mathcal{FV}((e_1\ e_2)[u/d])$
$\quad = \;$ [definition of $\_[\_/\_]$, definition of $\mathcal{FV}$]
$\mathcal{FV}(e_1[u/d]) \cup \mathcal{FV}(e_2[u/d])$
$\quad = \;$ [Theorem 4.4, induction hypothesis]
$(\mathcal{FV}(e_1) - \{u\}) \cup \mathcal{FV}(d) \cup (\mathcal{FV}(e_2) - \{u\}) \cup \mathcal{FV}(d)$
$\quad =$
$(\mathcal{FV}(e_1) - \{u\}) \cup (\mathcal{FV}(e_2) - \{u\}) \cup \mathcal{FV}(d)$
$\quad =$
$((\mathcal{FV}(e_1) \cup \mathcal{FV}(e_2)) - \{u\}) \cup \mathcal{FV}(d)$
$\quad =$
$(\mathcal{FV}(e_1\ e_2) - \{u\}) \cup \mathcal{FV}(d)$

If $u \in \mathcal{FV}(e_1) \;\wedge\; u \notin \mathcal{FV}(e_2)$:

$\mathcal{FV}((e_1\ e_2)[u/d])$
$\quad = \;$ [definition of $\_[\_/\_]$, definition of $\mathcal{FV}$]
$\mathcal{FV}(e_1[u/d]) \cup \mathcal{FV}(e_2[u/d])$
$\quad = \;$ [Theorem 4.4, induction hypothesis, Theorem 4.1]
$(\mathcal{FV}(e_1) - \{u\}) \cup \mathcal{FV}(d) \cup \mathcal{FV}(e_2)$
$\quad = \;$ [$u \notin \mathcal{FV}(e_2) \;\Rightarrow\; \mathcal{FV}(e_2) = \mathcal{FV}(e_2) - \{u\}$]
$(\mathcal{FV}(e_1) - \{u\}) \cup (\mathcal{FV}(e_2) - \{u\}) \cup \mathcal{FV}(d)$
$\quad = \;$ [as above]
$(\mathcal{FV}(e_1\ e_2) - \{u\}) \cup \mathcal{FV}(d)$

If $u \notin \mathcal{FV}(e_1) \wedge u \in \mathcal{FV}(e_2)$, the proof is analogous to the previous one, with e_1 and e_2 swapped.

Finally, $u \notin \mathcal{FV}(e_1) \;\wedge\; u \notin \mathcal{FV}(e_2) \;\Rightarrow\; u \notin \mathcal{FV}(e_1\ e_2)$.

$e_1 \equiv e_2$, if $e_0\ e_1\ e_2$)

Analogous to $e_1\ e_2$, using Theorem 4.5 and Theorem 4.6 instead of Theorem 4.4.

$\lambda v\!:\!T.\ e$)

1. $u \in \mathcal{FV}(\lambda v\!:\!T.\ e)$ [hypothesis]
2. $OKsbs(\lambda v\!:\!T.\ e, u, d)$ [hypothesis]
3. $u \neq v$ [1, $\mathcal{FV}(\lambda v\!:\!T.\ e) = \mathcal{FV}(e) - \{v\}$]
4. $OKsbs(e, u, d)$ [Theorem 4.7, 2, 3]
5. $\mathcal{FV}(d) \cap (\{v\} \cup \mathcal{CV}(e, u)) = \emptyset$ [1, 2]
6. $\mathcal{FV}((\lambda v\!:\!T.\ e)[u/d])$
   $\quad = \;$ [3]
   $\mathcal{FV}(\lambda v\!:\!T.\ e[u/d])$
   $\quad =$
   $\mathcal{FV}(e[u/d]) - \{v\}$

$= $ [induction hypothesis, 4, ($1 \Rightarrow\ u \in \mathcal{FV}(e)$)]
$((\mathcal{FV}(e) - \{u\}) \cup \mathcal{FV}(d)) - \{v\}$
$=$
$((\mathcal{FV}(e) - \{u\}) - \{v\}) \cup (\mathcal{FV}(d) - \{v\})$
$=$
$((\mathcal{FV}(e) - \{v\}) - \{u\}) \cup (\mathcal{FV}(d) - \{v\})$
$= $ [definition of $\mathcal{FV}$]
$(\mathcal{FV}(\lambda v\!:\!T.\ e) - \{u\}) \cup (\mathcal{FV}(d) - \{v\})$
$= $ [$5 \Rightarrow\ v \notin \mathcal{FV}(d)$]
$(\mathcal{FV}(\lambda v\!:\!T.\ e) - \{u\}) \cup \mathcal{FV}(d)$

Proof of Theorem 4.9

If $u \notin \mathcal{FV}(e)$:

1. $e[u/d] = e$ [Theorem 4.1]
2. $u \notin \mathcal{FV}(e[u/d])$ [1, hypothesis $u \notin \mathcal{FV}(e)$]

If $u \in \mathcal{FV}(e)$:

1. $\mathcal{FV}(e[u/d]) = (\mathcal{FV}(e) - \{u\}) \cup \mathcal{FV}(d)$ [Theorem 4.8, hypothesis $OKsbs(e, u, d)$]
2. $u \notin \mathcal{FV}(e) - \{u\}$
3. $u \notin \mathcal{FV}(d)$ [hypothesis]
4. $u \notin \mathcal{FV}(e[u/d])$ [1, 2, 3]

Proof of Theorem 4.10

By induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

$e[u/d][u'/d'] = e[u'/d'] = e = e[u/d] = e[u'/d'][u/d]$ [definition of $\_[\_/\_]$]

u)

$u[u/d][u'/d']$
$= $ [definition of $\_[\_/\_]$]
$d[u'/d']$
$= $ [Theorem 4.1, hypothesis $u' \notin \mathcal{FV}(d)$]
d
$= $ [definition of $\_[\_/\_]$]
$u[u/d]$
$= $ [definition of $\_[\_/\_]$, hypothesis $u \neq u'$]
$u[u'/d'][u/d]$

u')
Symmetric to previous proof.

$v \notin \{u, u'\}$)

$v[u/d][u'/d'] = v[u'/d'] = v = v[u/d] = v[u'/d'][u/d]$ [definition of $\_[\_/\_]$]

$e_1\ e_2$)

$(e_1\ e_2)[u/d][u'/d']$
$\quad =$
$e_1[u/d][u'/d']\ e_2[u/d][u'/d']$
$\quad =$ [induction hypothesis]
$e_1[u'/d'][u/d]\ e_2[u'/d'][u/d]$
$\quad =$
$(e_1\ e_2)[u'/d'][u/d]$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v{:}T.\ e$)
If $v = u$:

1. $v \neq u'$ [hypothesis $u \neq u'$]
2. $(\lambda v{:}T.\ e)[u/d][u'/d']$
$\quad =$ $[v = u]$
$(\lambda v{:}T.\ e)[u'/d']$
$\quad =$ [1]
$\lambda v{:}T.\ e[u'/d']$
$\quad =$ $[v = u]$
$(\lambda v{:}T.\ e[u'/d'])[u/d]$
$\quad =$ [1]
$(\lambda v{:}T.\ e)[u'/d'][u/d]$

If $v = u'$, the proof is symmetric to the previous one.

If $v \notin \{u, u'\}$:

$(\lambda v{:}T.\ e)[u/d][u'/d']$
$\quad =$
$\lambda v{:}T.\ e[u/d][u'/d']$
$\quad =$ [induction hypothesis]
$\lambda v{:}T.\ e[u'/d'][u/d]$
$\quad =$
$(\lambda v{:}T.\ e)[u'/d'][u/d]$

Proof of Theorem 4.11

By induction on e:

u)

$u[u/d][u/d'] = d[u/d'] = u[u/d[u/d']]$ [definition of _[_/_]]

$v \neq u$)

$v[u/d][u/d'] = v = v[u/d[u/d']]$ [definition of _[_/_]]

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$) :
Similar to case $v \neq u$ above.

$e_1\ e_2$)

$(e_1\ e_2)[u/d][u/d']$
$\quad =$ [definition of _[_/_]]
$e_1[u/d][u/d']\ e_2[u/d][u/d']$
$\quad =$ [induction hypothesis]
$e_1[u/d[u/d']]\ e_2[u/d[u/d']]$

$= $ [definition of $\_[\_/\_]$]
$(e_1\ e_2)[u/d[u/d']]$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $v = u$:

$$(\lambda v\!:\!T.\ e)[u/d][u/d'] = \lambda v\!:\!T.\ e = (\lambda v\!:\!T.\ e)[u/d[u/d']] \qquad \text{[definition of } \_[\_/\_]\text{]}$$

If $v \neq u$:

$(\lambda v\!:\!T.\ e)[u/d][u/d']$
$= $ [definition of $\_[\_/\_]$]
$\lambda v\!:\!T.\ e[u/d][u/d']$
$= $ [induction hypothesis]
$\lambda v\!:\!T.\ e[u/d[u/d']]$
$= $ [definition of $\_[\_/\_]$]
$(\lambda v\!:\!T.\ e)[u/d[u/d']]$

Proof of Theorem 4.12

If $u = u'$:

$$e[u/u'][u/u'] = e = e[u/u'] \qquad \text{[Theorem 4.3]}$$

If $u \neq u'$, the proof is by induction on e:

u)

$$u[u/u'][u/u'] = u' = u[u/u'] \qquad \text{[definition of } \_[\_/\_]\text{]}$$

$v \neq u$)

$$v[u/u'][u/u'] = v = v[u/u'] \qquad \text{[definition of } \_[\_/\_]\text{]}$$

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$) :

$$e[u/u'][u/u'] = e = e[u/u'] \qquad \text{[Theorem 4.1]}$$

$e_1\ e_2$)

$$(e_1\ e_2)[u/u'][u/u'] = e_1[u/u'][u/u']\ e_2[u/u'][u/u'] = e_1[u/u']\ e_2[u/u'] = (e_1\ e_2)[u/u']$$
[definition of $\_[\_/\_]$, induction hypothesis]

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u = v$:

$$(\lambda v\!:\!T.\ e)[u/u'][u/u'] = \lambda v\!:\!T.\ e = (\lambda v\!:\!T.\ e)[u/u'] \qquad \text{[definition of } \_[\_/\_]\text{]}$$

If $u \neq v$:

$$(\lambda v\!:\!T.\ e)[u/u'][u/u'] = \lambda v\!:\!T.\ e[u/u'][u/u'] = \lambda v\!:\!T.\ e[u/u'] = (\lambda v\!:\!T.\ e)[u/u']$$
[definition of $\_[\_/\_]$, induction hypothesis]

Proof of Theorem 4.13

By induction on e:

w)

$$\begin{aligned}
& w[w/d][u/u'] \\
& \quad = \text{[definition of \_[\_/\_]]} \\
& d[u/u'] \\
& \quad = \text{[Theorem 4.12]} \\
& d[u/u'][u/u'] \\
& \quad = \text{[definition of \_[\_/\_]]} \\
& w[w/d[u/u']][u/u']
\end{aligned}$$

$v \neq w$)

$$v[w/d][u/u'] = v[u/u'] = v[w/d[u/u']][u/u'] \qquad \text{[definition of \_[\_/\_]]}$$

$o[\overline{T}]$, ι_T, $\textsf{proj}\ f$) :

$$e[w/d][u/u'] = e = e[w/d[u/u']][u/u'] \qquad \text{[Theorem 4.1]}$$

$e_1\ e_2$)

1. $\mathit{OKsbs}(e_1, w, d)\ \wedge\ \mathit{OKsbs}(e_2, w, d)$ [Theorem 4.4, hypothesis]
2. $(e_1\ e_2)[w/d][u/u']$
   $= \text{[definition of \_[\_/\_]]}$
   $e_1[w/d][u/u']\ e_2[w/d][u/u']$
   $= \text{[induction hypothesis, 1]}$
   $e_1[w/d[u/u']][u/u']\ e_2[w/d[u/u']][u/u']$
   $= \text{[definition of \_[\_/\_]]}$
   $(e_1\ e_2)[w/d[u/u']][u/u']$

$e_1 \equiv e_2$, $\textsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $w \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$:

$$(\lambda v\!:\!T.\ e)[w/d][u/u'] = (\lambda v\!:\!T.\ e)[u/u'] = (\lambda v\!:\!T.\ e)[w/d[u/u']][u/u']$$
[definition of _[_/_], Theorem 4.1]

If $w \in \mathcal{FV}(\lambda v\!:\!T.\ e)$:

1. $w \in \mathcal{FV}(e) - \{v\}$ [definition of _[_/_]]
2. $w \neq v$ [1]

If $u \notin \mathcal{FV}(d)$:

3. $(\lambda v\!:\!T.\ e)[w/d][u/u']$
   $= \text{[definition of \_[\_/\_], 2]}$
   $(\lambda v\!:\!T.\ e[w/d])[u/u']$
   $= \text{[hypothesis } u \notin \mathcal{FV}(d)\text{, Theorem 4.1]}$
   $(\lambda v\!:\!T.\ e[w/d[u/u']])[u/u']$
   $= \text{[definition of \_[\_/\_], 2]}$
   $(\lambda v\!:\!T.\ e)[w/d[u/u']][u/u']$

If $u \in \mathcal{FV}(d)$:

3. $\mathcal{FV}(d) \cap \mathcal{CV}(\lambda v\!:\!T.\ e, w) = \emptyset$ [hypothesis $\mathit{OKsbs}(\lambda v\!:\!T.\ e, w, d)$]

4. $\mathcal{CV}(\lambda v\!:\!T.\ e, w) = \{v\} \cup \mathcal{CV}(e, w)$ [definition of $\mathcal{CV}$, 1]
5. $v \notin \mathcal{FV}(d)$ [3, 4]
6. $v \neq u$ [5, hypothesis $u \in \mathcal{FV}(d)$]
7. $OKsbs(e, w, d)$ [Theorem 4.7, hypothesis $OKsbs(\lambda v\!:\!T.\ e, w, d)$, 2]
8. $(\lambda v\!:\!T.\ e)[w/d][u/u']$
   $=$ [definition of $\_[\_/\_]$, 2, 6]
   $\lambda v\!:\!T.\ e[w/d][u/u']$
   $=$ [induction hypothesis, 7]
   $\lambda v\!:\!T.\ e[w/d[u/u']][u/u']$
   $=$ [definition of $\_[\_/\_]$, 2, 6]
   $(\lambda v\!:\!T.\ e)[w/d[u/u']][u/u']$

Proof of Theorem 4.14

By induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)
$e[u/u'][u'/u] = e[u'/u] = e$

u)
$u[u/u'][u'/u] = u'[u'/u] = u$

u')
$u' \in \mathcal{FV}(u')$

$v \notin \{u, u'\}$)
$v[u/u'][u'/u] = v[u'/u] = v$

$e_1\ e_2$)

1. $OKsbs(e_1\ e_2, u, u')$ [hypothesis]
2. $u' \notin \mathcal{FV}(e_1\ e_2)$ [hypothesis]
3. $OKsbs(e_1, u, u')\ \wedge\ OKsbs(e_1, u, u')$ [Theorem 4.4, 1]
4. $u' \notin \mathcal{FV}(e_1)\ \wedge\ u' \notin \mathcal{FV}(e_2)$ [2]
5. $(e_1\ e_2)[u/u'][u'/u]$
   $=$
   $e_1[u/u'][u'/u]\ e_2[u/u'][u'/u]$
   $=$ [induction hypothesis, 3, 4]
   $e_1\ e_2$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$:

$(\lambda v\!:\!T.\ e)[u/u'][u'/u]$
$=$ [Theorem 4.1, hypothesis $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
$(\lambda v\!:\!T.\ e)[u'/u]$
$=$ [Theorem 4.1, hypothesis $u' \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
$\lambda v\!:\!T.\ e$

If $u \in \mathcal{FV}(\lambda v\!:\!T.\ e)$:

1. $u \in \mathcal{FV}(e) - \{v\}$ [$\mathcal{FV}(\lambda v\!:\!T.\ e) = \mathcal{FV}(e) - \{v\}$]

2. $u \neq v$ [1]
3. $\{u'\} \cap (\{v\} \cup \mathcal{CV}(e, u)) = \emptyset$ [hypothesis $OKsbs(\lambda v\!:\!T.\ e, u, u')$, 1]
4. $u' \neq v$ [3]
5. $OKsbs(e, u, u')$ [Theorem 4.7, hypothesis $OKsbs(\lambda v\!:\!T.\ e, u, u')$, 2]
6. $u' \notin \mathcal{FV}(e) - \{v\}$ [hypothesis $u' \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
7. $u' \notin \mathcal{FV}(e)$ [6, 4]
8. $(\lambda v\!:\!T.\ e)[u/u'][u'/u]$
   $=$ [2]
   $(\lambda v\!:\!T.\ e[u/u'])[u'/u]$
   $=$ [4]
   $\lambda v\!:\!T.\ e[u/u'][u'/u]$
   $=$ [induction hypothesis, 5, 7]
   $\lambda v\!:\!T.\ e$

Proof of Theorem 4.15

By induction on e:

$o[\overline{T}]$, ι_T, proj f)
$\mathcal{CV}(e, u) = \emptyset = \mathcal{CV}(e, u') = \mathcal{CV}(e[u/u'], u')$

u)
$\mathcal{CV}(u, u) = \emptyset = \mathcal{CV}(u', u') = \mathcal{CV}(u[u/u'], u')$

$v \neq u$)
$\mathcal{CV}(v, u) = \emptyset = \mathcal{CV}(v, u') = \mathcal{CV}(v[u/u'], u')$

$e_1\ e_2$)

1. $OKsbs(e_1, u, u') \ \wedge\ OKsbs(e_2, u, u')$ [Theorem 4.4, hypothesis $OKsbs(e_1\ e_2, u, u')$]
2. $u' \notin \mathcal{FV}(e_1) \ \wedge\ u' \notin \mathcal{FV}(e_2)$ [hypothesis $u' \notin \mathcal{FV}(e_1\ e_2)$]
3. $\mathcal{CV}(e_1\ e_2, u)$
   $=$
   $\mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u)$
   $=$ [induction hypothesis, 1, 2]
   $\mathcal{CV}(e_1[u/u'], u') \cup \mathcal{CV}(e_2[u/u'], u')$
   $=$
   $\mathcal{CV}(e_1[u/u']\ e_2[u/u'], u')$
   $=$
   $\mathcal{CV}((e_1\ e_2)[u/u'], u')$

$e_1 \equiv e_2$, if $e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$:

$\mathcal{CV}(\lambda v\!:\!T.\ e, u)$
$=$ [hypothesis $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
$\emptyset$
$=$ [hypothesis $u' \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
$\mathcal{CV}(\lambda v\!:\!T.\ e, u')$
$=$ [Theorem 4.1, hypothesis $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
$\mathcal{CV}((\lambda v\!:\!T.\ e)[u/u'], u)$

If $u \in \mathcal{FV}(\lambda v\!:\!T.\ e)$:

1. $u \in \mathcal{FV}(e) - \{v\}$ $\quad [\mathcal{FV}(\lambda v\!:\!T.\ e) = \mathcal{FV}(e) - \{v\}]$
2. $u \neq v$ $\quad [1]$
3. $\{u'\} \cap (\{v\} \cup \mathcal{CV}(e, u)) = \emptyset$ $\quad$ [hypothesis $\mathit{OKsbs}(\lambda v\!:\!T.\ e, u, u')$, 1]
4. $u' \neq v$ $\quad [3]$
5. $\mathit{OKsbs}(e, u, u')$ $\quad$ [Theorem 4.7, hypothesis $\mathit{OKsbs}(\lambda v\!:\!T.\ e, u, u')$, 2]
6. $u' \notin \mathcal{FV}(e) - \{v\}$ $\quad$ [hypothesis $u' \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
7. $u' \notin \mathcal{FV}(e)$ $\quad [6, 4]$
8. $u \in \mathcal{FV}(e)$ $\quad [1]$
9. $\mathcal{FV}(e[u/u']) = (\mathcal{FV}(e) - \{u\}) \cup \{u'\}$ $\quad$ [Theorem 4.8, 5, 8]
10. $u' \in \mathcal{FV}(e[u/u'])$ $\quad [9]$
11. $u' \in \mathcal{FV}(e[u/u']) - \{v\}$ $\quad [10, 4]$
12. $\mathcal{CV}(\lambda v\!:\!T.\ e, u)$
    $= [1]$
    $\{v\} \cup \mathcal{CV}(e, u)$
    $=$ [induction hypothesis, 5, 7]
    $\{v\} \cup \mathcal{CV}(e[u/u'], u')$
    $= [11]$
    $\mathcal{CV}(\lambda v\!:\!T.\ e[u/u'], u')$
    $= [2]$
    $\mathcal{CV}((\lambda v\!:\!T.\ e)[u/u'], u')$

Proof of Theorem 4.16

By induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)
$$e[u/u'] = e \ \Rightarrow\ \mathcal{CV}(e, w) = \mathcal{CV}(e[u/u'], w)$$

u)
$$\mathcal{CV}(u, w) = \emptyset = \mathcal{CV}(u', w) = \mathcal{CV}(u[u/u'], w)$$

$v \neq u$)
$$v[u/u'] = v \ \Rightarrow\ \mathcal{CV}(v, w) = \mathcal{CV}(v[u/u'], w)$$

$e_1\ e_2$)

1. $\mathit{OKsbs}(e_1, u, u') \ \wedge\ \mathit{OKsbs}(e_2, u, u')$ $\quad$ [Theorem 4.4, hypothesis $\mathit{OKsbs}(e_1\ e_2, u, u')$]
2. $\mathcal{CV}(e_1\ e_2, w)$
    $=$
    $\mathcal{CV}(e_1, w) \cup \mathcal{CV}(e_2, w)$
    $=$ [induction hypothesis, 1]
    $\mathcal{CV}(e_1[u/u'], w) \cup \mathcal{CV}(e_2[u/u'], w)$
    $=$
    $\mathcal{CV}(e_1[u/u']\ e_2[u/u'], w)$
    $=$
    $\mathcal{CV}((e_1\ e_2)[u/u'], w)$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)
If $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$:

1. $(\lambda v{:}T.\ e)[u/u'] = \lambda v{:}T.\ e$ [Theorem 4.1, hypothesis $u \notin \mathcal{FV}(\lambda v{:}T.\ e)$]
2. $\mathcal{CV}(\lambda v{:}T.\ e, w) = \mathcal{CV}((\lambda v{:}T.\ e)[u/u'], w)$ [1]

If $u \in \mathcal{FV}(\lambda v{:}T.\ e)$:

1. $\mathcal{CV}(\lambda v{:}T.\ e, w) = \begin{cases} \{v\} \cup \mathcal{CV}(e, w) & \text{if} \quad w \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \text{otherwise} \end{cases}$ [$\mathcal{CV}$]
2. $u \in \mathcal{FV}(\lambda v{:}T.\ e)$ [hypothesis]
3. $u \in \mathcal{FV}(e) - \{v\}$ [2, $\mathcal{FV}$]
4. $u \neq v$ [3]
5. $\mathcal{CV}((\lambda v{:}T.\ e)[u/u'], w) = \mathcal{CV}(\lambda v{:}T.\ e[u/u'], w)$ [4, _[_/_]]
6. $\mathcal{CV}(\lambda v{:}T.\ e[u/u'], w) = \begin{cases} \{v\} \cup \mathcal{CV}(e[u/u'], w) & \text{if} \quad w \in \mathcal{FV}(e[u/u']) - \{v\} \\ \emptyset & \text{otherwise} \end{cases}$ [$\mathcal{CV}$]
7. $\mathit{OKsbs}(\lambda v{:}T.\ e, u, u')$ [hypothesis]
8. $\mathit{OKsbs}(e, u, u')$ [Theorem 4.7, 7, 4]
9. $w \neq u \ \wedge \ w \neq u'$ [hypothesis]
10. $\mathcal{CV}(e, w) = \mathcal{CV}(e[u/u'], w)$ [induction hypothesis, 8, 9]
11. $u \in \mathcal{FV}(e)$ [3]
12. $\mathcal{FV}(e[u/u']) = (\mathcal{FV}(e) - \{u\}) \cup \{u'\}$ [Theorem 4.8, 8, 11]
13. $w \in \mathcal{FV}(e)$
    $\Rightarrow$ [9]
    $w \in \mathcal{FV}(e) - \{u\}$
    $\Rightarrow$
    $w \in (\mathcal{FV}(e) - \{u\}) \cup \{u'\}$
    $\Rightarrow$ [12]
    $w \in \mathcal{FV}(e[u/u'])$
14. $w \in \mathcal{FV}(e[u/u'])$
    $\Rightarrow$ [12]
    $w \in (\mathcal{FV}(e) - \{u\}) \cup \{u'\}$
    $\Rightarrow$ [9]
    $w \in \mathcal{FV}(e) - \{u\}$
    $\Rightarrow$
    $w \in \mathcal{FV}(e)$
15. $w \in \mathcal{FV}(e) \ \Leftrightarrow \ w \in \mathcal{FV}(e[u/u'])$ [13, 14]
16. $\mathcal{CV}(\lambda v{:}T.\ e, w) = \mathcal{CV}((\lambda v{:}T.\ e)[u/u'], w)$ [1, 10, 15, 5]

Proof of Theorem 4.17

If $v \notin \mathcal{FV}(e)$:

$\mathcal{FV}(\lambda v'{:}T.\ e[v/v'])$
  $=$ [Theorem 4.1, hypothesis $v \notin \mathcal{FV}(e)$]
$\mathcal{FV}(\lambda v'{:}T.\ e)$
  $=$ [$\mathcal{FV}$]
$\mathcal{FV}(e) - \{v'\}$
  $=$ [hypothesis $v' \notin \mathcal{FV}(e)$]
$\mathcal{FV}(e)$
  $=$ [hypothesis $v \notin \mathcal{FV}(e)$]
$\mathcal{FV}(e) - \{v\}$
  $=$ [$\mathcal{FV}$]
$\mathcal{FV}(\lambda v{:}T.\ e)$

If $v \in \mathcal{FV}(e)$:

1. $v' \notin \mathcal{CV}(e, v)$ [hypothesis]
2. $\{v'\} \cap \mathcal{CV}(e, v) = \emptyset$ [1]
3. $OKsbs(e, v, v')$ [2, $OKsbs$]
4. $\mathcal{FV}(\lambda v' : T.\ e[v/v'])$
   $=$ [$\mathcal{FV}$]
   $\mathcal{FV}(e[v/v']) - \{v'\}$
   $=$ [Theorem 4.8, hypothesis $v \in \mathcal{FV}(e)$, 3]
   $((\mathcal{FV}(e) - \{v\}) \cup \{v'\}) - \{v'\}$
   $=$
   $\mathcal{FV}(e) - \{v, v'\}$
   $=$ [hypothesis $v' \notin \mathcal{FV}(e)$]
   $\mathcal{FV}(e) - \{v\}$
   $=$ [$\mathcal{FV}$]
   $\mathcal{FV}(\lambda v : T.\ e)$

Proof of Theorem 4.18

By induction on e:

v)

$\mathcal{FV}(v[\sigma]) = \mathcal{FV}(v)$ [definition of _[_]]

$o[\overline{T}]$)

$\mathcal{FV}(o[\overline{T}][\sigma])$
$=$ [definition of _[_]]
$\mathcal{FV}(o[\overline{T}[\sigma]])$
$=$ [definition of $\mathcal{FV}$]
$\emptyset$
$=$ [definition of $\mathcal{FV}$]
$\mathcal{FV}(o[\overline{T}])$

ι_T, $\mathsf{proj}\ f$)
Analogous to $o[\overline{T}]$.

$e_1\ e_2$)

$\mathcal{FV}((e_1\ e_2)[\sigma])$
$=$ [definition of _[_]]
$\mathcal{FV}(e_1[\sigma]\ e_2[\sigma])$
$=$ [definition of $\mathcal{FV}$]
$\mathcal{FV}(e_1[\sigma]) \cup \mathcal{FV}(e_2[\sigma])$
$=$ [induction hypothesis]
$\mathcal{FV}(e_1) \cup \mathcal{FV}(e_1)$
$=$ [definition of $\mathcal{FV}$]
$\mathcal{FV}(e_1\ e_2)$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v : T.\ e$)

$$\begin{array}{l}
\mathcal{FV}((\lambda v\!:\!T.\ e)[\sigma]) \\
\quad = \text{[definition of \_[\_]]} \\
\mathcal{FV}(\lambda v\!:\!T[\sigma].\ e[\sigma]) \\
\quad = \text{[definition of } \mathcal{FV}\text{]} \\
\mathcal{FV}(e[\sigma]) - \{v\} \\
\quad = \text{[induction hypothesis]} \\
\mathcal{FV}(e) - \{v\} \\
\quad = \text{[definition of } \mathcal{FV}\text{]} \\
\mathcal{FV}(\lambda v\!:\!T.\ e)
\end{array}$$

Proof of Theorem 4.19

By induction on e:

v, $o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

$$\mathcal{CV}(e[\sigma], u) = \emptyset = \mathcal{CV}(e, u)$$

$e_1\ e_2$)

$$\begin{array}{l}
\mathcal{CV}((e_1\ e_2)[\sigma], u) \\
\quad = \\
\mathcal{CV}(e_1[\sigma], u) \cup \mathcal{CV}(e_2[\sigma], u) \\
\quad = \text{[induction hypothesis]} \\
\mathcal{CV}(e_1, u) \cup \mathcal{CV}(e_2, u) \\
\quad = \\
\mathcal{CV}(e_1\ e_2, u)
\end{array}$$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)

$$\begin{array}{l}
\mathcal{CV}((\lambda v\!:\!T.\ e)[\sigma], u) \\
\quad = \\
\mathcal{CV}(\lambda v\!:\!T[\sigma].\ e[\sigma], u) \\
\quad = \\
\begin{cases} \{v\} \cup \mathcal{CV}(e[\sigma], u) & \text{if} \quad u \in \mathcal{FV}(e[\sigma]) - \{v\} \\ \emptyset & \text{otherwise} \end{cases} \\
\quad = \text{[Theorem 4.18]} \\
\begin{cases} \{v\} \cup \mathcal{CV}(e[\sigma], u) & \text{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \text{otherwise} \end{cases} \\
\quad = \text{[induction hypothesis]} \\
\begin{cases} \{v\} \cup \mathcal{CV}(e, u) & \text{if} \quad u \in \mathcal{FV}(e) - \{v\} \\ \emptyset & \text{otherwise} \end{cases} \\
\quad = \\
\mathcal{CV}(\lambda v\!:\!T.\ e, u)
\end{array}$$

Proof of Theorem 4.20

By induction on e:

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

1. $\mathcal{FV}(e) = \emptyset$
2. $u \notin \mathcal{FV}(e)$ [1]
3. $e[u/d] = e$ [Theorem 4.1, 2]

4. $\mathcal{FTV}(e[u/d]) = \mathcal{FTV}(e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e) \\ \emptyset & \text{otherwise} \end{cases}$ [3, 2]

u)

$$\begin{array}{l} \mathcal{FTV}(u[u/d]) \\ \quad = \\ \mathcal{FTV}(d) \\ \quad = \;[\mathcal{FTV}(u) = \emptyset] \\ \mathcal{FTV}(u) \cup \mathcal{FTV}(d) \\ \quad = \;[u \in \{u\}] \\ \mathcal{FTV}(u) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(u) \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

$v \neq u$)

$$\begin{array}{l} \mathcal{FTV}(v[u/d]) \\ \quad = \\ \mathcal{FTV}(v) \\ \quad = \\ \emptyset \\ \quad = \;[\mathcal{FTV}(v) = \emptyset \;\wedge\; u \notin \{v\}] \\ \mathcal{FTV}(v) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(v) \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

$e_1\ e_2$)

$$\begin{array}{l} \mathcal{FTV}((e_1\ e_2)[u/d]) \\ \quad = \\ \mathcal{FTV}(e_1[u/d]) \cup \mathcal{FTV}(e_2[u/d]) \\ \quad = \;[\text{induction hypothesis}] \\ \mathcal{FTV}(e_1) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e_1) \\ \emptyset & \text{otherwise} \end{cases} \cup \mathcal{FTV}(e_2) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e_2) \\ \emptyset & \text{otherwise} \end{cases} \\ \quad = \\ \mathcal{FTV}(e_1\ e_2) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e_1) \;\vee\; u \in \mathcal{FV}(e_2) \\ \emptyset & \text{otherwise} \end{cases} \\ \quad = \\ \mathcal{FTV}(e_1\ e_2) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e_1\ e_2) \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

$e_1 \equiv e_2$, if $e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v\!:\!T.\ e$)

$$\begin{array}{l} \mathcal{FTV}((\lambda v\!:\!T.\ e)[u/d]) \\ \quad = \\ \mathcal{FTV}(\begin{cases} \lambda v\!:\!T.\ e & \text{if} \quad u = v \\ \lambda v\!:\!T.\ e[u/d] & \text{otherwise} \end{cases}) \\ \quad = \\ \begin{cases} \mathcal{FTV}(\lambda v\!:\!T.\ e) & \text{if} \quad u = v \\ \mathcal{FTV}(\lambda v\!:\!T.\ e[u/d]) & \text{otherwise} \end{cases} \\ \quad = \;[u = v \;\Rightarrow\; u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)] \\ \begin{cases} \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases} & \text{if} \quad u = v \\ \mathcal{FTV}(T) \cup \mathcal{FTV}(e[u/d]) & \text{otherwise} \end{cases} \\ \quad = \;[\text{induction hypothesis}] \end{array}$$

$$\begin{cases} \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases} & \text{if} \quad u = v \\ \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e) \\ \emptyset & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

$= \ [\mathcal{FTV}(\lambda v\!:\!T.\ e) = \mathcal{FTV}(T) \cup \mathcal{FTV}(e)]$

$$\begin{cases} \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases} & \text{if} \quad u = v \\ \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(e) \\ \emptyset & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

$= \ [u \neq v \ \wedge \ u \in \mathcal{FV}(e) \ \Rightarrow \ u \in \mathcal{FV}(\lambda v\!:\!T.\ e)]$

$$\begin{cases} \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases} & \text{if} \quad u = v \\ \mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

$=$

$$\mathcal{FTV}(\lambda v\!:\!T.\ e) \cup \begin{cases} \mathcal{FTV}(d) & \text{if} \quad u \in \mathcal{FV}(\lambda v\!:\!T.\ e) \\ \emptyset & \text{otherwise} \end{cases}$$

Proof of Theorem 4.21

$\mathcal{FTV}(e[u/u'])$

$= \ [\text{Theorem 4.20}]$

$$\mathcal{FTV}(e) \cup \begin{cases} \mathcal{FTV}(u') & \text{if} \quad u \in \mathcal{FV}(e) \\ \emptyset & \text{otherwise} \end{cases}$$

$= \ [\mathcal{FTV}(u') = \emptyset]$

$\mathcal{FTV}(e)$

Proof of Theorem 4.22

By induction on e:

u)

$u[u/d][\sigma]$

$=$

$d[\sigma]$

$=$

$u[\sigma][u/d[\sigma]]$

$v \neq u$)

$v[u/d][\sigma]$

$=$

v

$=$

$v[\sigma][u/d[\sigma]]$

$o[\overline{T}]$, ι_T, $\mathsf{proj}\ f$)

$e[u/d][\sigma]$

$= \ [\text{Theorem 4.1}]$

$e[\sigma]$

$= \ [\text{Theorem 4.1, Theorem 4.18}]$

$e[\sigma][u/d[\sigma]]$

$e_1\ e_2$)

$(e_1\ e_2)[u/d][\sigma]$
$\quad =$
$e_1[u/d][\sigma]\ e_2[u/d][\sigma]$
$\quad =$ [induction hypothesis]
$e_1[\sigma][u/d[\sigma]]\ e_2[\sigma][u/d[\sigma]]$
$\quad =$
$(e_1\ e_2)[\sigma][u/d[\sigma]]$

$e_1 \equiv e_2$, $\mathsf{if}\ e_0\ e_1\ e_2$)
Analogous to $e_1\ e_2$.

$\lambda v{:}T.\ e$)

$(\lambda v{:}T.\ e)[u/d][\sigma]$
$\quad =$
$\begin{cases} \lambda v{:}T[\sigma].\ e[\sigma] & \text{if} \quad u = v \\ \lambda v{:}T[\sigma].\ e[u/d][\sigma] & \text{otherwise} \end{cases}$
$\quad =$ [induction hypothesis]
$\begin{cases} \lambda v{:}T[\sigma].\ e[\sigma] & \text{if} \quad u = v \\ \lambda v{:}T[\sigma].\ e[\sigma][u/d[\sigma]] & \text{otherwise} \end{cases}$
$\quad =$
$(\lambda v{:}T.\ e)[\sigma][u/d[\sigma]]$

Proof of Theorem 4.23

$e[u/u'][\sigma]$
$\quad =$ [Theorem 4.22]
$e[\sigma][u/u'[\sigma]]$
$\quad =$
$e[\sigma][u/u']$

Proof of Theorem 4.24

Immediate by inspection of the definition of expression substitution in context elements and contexts.

Proof of Theorem 4.25

By induction on x:

β)

$\beta[\emptyset] = \beta$ [definition of $\_[\_]$, $\beta \notin \mathcal{D}(\emptyset)$]

Bool, v, $\mathsf{ty}\ \tau{:}n$, $\mathsf{tvar}\ \beta$, ϵ)

$x[\emptyset] = x$ [definition of $\_[\_]$]

$T_1 \rightarrow T_2$)

$(T_1 \rightarrow T_2)[\emptyset]$
$\quad =$
$T_1[\emptyset] \rightarrow T_2[\emptyset]$
$\quad =$ [induction hypothesis]
$T_1 \rightarrow T_2$

$\mathsf{op}\ o{:}\{\overline{\beta}\}\ T$)

$(\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T)[\emptyset]$
$\quad =$
$\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\emptyset\!\uparrow_{\overline{\beta}}]$
$\quad = \ [\emptyset\!\uparrow_{\overline{\beta}} = \emptyset]$
$\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\emptyset]$
$\quad = \ $[induction hypothesis]
$\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T$

$\mathsf{def}\ \tau[\overline{\beta}] = T$, $\mathsf{ax}\ \{\overline{\beta}\}\ e$, $\mathsf{lem}\ \{\overline{\beta}\}\ e$)
Analogous to $\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T$.

all other kinds)
Analogous to $T_1 \rightarrow T_2$.

Proof of Theorem 4.26

By induction on x:

β)
If $\beta \notin \mathcal{D}(\sigma)$:

$\beta[\sigma]$
$\quad =$
β
$\quad = \ [\beta \notin \mathcal{D}(\emptyset)]$
$\beta[\emptyset]$
$\quad = \ $[hypothesis $\beta \notin \mathcal{D}(\sigma)$]
$\beta[\sigma\!\downarrow_{\{\beta\}}]$
$\quad = \ [\mathcal{FTV}(\beta) = \{\beta\}]$
$\beta[\sigma\!\downarrow_{\mathcal{FTV}(\beta)}]$

If $\beta \in \mathcal{D}(\sigma)$:

$\beta[\sigma]$
$\quad =$
$\sigma(\beta)$
$\quad =$
$\{\langle \beta, \sigma(\beta)\rangle\}(\beta)$
$\quad = \ $[hypothesis $\beta \in \mathcal{D}(\sigma)$]
$\sigma\!\downarrow_{\{\beta\}}(\beta)$
$\quad = \ $[hypothesis $\beta \in \mathcal{D}(\sigma)$]
$\beta[\sigma\!\downarrow_{\{\beta\}}]$
$\quad = \ [\mathcal{FTV}(\beta) = \{\beta\}]$
$\beta[\sigma\!\downarrow_{\mathcal{FTV}(\beta)}]$

Bool, v, $\mathsf{ty}\ \tau\!:\!n$, $\mathsf{tvar}\ \beta$, ϵ)

$x[\sigma]$
$\quad =$
x
$\quad =$
$x[\emptyset]$
$\quad = \ [\mathcal{FTV}(x) = \emptyset]$
$x[\sigma\!\downarrow_{\mathcal{FTV}(x)}]$

$T_1 \rightarrow T_2$)

$$\begin{aligned}
&(T_1 \to T_2)[\sigma] \\
&\quad = \\
&T_1[\sigma] \to T_2[\sigma] \\
&\quad = \ \text{[induction hypothesis]} \\
&T_1[\sigma\!\downarrow_{\mathcal{FTV}(T_1)}] \to T_2[\sigma\!\downarrow_{\mathcal{FTV}(T_2)}] \\
&\quad = \ [\mathcal{FTV}(T_1) \subseteq \mathcal{FTV}(T_1 \to T_2) \ \wedge \ \mathcal{FTV}(T_2) \subseteq \mathcal{FTV}(T_1 \to T_2)] \\
&T_1[\sigma\!\downarrow_{\mathcal{FTV}(T_1 \to T_2)}\!\downarrow_{\mathcal{FTV}(T_1)}] \to T_2[\sigma\!\downarrow_{\mathcal{FTV}(T_1 \to T_2)}\!\downarrow_{\mathcal{FTV}(T_2)}] \\
&\quad = \ \text{[induction hypothesis]} \\
&T_1[\sigma\!\downarrow_{\mathcal{FTV}(T_1 \to T_2)}] \to T_2[\sigma\!\downarrow_{\mathcal{FTV}(T_1 \to T_2)}] \\
&\quad = \\
&(T_1 \to T_2)[\sigma\!\downarrow_{\mathcal{FTV}(T_1 \to T_2)}]
\end{aligned}$$

$\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T)$

$$\begin{aligned}
&(\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T)[\sigma] \\
&\quad = \\
&\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\sigma\!\uparrow_{\overline{\beta}}] \\
&\quad = \ \text{[induction hypothesis]} \\
&\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\sigma\!\uparrow_{\overline{\beta}}\!\downarrow_{\mathcal{FTV}(T)}] \\
&\quad = \ [(\mathcal{D}(\sigma) - \overline{\beta}) \cap \mathcal{FTV}(T) = (\mathcal{D}(\sigma) \cap \mathcal{FTV}(T)) - \overline{\beta}] \\
&\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\sigma\!\downarrow_{\mathcal{FTV}(T)}\!\uparrow_{\overline{\beta}}] \\
&\quad = \ [(\mathcal{D}(\sigma) \cap \mathcal{FTV}(T)) - \overline{\beta} = (\mathcal{D}(\sigma) \cap (\mathcal{FTV}(T) - \overline{\beta})) - \overline{\beta}] \\
&\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T[\sigma\!\downarrow_{\mathcal{FTV}(T) - \overline{\beta}}\!\uparrow_{\overline{\beta}}] \\
&\quad = \ [\mathcal{FTV}(\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T) = \mathcal{FTV}(T) - \overline{\beta}] \\
&(\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T)[\sigma\!\downarrow_{\mathcal{FTV}(\mathsf{op}\ \alpha\{\overline{\beta}\}\ T)}\!\uparrow_{\overline{\beta}}] \\
&\quad = \\
&(\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T)[\sigma\!\downarrow_{\mathcal{FTV}(\mathsf{op}\ \alpha\{\overline{\beta}\}\ T)}]
\end{aligned}$$

$\mathsf{def}\ \tau[\overline{\beta}] = T$, $\mathsf{ax}\ \{\overline{\beta}\}\ e$, $\mathsf{lem}\ \{\overline{\beta}\}\ e)$
Analogous to $\mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T$.

all other kinds)
Analogous to $T_1 \to T_2$.

Proof of Theorem 4.27

$$\begin{aligned}
&x[\sigma] \\
&\quad = \ \text{[Theorem 4.26]} \\
&x[\sigma\!\downarrow_{\mathcal{FTV}(x)}] \\
&\quad = \ [\text{hypothesis } \mathcal{D}(\sigma) \cap \mathcal{FTV}(x) = \emptyset] \\
&x[\emptyset] \\
&\quad = \ \text{[Theorem 4.25]} \\
&x
\end{aligned}$$

Proof of Theorem 4.28

By induction on x:

Bool, $v)$

$$\begin{aligned}
&\mathcal{FTV}(x[\sigma]) \\
&\quad = \\
&\mathcal{FTV}(x) \\
&\quad = \\
&\emptyset
\end{aligned}$$

$=$
$(\emptyset - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \emptyset \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$
$= [\mathcal{FTV}(x) = \emptyset]$
$(\mathcal{FTV}(x) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(x) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$

β)

If $\beta \notin \mathcal{D}(\sigma)$:

$\mathcal{FTV}(\beta[\sigma])$
$=$
$\mathcal{FTV}(\beta)$
$=$
$\{\beta\}$
$= [\beta \notin \mathcal{D}(\sigma)]$
$(\{\beta\} - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \{\beta\} \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$
$= [\mathcal{FTV}(\beta) = \{\beta\}]$
$(\mathcal{FTV}(\beta) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(\beta) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$

If $\beta \in \mathcal{D}(\sigma)$:

$\mathcal{FTV}(\beta[\sigma])$
$=$
$\mathcal{FTV}(\sigma(\beta))$
$= [\beta \in \mathcal{D}(\sigma) \;\Rightarrow\; \mathcal{FTV}(\beta) - \mathcal{D}(\sigma) = \{\beta\} - \mathcal{D}(\sigma) = \emptyset]$
$(\mathcal{FTV}(\beta) - \mathcal{D}(\sigma)) \cup \mathcal{FTV}(\sigma(\beta))$
$=$
$(\mathcal{FTV}(\beta) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \{\beta\}} \mathcal{FTV}(\sigma(\alpha))$
$= [\mathcal{FTV}(\beta) = \{\beta\} \;\wedge\; \beta \in \mathcal{D}(\sigma)]$
$(\mathcal{FTV}(\beta) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(\beta) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$

$T_1 \to T_2$)

$\mathcal{FTV}((T_1 \to T_2)[\sigma])$
$=$
$\mathcal{FTV}(T_1[\sigma]) \cup \mathcal{FTV}(T_2[\sigma])$
$=$ [induction hypothesis]
$(\mathcal{FTV}(T_1) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(T_1) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha)) \cup$
$(\mathcal{FTV}(T_2) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(T_2) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$
$=$
$((\mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2)) - \mathcal{D}(\sigma)) \cup$
$\bigcup_{\alpha \in (\mathcal{FTV}(T_1) \cap \mathcal{D}(\sigma)) \cup (\mathcal{FTV}(T_1) \cap \mathcal{D}(\sigma))} \mathcal{FTV}(\sigma(\alpha))$
$=$
$(\mathcal{FTV}(T_1 \to T_2) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in (\mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2)) \cap \mathcal{D}(\sigma)} \mathcal{FTV}(\sigma(\alpha))$
$=$
$(\mathcal{FTV}(T_1 \to T_2) - \mathcal{D}(\sigma)) \cup \bigcup_{\alpha \in \mathcal{FTV}(T_1 \to T_2) \cap \mathcal{D}(\sigma)} \sigma(\alpha)$

all other kinds)

Analogous to $T_1 \to T_2$.

Proof of Theorem 4.29

By induction on x:

Bool, v)

$x[\sigma][\sigma'] = x = x[\sigma[\sigma']]$

β)

If $\beta \notin \mathcal{D}(\sigma)$:

1. $\beta[\sigma][\sigma'] = \beta[\sigma']$
2. $(\mathcal{FTV}(\beta) - \mathcal{D}(\sigma)) \cap \mathcal{D}(\sigma') = \emptyset$ [hypothesis]
3. $\beta \notin \mathcal{D}(\sigma')$ [2, hypothesis $\beta \notin \mathcal{D}(\sigma)$]
4. $\beta[\sigma'] = \beta$ [3]
5. $\mathcal{D}(\sigma[\sigma']) = \mathcal{D}(\sigma)$ [definition of _[_]]
6. $\beta \notin \mathcal{D}(\sigma[\sigma'])$ [hypothesis $\beta \notin \mathcal{D}(\sigma)$, 5]
7. $\beta[\sigma[\sigma']] = \beta$ [6]
8. $\beta[\sigma][\sigma'] = \beta[\sigma[\sigma']]$ [1, 4, 7]

If $\beta \in \mathcal{D}(\sigma)$:

$\beta[\sigma][\sigma']$
$=$
$\sigma(\beta)[\sigma']$
$=$ [definition of $\sigma[\sigma']$]
$\sigma[\sigma'](\beta)$
$=$ [$\mathcal{D}(\sigma[\sigma']) = \mathcal{D}(\sigma)$]
$\beta[\sigma[\sigma']]$

$T_1 \to T_2$)

1. $\mathcal{FTV}(T_1 \to T_2) = \mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2)$
2. $(\mathcal{FTV}(T_1 \to T_2) - \mathcal{D}(\sigma)) \cap \mathcal{D}(\sigma') = \emptyset$ [hypothesis]
3. $(\mathcal{FTV}(T_1) - \mathcal{D}(\sigma)) \cap \mathcal{D}(\sigma') = (\mathcal{FTV}(T_2) - \mathcal{D}(\sigma)) \cap \mathcal{D}(\sigma') = \emptyset$ [1, 2]
4. $(T_1 \to T_2)[\sigma][\sigma']$
   $=$
   $T_1[\sigma][\sigma'] \to T_2[\sigma][\sigma']$
   $=$ [induction hypothesis, 3]
   $T_1[\sigma[\sigma']] \to T_2[\sigma[\sigma']]$
   $=$
   $(T_1 \to T_2)[\sigma[\sigma']]$

all other kinds)

Analogous to $T_1 \to T_2$.

Proof of Theorem 4.30

By induction on x:

Bool, v)

$x[\sigma][\sigma'] = x[\sigma'][\sigma[\sigma']]$

β)

If $\beta \notin \mathcal{D}(\sigma) \ \wedge \ \beta \notin \mathcal{D}(\sigma')$:

$\beta[\sigma][\sigma']$
$=$
β
$=$ [$\mathcal{D}(\sigma[\sigma']) = \mathcal{D}(\sigma)$]
$\beta[\sigma'][\sigma[\sigma']]$

If $\beta \in \mathcal{D}(\sigma)$:

1. $\beta \notin \mathcal{D}(\sigma')$ [hypothesis $\mathcal{D}(\sigma) \cap \mathcal{D}(\sigma') = \emptyset$]
2. $\beta[\sigma][\sigma']$
   $=$ [hypothesis $\beta \in \mathcal{D}(\sigma)$]
   $\sigma(\beta)[\sigma']$
   $=$ [definition of $\sigma[\sigma']$]
   $\sigma[\sigma'](\beta)$
   $=$ [$\mathcal{D}(\sigma[\sigma']) = \mathcal{D}(\sigma)$]
   $\beta[\sigma[\sigma']]$
   $=$ [1]
   $\beta[\sigma'][\sigma[\sigma']]$

If $\beta \in \mathcal{D}(\sigma')$:

1. $\beta \notin \mathcal{D}(\sigma)$ [hypothesis $\mathcal{D}(\sigma) \cap \mathcal{D}(\sigma') = \emptyset$]
2. $\forall \alpha \in \mathcal{FTV}(\beta) \cap \mathcal{D}(\sigma').\ \mathcal{FTV}(\sigma'(\alpha)) \cap \mathcal{D}(\sigma) = \emptyset$ [hypothesis]
3. $\mathcal{FTV}(\sigma'(\beta)) \cap \mathcal{D}(\sigma) = \emptyset$ [hypothesis $\beta \in \mathcal{D}(\sigma')$, 2]
4. $\beta[\sigma][\sigma']$
   $=$ [1]
   $\beta[\sigma']$
   $=$ [hypothesis $\beta \in \mathcal{D}(\sigma')$]
   $\sigma'(\beta)$
   $=$ [3, $\mathcal{D}(\sigma[\sigma']) = \mathcal{D}(\sigma)$, Theorem 4.27]
   $\sigma'(\beta)[\sigma[\sigma']]$
   $=$ [hypothesis $\beta \in \mathcal{D}(\sigma')$]
   $\beta[\sigma'][\sigma[\sigma']]$

$T_1 \rightarrow T_2$)

1. $\mathcal{FTV}(T_1 \rightarrow T_2) = \mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2)$
2. $\forall \alpha \in \mathcal{FTV}(T_1 \rightarrow T_2) \cap \mathcal{D}(\sigma').\ \mathcal{FTV}(\sigma'(\alpha)) \cap \mathcal{D}(\sigma) = \emptyset$ [hypothesis]
3. $\forall \alpha \in \mathcal{FTV}(T_1) \cap \mathcal{D}(\sigma').\ \mathcal{FTV}(\sigma'(\alpha)) \cap \mathcal{D}(\sigma) = \emptyset$ [1, 2]
4. $\forall \alpha \in \mathcal{FTV}(T_2) \cap \mathcal{D}(\sigma').\ \mathcal{FTV}(\sigma'(\alpha)) \cap \mathcal{D}(\sigma) = \emptyset$ [1, 2]
5. $(T_1 \rightarrow T_2)[\sigma][\sigma']$
   $=$
   $T_1[\sigma][\sigma'] \rightarrow T_2[\sigma][\sigma']$
   $=$ [induction hypothesis, 3, 4]
   $T_1[\sigma'][\sigma[\sigma']] \rightarrow T_2[\sigma'][\sigma[\sigma']]$
   $=$
   $(T_1 \rightarrow T_2)[\sigma'][\sigma[\sigma']]$

all other kinds)
Analogous to $T_1 \rightarrow T_2$.

Proof of Theorem 4.31

By straightforward calculation, and by induction on n for $\forall v_1 : T_1, \ldots, v_n : T_n.\ e$ and for $\exists v_1 : T_1, \ldots, v_n : T_n.\ e$.

Proof of Theorem 4.32

By straightforward calculation, and by induction on n for $\forall v_1 : T_1, \ldots, v_n : T_n.\ e$ and for $\exists v_1 : T_1, \ldots, v_n : T_n.\ e$. Theorem 4.1 can be readily used for the expressions that do not have free variables.

Proof of Theorem 4.33

By straightforward calculation, and by induction on n for $\forall v_1 : T_1, \ldots, v_n : T_n.\ e$ and for $\exists v_1 : T_1, \ldots, v_n : T_n.\ e$ (using Theorem 4.1 for the base cases $\forall \epsilon.\ e$ and $\exists \epsilon.\ e$, which both stand for e, when $u \notin \mathcal{FV}(e)$). Theorem 4.1 can be readily used for the expressions that do not have free variables.

Proof of Theorem 4.34

By straightforward calculation, and by induction on n for $\forall v_1 : T_1, \ldots, v_n : T_n.\ e$ and for $\exists v_1 : T_1, \ldots, v_n : T_n.\ e$. Recall that a projection $e.f$ is implicitly decorated by a record type $\prod_i f_i\ T_i$.

Proof of Theorem 4.35

By straightforward calculation, and by induction on n for $\forall v_1 : T_1, \ldots, v_n : T_n.\ e$ and for $\exists v_1 : T_1, \ldots, v_n : T_n.\ e$. Recall that a projection $e.f$ is implicitly decorated by a record type $\prod_i f_i\ T_i$.

Proof of Theorem 4.36

By induction on the length of cx_2, using the fact that a judgement of the form $\vdash cx, cxel : \textsc{context}$ can be only derived by a rule in §3.1 that has $\vdash cx : \textsc{context}$ as one of its premises.

Proof of Theorem 4.37

By induction on the proof of $(cx \vdash \ldots)$. All the inference rules whose conclusion is a judgement of the form $(cx \vdash \ldots)$ either have $\vdash cx : \textsc{context}$ as one of their premises, which immediately proves $\vdash cx : \textsc{context}$, or have at least one premise of the form $(cx \vdash \ldots)$, which proves $\vdash cx : \textsc{context}$ by induction hypothesis.

Proof of Theorem 4.38

1. $\vdash cx_1, \mathsf{ty}\ \tau : n, cx_2 : \textsc{context}$ [hypothesis]
2. $\vdash cx_1, \mathsf{ty}\ \tau : n : \textsc{context}$ [Theorem 4.36, 1]
3. $\tau \notin \mathcal{TN}(cx_1)$ [premise of $\textsc{cxTdec}$, only rule that can derive 2]

Consider $\tau' \in \mathcal{TN}(cx_2)$:

4. $\exists cx_2', cx_2'', n'.\ \ cx_2 = cx_2', \mathsf{ty}\ \tau' : n', cx_2''$
5. $\vdash cx_1, \mathsf{ty}\ \tau : n, cx_2', \mathsf{ty}\ \tau' : n' : \textsc{context}$ [1, 4, Theorem 4.36]
6. $\tau' \notin \mathcal{TN}(cx_1, \mathsf{ty}\ \tau : n, cx_2')$ [premise of $\textsc{cxTdec}$, only rule that can derive 5]
7. $\mathcal{TN}(cx_1, \mathsf{ty}\ \tau : n, cx_2') = \mathcal{TN}(cx_1) \cup \{\tau\} \cup \mathcal{TN}(cx_2')$
8. $\tau \neq \tau'$ [6, 7]

Since τ' is arbitrary:

9. $\forall \tau' \in \mathcal{TN}(cx_2).\ \ \tau \neq \tau'$
10. $\tau \notin \mathcal{TN}(cx_2)$ [9]
11. $\tau \notin \mathcal{TN}(cx_1) \cup \mathcal{TN}(cx_2)$ [3, 10]

Proof of Theorem 4.39

Analogous to Theorem 4.38.

Proof of Theorem 4.40

Analogous to Theorem 4.38.

Proof of Theorem 4.41

Analogous to Theorem 4.38.

Proof of Theorem 4.42

1. $\vdash cx_1, \mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T, cx_2 : \textsc{context}$ [hypothesis]
2. $\vdash cx_1, \mathsf{op}\ o\!:\!\{\overline{\beta}\}\ T : \textsc{context}$ [Theorem 4.36, 1]
3. $cx_1, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type}$ [premise of cxOdec, only rule that can derive 2]

Proof of Theorem 4.43

Analogous to Theorem 4.42, using cxTdef instead of cxOdec.

Proof of Theorem 4.44

Analogous to Theorem 4.42, using cxAx instead of cxOdec.

Proof of Theorem 4.45

Analogous to Theorem 4.42, using cxLem instead of cxOdec.

Proof of Theorem 4.46

Analogous to Theorem 4.42, using cxVdec instead of cxOdec.

Proof of Theorem 4.47

The only rule that can derive a judgement of the form $cx \vdash \beta : \textsc{type}$ is tyVar, which has $\beta \in \mathcal{TV}(cx)$ as its premise.

Proof of Theorem 4.48

The only rule that can derive a judgement of the form $cx \vdash \tau[\overline{T}] : \textsc{type}$ is tyInst, which has $\mathsf{ty}\ \tau\!:\!n \in cx$, with $n = |\overline{T}|$, as well as $\forall i.\ \ cx \vdash T_i : \textsc{type}$ as two of its premises.

Proof of Theorem 4.49

The only rule that can derive a judgement of the form $cx \vdash T_1 \rightarrow T_2 : \textsc{type}$ is tyArr, which has $cx \vdash T_1 : \textsc{type}$ and $cx \vdash T_2 : \textsc{type}$ as its premises.

Proof of Theorem 4.50

The only rule that can derive a judgement of the form $cx \vdash \prod_i f_i\ T_i : \textsc{type}$ is tyRec, which has $\forall i.\ \ cx \vdash T_i : \textsc{type}$ as one of its premises.

Proof of Theorem 4.51

The only rule that can derive a judgement of the form $cx \vdash T|r : \textsc{type}$ is tyRestr, which has $cx \vdash r : T \rightarrow \mathsf{Bool}$ as one of its premises.

Proof of Theorem 4.52

The only rule that can derive a judgement of the form $cx \vdash T|r : \textsc{type}$ is tyRestr, which has $\mathcal{FV}(r) = \emptyset$ as one of its premises.

Proof of Theorem 4.53

By induction on the derivation of $cx \vdash T_1 \prec_r T_2$:

stRestr)

1. $cx \vdash T|r : \textsc{type}$ [premise of stRestr]
2. $\mathcal{FV}(r) = \emptyset$ [Theorem 4.52, 1]

stRefl)

$\mathcal{FV}(\lambda v\!:\!T.\ \mathsf{true}) = \emptyset$ [Theorem 4.31]

stArr)

1. $\mathcal{FV}(r) = \emptyset$ [induction hypothesis]
2. $\mathcal{FV}(\lambda v\!:\!T \to T_2.\ \forall v'\!:\!T.\ r\ (v\ v'))$
   $=$ [Theorem 4.31]
   $\mathcal{FV}(r) - \{v, v'\}$
   $=$ [1]
   $\emptyset$

stRec)

1. $\forall i.\ \ \mathcal{FV}(r_i) = \emptyset$ [induction hypothesis]
2. $\mathcal{FV}(\lambda v\!:\!\prod_i f_i\ T'_i.\ \bigwedge_i r_i\ v.f_i)$
   $=$ [Theorem 4.31]
   $\bigcup_i \mathcal{FV}(r_i) - \{v\}$
   $=$ [1]
   $\emptyset$

stTE)

$\mathcal{FV}(r) = \emptyset$ [induction hypothesis]

Proof of Theorem 4.54

We prove the first three implications by induction on the derivation of the judgements in the antecedents of the implications:

cxMT)
This rule is impossible because $cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e, cx_2 \neq \epsilon$.

cxTdec, cxOdec, cxTdef, cxLem, cxTVdec, cxVdec)

1. $cx_2 \neq \epsilon$
2. $\exists cx'_2, cxel.\ \ cx_2 = cx'_2, cxel$ [1]
3. $\vdash cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e, cx'_2 : \textsc{context}$ [premise of rule, 2]
4. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx_1)$ [induction hypothesis, 3]

cxAx)
If $cx_2 = \epsilon$:

1. $\vdash cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e : \textsc{context}$ [hypothesis]
2. $cx_1, \mathsf{tvar}\ \overline{\beta} \vdash e : \mathsf{Bool}$ [Theorem 4.44, 1]
3. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx_1, \mathsf{tvar}\ \overline{\beta}) = \mathcal{V}(cx_1)$ [induction hypothesis, 2]

If $cx_2 \neq \epsilon$, the proof is analogous to CXTDEC.

EXOP, EXTHE, EXPROJ)
$\mathcal{FV}(e) = \emptyset$

EXVAR)

1. $\mathsf{var}\ v\!:\!T \in cx$ [premise of EXVAR]
2. $v \in \mathcal{V}(cx)$ [1]
3. $\mathcal{FV}(v) = \{v\} \subseteq \mathcal{V}(cx)$ [2]

EXAPP)

$\mathcal{FV}(e_1\ e_2)$
$=$
$\mathcal{FV}(e_1) \cup \mathcal{FV}(e_2)$
$\subseteq$ [induction hypothesis]
$\mathcal{V}(cx)$

EXEQ, EXIF, EXIF0, EXSUPER, EXSUB)
Analogous to EXAPP.

EXABS)

1. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx, \mathsf{var}\ v\!:\!T)$ [induction hypothesis]
2. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx) \cup \{v\}$ [1]
3. $\mathcal{FV}(e) - \{v\} \subseteq \mathcal{V}(cx) - \{v\}$ [2]
4. $\mathcal{FV}(\lambda v\!:\!T.\ e) \subseteq \mathcal{V}(cx)$ [3]

EXABSALPHA)

1. $\mathcal{FV}(\lambda v\!:\!T.\ e) \subseteq \mathcal{V}(cx)$ [induction hypothesis]
2. $v' \notin \mathcal{FV}(e) \cup \mathcal{CV}(e, v)$ [premise of EXABSALPHA]
3. $\mathcal{FV}(\lambda v\!:\!T.\ e) = \mathcal{FV}(\lambda v'\!:\!T.\ e[v/v'])$ [Theorem 4.17, 2]
4. $\mathcal{FV}(\lambda v'\!:\!T.\ e[v/v']) \subseteq \mathcal{V}(cx)$ [1, 3]

THAX)

1. $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx$ [premise of THAX]
2. $\exists cx_1, cx_2.\ \ cx = cx_1, \mathsf{ax}\ \{\overline{\beta}\}\ e, cx_2$ [1]
3. $\vdash cx : \textsc{context}$ [premise of THAX]
4. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx_1)$ [induction hypothesis, 2, 3]
5. $\mathcal{V}(cx_1) \subseteq \mathcal{V}(cx)$ [2]
6. $\mathcal{FV}(e[\overline{\beta}/\overline{T}]) = \mathcal{FV}(e)$ [Theorem 4.18]
7. $\mathcal{FV}(e[\overline{\beta}/\overline{T}]) \subseteq \mathcal{V}(cx)$ [6, 4, 5]

THABS)

1. $\mathcal{FV}((\lambda v\!:\!T.\ e)\ e' \equiv e[v/e']) = \mathcal{FV}((\lambda v\!:\!T.\ e)\ e') \cup \mathcal{FV}(e[v/e'])$
2. $cx \vdash (\lambda v\!:\!T.\ e)\ e' : \ldots$ [premise of THABS]

3. $\mathcal{FV}((\lambda v{:}T.\ e)\ e') \subseteq \mathcal{V}(cx)$ [induction hypothesis, 2]
4. $\mathcal{FV}(\lambda v{:}T.\ e) \subseteq \mathcal{V}(cx)$ [3]
5. $\mathcal{FV}(e) - \{v\} \subseteq \mathcal{V}(cx)$ [4]

If $v \notin \mathcal{FV}(e)$:

6. $\mathcal{FV}(e[v/e']) = \mathcal{FV}(e)$ [Theorem 4.1]
7. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx)$ [5, hypothesis $v \notin \mathcal{FV}(e)$]
8. $\mathcal{FV}((\lambda v{:}T.\ e)\ e' \equiv e[v/e']) \subseteq \mathcal{V}(cx)$ [1, 3, 6, 7]

If $v \in \mathcal{FV}(e)$:

6. $OKsbs(e, v, e')$ [premise of THABS]
7. $\mathcal{FV}(e[v/e']) = (\mathcal{FV}(e) - \{v\}) \cup \mathcal{FV}(e')$ [Theorem 4.8, 6, hypothesis $v \in \mathcal{FV}(e)$]
8. $\mathcal{FV}(e') \subseteq \mathcal{V}(cx)$ [3]
9. $\mathcal{FV}((\lambda v{:}T.\ e)\ e' \equiv e[v/e']) \subseteq \mathcal{V}(cx)$ [1, 3, 7, 8, 5]

THSUB)

1. $\mathcal{FV}(r\ e) = \mathcal{FV}(r) \cup \mathcal{FV}(e)$
2. $cx \vdash e : T$ [premise of THSUB]
3. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx)$ [induction hypothesis, 2]
4. $cx \vdash T \prec_r T'$ [premise of THSUB]
5. $\mathcal{FV}(r) = \emptyset$ [Theorem 4.53, 4]
6. $\mathcal{FV}(r\ e) \subseteq \mathcal{V}(cx)$ [1, 3, 5]

all other rules)

The conclusion $cx \vdash e$ of some rules is such that $\mathcal{FV}(e) = \emptyset$ (proved by the definition of $\mathcal{FV}$ and in some cases by Theorem 4.31), which immediately proves $\mathcal{FV}(e) \subseteq \mathcal{V}(cx)$. The conclusion $cx \vdash e$ of the other rules is such that every variable in $\mathcal{FV}(e)$ is also in some $\mathcal{FV}(e')$ that occurs in a premise $cx' \vdash e' : \ldots$ or $cx' \vdash e'$ of the rule such that $\mathcal{V}(cx') = \mathcal{V}(cx)$, which proves $\mathcal{FV}(e) \subseteq \mathcal{V}(cx)$ by induction hypothesis (the only rules for which $cx' \neq cx$ are THIFSUBST and THIF, where $cx' = cx, \mathsf{ax}\ \ldots$ and so clearly $\mathcal{V}(cx') = \mathcal{V}(cx)$).

The fourth implication is proved as follows:

1. $\vdash cx_1, \mathsf{lem}\ \{\overline{\beta}\}\ e, cx_2 : \text{CONTEXT}$ [hypothesis]
2. $\vdash cx_1, \mathsf{lem}\ \{\overline{\beta}\}\ e : \text{CONTEXT}$ [Theorem 4.36, 1]
3. $cx_1, \mathsf{tvar}\ \overline{\beta} \vdash e$ [Theorem 4.45, 2]
4. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx_1, \mathsf{tvar}\ \overline{\beta}) = \mathcal{V}(cx_1)$ [second implication of this theorem, 3]

Proof of Theorem 4.55

The theorem statement is equivalent to

$$\begin{array}{lll}
cx \vdash T : \text{TYPE} & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T) \\
cx \vdash T_1 \approx T_2 & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T_1) \cup \mathcal{FTV}(T_2) \\
cx \vdash T_1 \prec_r T_2 & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T_1) \cup \mathcal{FTV}(r) \cup \mathcal{FTV}(T_2) \\
cx \vdash e : T & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(e) \cup \mathcal{FTV}(T) \\
cx \vdash e & \Rightarrow & \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(e) \\
\vdash cx_1, cxel, cx_2 : \text{CONTEXT} & \Rightarrow & \mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1, cxel)
\end{array}$$

because $A \subseteq B \;\Leftrightarrow\; A \cup B \subseteq B$ for all sets A and B.

Before proving the above implications, note that from the last one we can prove

$$\vdash cx : \textsc{context} \quad \Rightarrow \quad \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \qquad (*)$$

If $cx = \epsilon$:

$$\mathcal{TV}(cx) = \emptyset \supseteq \emptyset = \mathcal{FTV}(cx)$$

If $cx \neq \epsilon$:

1. $\exists cx_0, cxel.\;\; cx = cx_0, cxel$
2. $\mathcal{TV}(cx)$
   $\supseteq$ [1]
   $\mathcal{TV}(cx_0)$
   $\supseteq$ [hypothesis]
   $\mathcal{FTV}(cx_0, cxel)$
   $=$ [1]
   $\mathcal{FTV}(cx)$

This concludes the proof of $(*)$.

We prove the six implications above by induction on the derivation of the judgements in the antecedents of the implications[5]:

cxMT)

This rule is impossible because $cx_1, cxel, cx_2 \neq \epsilon$.

cxTdec)

If $cx_2 \neq \epsilon$:

1. $\exists cx_2', \tau, n.\;\; cx_2 = cx_2', \mathsf{ty}\ \tau : n$
2. $\vdash cx_1, cxel, cx_2' : \textsc{context}$ [premise of cxTdec, 1]
3. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1, cxel)$ [induction hypothesis, 2]

If $cx_2 = \epsilon$:

1. $\exists \tau, n.\;\; cxel = \mathsf{ty}\ \tau : n$
2. $\mathcal{FTV}(cxel) = \emptyset$ [1]
3. $\vdash cx_1 : \textsc{context}$ [premise of cxTdec]
4. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1)$ [induction hypothesis, 3, $(*)$]
5. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1, cxel)$ [4, 2]

cxTVdec)

Analogous to cxTdec.

cxOdec)

If $cx_2 \neq \epsilon$, the proof is analogous to cxTdec.

If $cx_2 = \epsilon$:

1. $\exists o, \overline{\beta}, T.\;\; cxel = \mathsf{op}\ o : \{\overline{\beta}\}\ T$
2. $\mathcal{FTV}(cxel) = \mathcal{FTV}(T) - \overline{\beta}$ [1]
3. $cx_1, \mathsf{tvar}\ \overline{\beta} \vdash T : \textsc{type}$ [premise of cxOdec]

[5]The reason for proving this restatement of the theorem as opposed to the original statement is that the restatement can be proved by simple rule induction, while the original statement requires induction on the size of the derivation, which is currently not explicitly formalized in this document and which involves a more laborious mechanical proof in theorem provers like HOL.

4. $\mathcal{TV}(cx_1) \cup \overline{\beta} \supseteq \mathcal{FTV}(cx_1) \cup \mathcal{FTV}(T)$ [induction hypothesis, 3]
5. $(\mathcal{TV}(cx_1) \cup \overline{\beta}) - \overline{\beta} \supseteq (\mathcal{FTV}(cx_1) \cup \mathcal{FTV}(T)) - \overline{\beta}$ [4]
6. $\vdash cx_1, \mathsf{tvar}\ \overline{\beta} : \textsc{context}$ [Theorem 4.37, 3]
7. $\overline{\beta} \cap \mathcal{TV}(cx_1) = \emptyset$ [Theorem 4.40, 6]
8. $(\mathcal{TV}(cx_1) \cup \overline{\beta}) - \overline{\beta} = \mathcal{TV}(cx_1)$ [7]
9. $\vdash cx_1 : \textsc{context}$ [premise of cxOdec]
10. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1)$ [induction hypothesis, 9, (∗)]
11. $\overline{\beta} \cap \mathcal{FTV}(cx_1) = \emptyset$ [7, 10]
12. $\mathcal{FTV}(cx_1) - \overline{\beta} = \mathcal{FTV}(cx_1)$ [11]
13. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1, cxel)$ [5, 8, 12, 2]

cxTdef, cxAx, cxLem)
Analogous to cxOdec.

cxVdec)
If $cx_2 \neq \epsilon$, the proof is analogous to cxTdec.

If $cx_2 = \epsilon$:

1. $\exists v, T.\ \ cxel = \mathsf{var}\ v : T$
2. $cx_1 \vdash T : \textsc{type}$ [premise of cxVdec]
3. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1) \cup \mathcal{FTV}(T)$ [induction hypothesis, 2]
4. $\mathcal{TV}(cx_1) \supseteq \mathcal{FTV}(cx_1, cxel)$ [3, 1]

tyBool)

1. $\vdash cx : \textsc{context}$ [premise of tyBool]
2. $\mathcal{TV}(cx)$
   $\supseteq$ [induction hypothesis, 1, (∗)]
   $\mathcal{FTV}(cx)$
   $=$ [$\mathcal{FTV}(\mathsf{Bool}) = \emptyset$]
   $\mathcal{FTV}(cx) \cup \mathcal{FTV}(\mathsf{Bool})$

tyVar)

1. $\vdash cx : \textsc{context}$ [premise of tyVar]
2. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx)$ [induction hypothesis, 1, (∗)]
3. $\beta \in \mathcal{TV}(cx)$ [premise of tyVar]
4. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(\beta)$ [2, 3]

tyInst)

1. $\mathcal{FTV}(\tau[\overline{T}]) = \bigcup_i \mathcal{FTV}(T_i)$
2. $\forall i.\ \ cx \vdash T_i : \textsc{type}$ [premise of tyInst]
3. $\forall i.\ \ \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T_i)$ [induction hypothesis, 2]
4. $\forall i.\ \ \mathcal{TV}(cx) \supseteq \mathcal{FTV}(T_i)$ [3]
5. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(\tau[\overline{T}])$ [1, 4]
6. $\vdash cx : \textsc{context}$ [premise of tyInst]
7. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx)$ [induction hypothesis, 6, (∗)]
8. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(\tau[\overline{T}])$ [5, 7]

teDef)

1. $\mathcal{FTV}(\tau[\overline{T}]) = \bigcup_i \mathcal{FTV}(T_i)$
2. $\mathcal{FTV}(T[\overline{\beta}/\overline{T}]) = (\mathcal{FTV}(T) - \overline{\beta}) \cup \bigcup_{\beta_i \in \mathcal{FTV}(T)} \mathcal{FTV}(T_i)$ [Theorem 4.28]
3. $\forall i.\;\; cx \vdash T_i : \text{TYPE}$ [premise of teDef]
4. $\forall i.\;\; \mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T_i)$ [induction hypothesis, 3]
5. $\forall i.\;\; \mathcal{TV}(cx) \supseteq \mathcal{FTV}(T_i)$ [4]
6. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(\tau[\overline{T}])$ [5, 1]
7. $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx$ [premise of teDef]
8. $\exists cx_1, cx_2.\;\; cx = cx_1, \mathsf{def}\ \tau[\overline{\beta}] = T, cx_2$ [7]
9. $\vdash cx : \text{CONTEXT}$ [premise of teDef]
10. $\mathcal{TV}(cx)$
    $\supseteq$ [8]
    $\mathcal{TV}(cx_1)$
    $\supseteq$ [induction hypothesis, 9, 8]
    $\mathcal{FTV}(cx_1, \mathsf{def}\ \tau[\overline{\beta}] = T)$
    $\supseteq$
    $\mathcal{FTV}(T) - \overline{\beta}$
11. $\bigcup_{\beta_i \in \mathcal{FTV}(T)} \mathcal{FTV}(T_i) \subseteq \bigcup_i \mathcal{FTV}(T_i)$
12. $\mathcal{TV}(cx) \supseteq \bigcup_{\beta_i \in \mathcal{FTV}(T)} \mathcal{FTV}(T_i)$ [5, 11]
13. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx)$ [induction hypothesis, 9, $(*)$]
14. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(\tau[\overline{T}]) \cup \mathcal{FTV}(T[\overline{\beta}/\overline{T}])$ [13, 6, 2, 10, 12]

exVar)

1. $\mathsf{var}\ v\!:\!T \in cx$ [premise of exVar]
2. $\exists cx_1, cx_2.\;\; cx = cx_1, \mathsf{var}\ v\!:\!T, cx_2$ [1]
3. $\vdash cx : \text{CONTEXT}$ [premise of exVar]
4. $\mathcal{TV}(cx)$
   $\supseteq$ [2]
   $\mathcal{TV}(cx_1)$
   $\supseteq$ [induction hypothesis, 3, 2]
   $\mathcal{FTV}(cx_1) \cup \mathcal{FTV}(T)$
   $\supseteq$
   $\mathcal{FTV}(T)$
5. $\vdash cx : \text{CONTEXT}$ [premise of exVar]
6. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx)$ [induction hypothesis, 5, $(*)$]
7. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T)$ [6, 4]

exOp)

Analogous to teDef.

exAbs)

1. $cx, \mathsf{var}\ v\!:\!T \vdash e : T'$ [premise of exAbs]
2. $\mathcal{TV}(cx, \mathsf{var}\ v\!:\!T) \supseteq \mathcal{FTV}(cx, \mathsf{var}\ v\!:\!T) \cup \mathcal{FTV}(e) \cup \mathcal{FTV}(T')$ [induction hypothesis, 1]
3. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(T) \cup \mathcal{FTV}(e) \cup \mathcal{FTV}(T')$ [2]
4. $\mathcal{FTV}(\lambda v\!:\!T.\ e) = \mathcal{FTV}(T) \cup \mathcal{FTV}(e)$
5. $\mathcal{FTV}(T \to T') = \mathcal{FTV}(T) \cup \mathcal{FTV}(T')$

6. $\mathcal{TV}(cx) \supseteq \mathcal{FTV}(cx) \cup \mathcal{FTV}(\lambda v{:}T.\ e) \cup \mathcal{FTV}(T \to T')$ [3, 4, 5]

THAX)

Analogous to TEDEF.

all other rules)

Analogous to TYINST: each free type variable in the judgement $cx \vdash \ldots$ is also a free type variable in some premise $cx' \vdash \ldots$ of the rule such that $\mathcal{TV}(cx') = \mathcal{TV}(cx)$ (the only rules in which $cx' \neq cx$ are EXIF, THIFSUBST, THIF). We use Theorem 4.35 for STREFL, STARR, STREC, EXTHE, THBOOL, THEXT, THREC, and THPROJSUB. We use Theorem 4.21 for EXABSALPHA. We use Theorem 4.20 for THABS.

Proof of Theorem 4.56

By induction on the derivation of the judgements in the antecedents of the implications:

CXMT)

This rule is impossible because $cx_1, \mathsf{var}\ u{:}S, cx_2 \neq \epsilon$.

CXTDEC)

The judgement in the antecedent of the implication must end with a type declaration, so the cx_2 in the judgement must end with that type declaration. For convenience we "rename" cx_2 to $cx_2, \mathsf{ty}\ \tau{:}n$. The instance of CXTDEC used to derive the judgement is thus

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context}\\ \tau \notin \mathcal{TN}(cx_1, \mathsf{var}\ u{:}S, cx_2)\end{array}}{\vdash cx_1, \mathsf{var}\ u{:}S, cx_2, \mathsf{ty}\ \tau{:}n : \textsc{context}} \quad (0)$$

and the judgement $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'], \mathsf{ty}\ \tau{:}n : \textsc{context}$ is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] : \textsc{context}$ [induction hypothesis, 1]
3. $\tau \notin \mathcal{TN}(cx_1, \mathsf{var}\ u{:}S, cx_2)$ [0]
4. $\mathcal{TN}(cx_1, \mathsf{var}\ u{:}S, cx_2)$
   $=$
   $\mathcal{TN}(cx_1) \cup \mathcal{TN}(cx_2)$
   $=$ [Theorem 4.24]
   $\mathcal{TN}(cx_1) \cup \mathcal{TN}(cx_2[u/u'])$
   $=$
   $\mathcal{TN}(cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'])$
5. $\tau \notin \mathcal{TN}(cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'])$ [3, 4]
6. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'], \mathsf{ty}\ \tau{:}n : \textsc{context}$ [CXTDEC, 2, 5]

CXODEC, CXTDEF, CXAX, CXLEM, CXTVDEC)

Analogous to CXTDEC: starting with the instance of the rule used to derive the judgement in the antecedent of the implication, we replace u with u' in the premises of the rule instance and then we apply the rule to the transformed premises to obtain the conclusion with u replaced with u' (using Theorem 4.24 as needed).

CXVDEC)

If $cx_2 = \epsilon$, the instance of CXVDEC used to derive the judgement in the antecedent of the implication is

$$\frac{\begin{array}{c}\vdash cx_1 : \textsc{context}\\ u \notin \mathcal{V}(cx_1)\\ cx_1 \vdash T : \textsc{type}\end{array}}{\vdash cx_1, \mathsf{var}\ u{:}T : \textsc{context}} \quad (0)$$

and the judgement $\vdash cx_1, \mathsf{var}\ u'{:}T : \textsc{context}$ is derived as follows:

1. $\vdash cx_1 : \textsc{context}$ [0]
2. u' fresh [hypothesis]
3. $u' \notin \mathcal{V}(cx_1)$ [2]
4. $cx_1 \vdash T : \textsc{type}$ [0]
5. $\vdash cx_1, \mathsf{var}\ u' {:} T : \textsc{context}$ [cxVdec, 1, 3, 4]

If $cx_2 \neq \epsilon$, the judgement in the antecedent of the implication must end with a variable declaration, so the cx_2 in the judgement must end with that variable declaration. For convenience we "rename" cx_2 to $cx_2, \mathsf{var}\ v {:} T$. The instance of cxVdec used to derive the judgement is thus

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context} \\ v \notin \mathcal{V}(cx_1, \mathsf{var}\ u{:}S, cx_2) \\ cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T : \textsc{type}\end{array}}{\vdash cx_1, \mathsf{var}\ u{:}S, cx_2, \mathsf{var}\ v{:}T : \textsc{context}} \quad (0)$$

and the judgement $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'], \mathsf{var}\ v{:}T : \textsc{context}$ is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] : \textsc{context}$ [induction hypothesis, 1]
3. $v \notin \mathcal{V}(cx_1, \mathsf{var}\ u{:}S, cx_2)$ [0]
4. $\mathcal{V}(cx_1, \mathsf{var}\ u{:}S, cx_2) = \mathcal{V}(cx_1) \cup \{u\} \cup \mathcal{V}(cx_2)$
5. $v \notin \mathcal{V}(cx_1)$ [3, 4]
6. $\mathcal{V}(cx_2) = \mathcal{V}(cx_2[u/u'])$ [Theorem 4.24]
7. $v \notin \mathcal{V}(cx_2[u/u'])$ [3, 4, 6]
8. u' fresh [hypothesis]
9. $v \neq u'$ [8]
10. $\mathcal{V}(cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u']) = \mathcal{V}(cx_1) \cup \{u'\} \cup \mathcal{V}(cx_2[u/u'])$
11. $v \notin \mathcal{V}(cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'])$ [10, 5, 9, 7]
12. $cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T : \textsc{type}$ [0]
13. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T : \textsc{type}$ [12, induction hypothesis]
14. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'], \mathsf{var}\ v{:}T : \textsc{context}$ [cxVdec, 2, 11, 13]

tyBool)

Let the instance of tyBool used to derive the judgement in the antecedent of the implication be

$$\frac{\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context}}{cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash \mathsf{Bool} : \textsc{type}} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] : \textsc{context}$ [induction hypothesis, 1]
3. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 2]

exVar)

If the judgement in the antecedent of the implication is $cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash u : \ldots$, the instance of exVar used to derive the judgement is

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \textsc{context} \\ \mathsf{var}\ u{:}S \in cx_1, \mathsf{var}\ u{:}S, cx_2\end{array}}{cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash u : S} \quad (0)$$

(the type $\ldots$ is S by Theorem 4.41) and the judgement in the consequent of the implication is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u\!:\!S, cx_2 : \text{CONTEXT}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] : \text{CONTEXT}$ [induction hypothesis, 1]
3. $\mathsf{var}\ u'\!:\!S \in cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u']$
4. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash u' : S$ [exVar, 2, 3]
5. $u[u/u'] = u'$
6. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash u[u/u'] : S$ [4, 5]

If the judgement in the antecedent of the implication is $cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash v : T$ with $v \neq u$, the instance of exVar used to derive the judgement is

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{var}\ u\!:\!S, cx_2 : \text{CONTEXT}\\ \mathsf{var}\ v\!:\!T \in cx_1, \mathsf{var}\ u\!:\!S, cx_2\end{array}}{cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash v : T} \quad (0)$$

and the judgement in the consequent of the implication is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u\!:\!S, cx_2 : \text{CONTEXT}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] : \text{CONTEXT}$ [induction hypothesis, 1]
3. $\mathsf{var}\ v\!:\!T \in cx_1, \mathsf{var}\ u\!:\!S, cx_2$ [0]
4. $\mathsf{var}\ v\!:\!T \in cx_1, cx_2$ [3, hypothesis $v \neq u$]
5. $\mathsf{var}\ v\!:\!T \in cx_1, cx_2[u/u']$ [4, Theorem 4.24]
6. $\mathsf{var}\ v\!:\!T \in cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u']$ [5]
7. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash v : T$ [exVar, 2, 6]
8. $v[u/u'] = v$ [hypothesis $v \neq u$]
9. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash v[u/u'] : T$ [7, 8]

exAbs)

Let the instance of exAbs used to derive the judgement in the antecedent of the implication be

$$\frac{\begin{array}{c}cx_1, \mathsf{var}\ u\!:\!S, cx_2, \mathsf{var}\ v\!:\!T \vdash e : T'\\ cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash T' : \text{TYPE}\end{array}}{cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash \lambda v\!:\!T.\ e : T \to T'} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $cx_1, \mathsf{var}\ u\!:\!S, cx_2, \mathsf{var}\ v\!:\!T \vdash e : T'$ [0]
2. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'], \mathsf{var}\ v\!:\!T \vdash e[u/u'] :$ [induction hypothesis, 1]
3. $cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash T' : \text{TYPE}$ [0]
4. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash T' : \text{TYPE}$ [induction hypothesis, 3]
5. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v\!:\!T.\ e[u/u'] : T \to T'$ [exAbs, 2, 4]
6. $\vdash cx_1, \mathsf{var}\ u\!:\!S, cx_2, \mathsf{var}\ v\!:\!T : \text{CONTEXT}$ [Theorem 4.37, 1]
7. $v \neq u$ [Theorem 4.41, 6]
8. $(\lambda v\!:\!T.\ e)[u/u'] = \lambda v\!:\!T.\ e[u/u']$ [7]
9. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)[u/u'] : T \to T'$ [5, 8]

exAbsAlpha)

Let the instance of exAbsAlpha used to derive the judgement in the antecedent of the implication be

$$\frac{\begin{array}{c}cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash \lambda v\!:\!T.\ e : T'\\ v' \notin \mathcal{FV}(e) \cup \mathcal{CV}(e, v)\end{array}}{cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash \lambda v'\!:\!T.\ e[v/v'] : T'} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash \lambda v\!:\!T.\ e : T'$ [0]
2. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)[u/u'] : T'$ [induction hypothesis, 1]
3. $v' \notin \mathcal{FV}(e) \cup \mathcal{CV}(e, v)$ [0]
4. $\mathcal{FV}(\lambda v'\!:\!T.\ e[v/v']) = \mathcal{FV}(\lambda v\!:\!T.\ e)$ [Theorem 4.17, 3]

If $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$:

5. $(\lambda v\!:\!T.\ e)[u/u'] = \lambda v\!:\!T.\ e$ [Theorem 4.1]
6. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v\!:\!T.\ e : T'$ [2, 5]
7. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v'\!:\!T.\ e[v/v'] : T'$ [exAbsAlpha, 6, 3]
8. $u \notin \mathcal{FV}(\lambda v'\!:\!T.\ e[v/v'])$ [4, hypothesis $u \notin \mathcal{FV}(\lambda v\!:\!T.\ e)$]
9. $(\lambda v'\!:\!T.\ e[v/v'])[u/u'] = \lambda v'\!:\!T.\ e[v/v']$ [Theorem 4.1, 8]
10. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v'\!:\!T.\ e[v/v'])[u/u'] : T'$ [7, 9]

If $u \in \mathcal{FV}(\lambda v\!:\!T.\ e)$:

5. $u \in \mathcal{FV}(e) - \{v\}$
6. $u \in \mathcal{FV}(e)$ [5]
7. $u \neq v$ [5]
8. $(\lambda v\!:\!T.\ e)[u/u'] = \lambda v\!:\!T.\ e[u/u']$ [7]
9. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v\!:\!T.\ e[u/u'] : T'$ [2, 8]
10. u' fresh [hypothesis]
11. $u' \notin \mathcal{CV}(e, u)$ [10]
12. $OKsbs(e, u, u')$ [11]
13. $\mathcal{FV}(e[u/u']) = (\mathcal{FV}(e) - \{u\}) \cup \{u'\}$ [Theorem 4.8, 6, 12]
14. $v' \notin \mathcal{FV}(e)$ [3]
15. $v' \notin \mathcal{FV}(e) - \{u\}$ [14]
16. $v' \neq u'$ [10]
17. $v' \notin \mathcal{FV}(e[u/u'])$ [13, 15, 16]
18. $v \neq u'$ [10]
19. $\mathcal{CV}(e, v) = \mathcal{CV}(e[u/u'], v)$ [Theorem 4.16, 12, 7, 18]
20. $v' \notin \mathcal{CV}(e, v)$ [3]
21. $v' \notin \mathcal{CV}(e[u/u'], v)$ [19, 20]
22. $v' \notin \mathcal{FV}(e[u/u']) \cup \mathcal{CV}(e[u/u'], v)$ [17, 21]
23. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v'\!:\!T.\ e[u/u'][v/v'] : T'$ [exAbsAlpha, 9, 22]
24. $u \neq v'$ [6, 14]
25. $e[u/u'][v/v'] = e[v/v'][u/u']$ [Theorem 4.10, 7, 24, 18]
26. $(\lambda v'\!:\!T.\ e[v/v'])[u/u'] = \lambda v'\!:\!T.\ e[v/v'][u/u']$ [24]
27. $cx_1, \mathsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v'\!:\!T.\ e[v/v'])[u/u'] : T'$ [23, 25, 26]

thAx)

Let the instance of thAx used to derive the judgement in the antecedent of the implication be

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{var}\ u\!:\!S, cx_2 : \text{CONTEXT} \\ \mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1, \mathsf{var}\ u\!:\!S, cx_2 \\ \forall i.\ \ cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash T_i : \text{TYPE}\end{array}}{cx_1, \mathsf{var}\ u\!:\!S, cx_2 \vdash e[\overline{\beta}/\overline{T}]} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $\vdash cx_1, \mathsf{var}\ u{:}S, cx_2 : \text{CONTEXT}$ [0]
2. $\vdash cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] : \text{CONTEXT}$ [induction hypothesis, 1]
3. $\forall i.\ \ cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T_i : \text{TYPE}$ [0]
4. $\forall i.\ \ cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T_i : \text{TYPE}$ [induction hypothesis, 3]
5. $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1, \mathsf{var}\ u{:}S, cx_2$ [0]
6. $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1, cx_2$ [5]

If $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1$:

7. $\exists cx_1', cx_1''.\ \ cx_1 = cx_1', \mathsf{ax}\ \{\overline{\beta}\}\ e, cx_1''$
8. $cx_1', \mathsf{tvar}\ \overline{\beta} \vdash e : \mathsf{Bool}$ [Theorem 4.44, 1, 7]
9. $\mathcal{FV}(e) \subseteq \mathcal{V}(cx_1')$ [Theorem 4.54, 8]
10. $u \notin \mathcal{V}(cx_1')$ [Theorem 4.41, 1, 7]
11. $u \notin \mathcal{FV}(e)$ [10, 9]
12. $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u']$ [hypothesis $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_1$]
13. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[\overline{\beta}/\overline{T}]$ [thAx, 2, 12, 4]
14. $\mathcal{FV}(e[\overline{\beta}/\overline{T}]) = \mathcal{FV}(e)$ [Theorem 4.18]
15. $u \notin \mathcal{FV}(e[\overline{\beta}/\overline{T}])$ [11, 14]
16. $e[\overline{\beta}/\overline{T}][u/u'] = e[\overline{\beta}/\overline{T}]$ [Theorem 4.1, 15]
17. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[\overline{\beta}/\overline{T}][u/u']$ [13, 16]

If $\mathsf{ax}\ \{\overline{\beta}\}\ e \notin cx_1$:

7. $\mathsf{ax}\ \{\overline{\beta}\}\ e \in cx_2$ [6]
8. $\mathsf{ax}\ \{\overline{\beta}\}\ e[u/u'] \in cx_2[u/u']$ [Theorem 4.24, 7]
9. $\mathsf{ax}\ \{\overline{\beta}\}\ e[u/u'] \in cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u']$ [8]
10. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[u/u'][\overline{\beta}/\overline{T}]$ [thAx, 2, 9, 4]
11. $e[u/u'][\overline{\beta}/\overline{T}] = e[\overline{\beta}/\overline{T}][u/u']$ [Theorem 4.23]
12. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash e[\overline{\beta}/\overline{T}][u/u']$ [10, 11]

thAbsSubst)

Let the instance of thAbsSubst used to derive the judgement in the antecedent of the implication be

$$\frac{\begin{array}{c} cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash \lambda v{:}T.\ e : \ldots \\ cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash T \approx T' \end{array}}{cx_1, \mathsf{var}\ u{:}S, cx_2 \vdash \lambda v{:}T.\ e \equiv \lambda v{:}T'.\ e} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash (\lambda v{:}T.\ e)[u/u'] : \ldots$ [induction hypothesis, 0]
2. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash T \approx T'$ [induction hypothesis, 0]

If $u = v$:

3. $(\lambda v{:}T.\ e)[u/u'] = \lambda v{:}T.\ e\ \wedge\ (\lambda v{:}T'.\ e)[u/u'] = \lambda v{:}T'.\ e$
4. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash \lambda v{:}T.\ e : \ldots$ [3, 1]
5. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash \lambda v{:}T.\ e \equiv \lambda v{:}T'.\ e$ [thAbsSubst, 4, 2]
6. $cx_1, \mathsf{var}\ u'{:}S, cx_2[u/u'] \vdash (\lambda v{:}T.\ e \equiv \lambda v{:}T'.\ e)[u/u']$ [5, 3]

If $u \neq v$:

3. $(\lambda v\!:\!T.\ e)[u/u'] = \lambda v\!:\!T.\ e[u/u'] \ \wedge\ (\lambda v\!:\!T'.\ e)[u/u'] = \lambda v\!:\!T'.\ e[u/u']$
4. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v\!:\!T.\ e[u/u'] : \ldots$ [3, 1]
5. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash \lambda v\!:\!T.\ e[u/u'] \equiv \lambda v\!:\!T'.\ e[u/u']$ [thAbsSubst, 4, 2]
6. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e \equiv \lambda v\!:\!T'.\ e)[u/u']$ [5, 3]

thAbs)

Let the instance of thAbs used to derive the judgement in the antecedent of the implication be

$$\frac{\begin{array}{c} cx_1, \textsf{var}\ u\!:\!S, cx_2 \vdash (\lambda v\!:\!T.\ e)\ e' : \ldots \\ OKsbs(e, v, e') \end{array}}{cx_1, \textsf{var}\ u\!:\!S, cx_2 \vdash (\lambda v\!:\!T.\ e)\ e' \equiv e[v/e']} \quad (0)$$

The judgement in the consequent of the implication is derived as follows:

1. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash ((\lambda v\!:\!T.\ e)\ e')[u/u'] : \ldots$ [induction hypothesis, 0]
2. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)[u/u']\ e'[u/u'] : \ldots$ [1, definition of $\_[\_/\_]$]
3. $OKsbs(e, v, e')$ [0]
4. u' fresh [hypothesis]
5. $OKsbs(e', u, u')$ [4]

If $u = v$:

6. $(\lambda v\!:\!T.\ e)[u/u'] = \lambda v\!:\!T.\ e$ [definition of $\_[\_/\_]$]
7. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)\ e'[u/u'] : \ldots$ [2, 6]

If $u \notin \mathcal{FV}(e')$:

8. $e'[u/u'] = e'$ [Theorem 4.1]
9. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)\ e' : \ldots$ [7, 8]
10. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)\ e' \equiv e[v/e']$ [thAbs, 9, 3]
11. $v \notin \mathcal{FV}(e')$ [hypothesis $u \notin \mathcal{FV}(e')$, hypothesis $u = v$]
12. $v \notin \mathcal{FV}(e[v/e'])$ [Theorem 4.9, 3, 11]
13. $u \notin \mathcal{FV}(e[v/e'])$ [12, hypothesis $u = v$]
14. $e[v/e'][u/u'] = e[v/e']$ [Theorem 4.1, 13]
15. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash ((\lambda v\!:\!T.\ e)\ e' \equiv e[v/e'])[u/u']$ [10, 6, 8, 14]

If $u \in \mathcal{FV}(e')$:

8. $\mathcal{FV}(e'[u/u']) = (\mathcal{FV}(e') - \{u\}) \cup \{u'\}$ [Theorem 4.8, 5]
9. $\mathcal{FV}(e') \cap \mathcal{CV}(e, v) = \emptyset$ [3]
10. $(\mathcal{FV}(e') - \{u\}) \cap \mathcal{CV}(e, v) = \emptyset$ [9]
11. $u' \notin \mathcal{CV}(e, v)$ [4]
12. $OKsbs(e, v, e'[u/u'])$ [8, 10, 11]
13. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)\ e'[u/u'] \equiv e[v/e'[u/u']]$ [thAbs, 7, 12]
14. $e[v/e'[u/u']] = e[u/e'[u/u']]$ [hypothesis $u = v$]
15. $e[u/e'[u/u']] = e[u/e'][u/u']$ [Theorem 4.11]
16. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash (\lambda v\!:\!T.\ e)\ e'[u/u'] \equiv e[v/e'][u/u']$ [13, 14, 15, hypothesis $u = v$]
17. $cx_1, \textsf{var}\ u'\!:\!S, cx_2[u/u'] \vdash ((\lambda v\!:\!T.\ e)\ e' \equiv e[v/e'])[u/u']$ [16, 6]

If $u \neq v$:

6. $cx_1, \mathsf{var}\ u' : S, cx_2[u/u'] \vdash (\lambda v : T.\ e[u/u'])\ e'[u/u'] : \ldots$ [2]
7. $\mathit{OKsbs}(e, u, u')$ [4]
8. $v \neq u'$ [4]
9. $\mathcal{CV}(e, v) = \mathcal{CV}(e[u/u'], v)$ [Theorem 4.16, 7, hypothesis $v \neq u$, 8]
10. $\mathcal{FV}(e') \cap \mathcal{CV}(e, v) = \emptyset$ [3]
11. $\mathcal{FV}(e') \cap \mathcal{CV}(e[u/u'], v) = \emptyset$ [9, 10]

If $u \notin \mathcal{FV}(e')$:

12. $e'[u/u'] = e'$ [Theorem 4.1]
13. $\mathcal{FV}(e'[u/u']) = \mathcal{FV}(e')$ [12]
14. $\mathit{OKsbs}(e[u/u'], v, e'[u/u'])$ [11, 13]
15. $cx_1, \mathsf{var}\ u' : S, cx_2[u/u'] \vdash (\lambda v : T.\ e[u/u'])\ e'[u/u'] \equiv e[u/u'][v/e'[u/u']]$ [THABS, 6, 14]
16. $u' \neq u$ [4]
17. $u \notin \mathcal{FV}(e'[u/u'])$ [Theorem 4.9, 5, 16]
18. $e[u/u'][v/e'[u/u']] = e[v/e'[u/u']][u/u']$ [Theorem 4.10, hypothesis $u \neq v$, 8, 17]
19. $e[v/e'[u/u']][u/u'] = e[v/e'][u/u']$ [Theorem 4.13, 3]
20. $(\lambda v : T.\ e)[u/u'] = \lambda v : T.\ e[u/u']$ [hypothesis $v \neq u$]
21. $cx_1, \mathsf{var}\ u' : S, cx_2[u/u'] \vdash ((\lambda v : T.\ e)\ e' \equiv e[v/e'])[u/u']$ [15, 20, 18, 19]

If $u \in \mathcal{FV}(e')$:

12. $\mathcal{FV}(e'[u/u']) = (\mathcal{FV}(e') - \{u\}) \cup \{u'\}$ [Theorem 4.8, 5]
13. $(\mathcal{FV}(e') - \{u\}) \cap \mathcal{CV}(e[u/u'], v) = \emptyset$ [11]
14. $u' \notin \mathcal{CV}(e[u/u'], v)$ [4]
15. $\mathcal{FV}(e'[u/u']) \cap \mathcal{CV}(e[u/u'], v) = \emptyset$ [12, 13, 14]
16. $\mathit{OKsbs}(e[u/u'], v, e'[u/u'])$ [15]
17. $cx_1, \mathsf{var}\ u' : S, cx_2[u/u'] \vdash ((\lambda v : T.\ e)\ e' \equiv e[v/e'])[u/u']$
[same derivation as 14–21 above for case $u \notin \mathcal{FV}(e')$]

all other rules)

Analogous to TYBOOL: starting with the instance of the rule used to derive the judgement in the antecedent of the implication, we replace u with u' in the premises of the rule instance and then we apply the rule to the transformed premises to obtain the conclusion with u replaced with u'. We use Theorem 4.24 for TYVAR, TYINST, TEDEF, and EXOP and we use Theorem 4.32 for EXIF, THIFSUBST, and THIF.

For TYRESTR, we use the fact that $\mathcal{FV}(r) = \emptyset$ (explicit premise of the rule instance) and so $r[u/u'] = r$ by Theorem 4.1.

For TERESTR, we use the fact that $\mathcal{FV}(r) = \mathcal{FV}(r') = \emptyset$ (by Theorem 4.52) and therefore $(r \equiv r')[u/u'] = (r \equiv r')$ by Theorem 4.1.

For EXSUB we use the fact that $\mathcal{FV}(r) = \emptyset$ by Theorem 4.52 and so $(r\ e)[u/u'] = r\ e[u/u']$ by Theorem 4.1.

For THSUB we use the fact that $\mathcal{FV}(r) = \emptyset$ by Theorem 4.53 and so $(r\ e)[u/u'] = r\ e[u/u']$ by Theorem 4.1.

For THBOOL, THEXT, THREC, and THPROJSUBwe use the fact that the theorem e that is the conclusion of the rule has no free variables (by Theorem 4.31) and so $e[u/u'] = e$ by Theorem 4.1.

Proof of Theorem 4.57

By induction on the derivation of the judgement $cx \vdash \ldots$:

TYINST)

1. $\vdash cx' : \text{CONTEXT}$ [hypothesis]
2. $cx \subseteq cx'$ [hypothesis]
3. $\mathsf{ty}\ \tau : n \in cx$ [premise of TYINST]
4. $\mathsf{ty}\ \tau : n \in cx'$ [3, 2]
5. $|\overline{T}| = n$ [premise of TYINST]
6. $\forall i.\ \ cx \vdash T_i : \text{TYPE}$ [premise of TYINST]
7. $\forall i.\ \ cx' \vdash T_i : \text{TYPE}$ [induction hypothesis, 6]
8. $cx' \vdash \tau[\overline{T}] : \text{TYPE}$ [TYINST, 1, 4, 5, 7]

all other rules except EXABS)

Analogous to TYINST: starting with the rule used to derive the judgement $cx \vdash \ldots$, we replace cx with cx' in the premises of the rule and then we apply the rule to the transformed premises to obtain the conclusion with cx' instead of cx. For EXIF, THIFSUBST, and THIF, we use the fact that

$$\vdash cx' : \text{CONTEXT} \quad \wedge \quad cx \subseteq cx' \quad \wedge \quad cx, \mathsf{ax}\ e \vdash \ldots \quad \Rightarrow \quad \vdash cx', \mathsf{ax}\ e : \text{CONTEXT}$$

which is proved as follows:

1. $cx, \mathsf{ax}\ e \vdash \ldots$ [hypothesis]
2. $\vdash cx, \mathsf{ax}\ e : \text{CONTEXT}$ [Theorem 4.37, 1]
3. $cx \vdash e : \mathsf{Bool}$ [Theorem 4.44, 2]
4. $\vdash cx' : \text{CONTEXT}$ [hypothesis]
5. $cx \subseteq cx'$ [hypothesis]
6. $cx' \vdash e : \mathsf{Bool}$ [induction hypothesis, 5, 4, 3]
7. $\vdash cx', \mathsf{ax}\ e : \text{CONTEXT}$ [CXAX, 4, 6]

(This fact is used twice in each of EXIF, THIFSUBST, and THIF, the first time with $e = e_0$ and the second time with $e = \neg\, e_0$.)

EXABS)

1. $cx, \mathsf{var}\ v : T \vdash e : T'$ [premise of EXABS]
2. $\vdash cx, \mathsf{var}\ v : T : \text{CONTEXT}$ [Theorem 4.37, 1]
3. $cx \vdash T : \text{TYPE}$ [Theorem 4.46, 2]
4. $cx' \vdash T : \text{TYPE}$ [induction hypothesis, 3]
5. $\vdash cx' : \text{CONTEXT}$ [hypothesis]
6. $cx \subseteq cx'$ [hypothesis]
7. $cx \vdash T' : \text{TYPE}$ [premise of EXABS]
8. $cx' \vdash T' : \text{TYPE}$ [induction hypothesis, 7]

If $v \notin \mathcal{V}(cx')$:

9. $\vdash cx', \mathsf{var}\ v : T : \text{CONTEXT}$ [CXVDEC, 5, 4]
10. $cx, \mathsf{var}\ v : T \subseteq cx', \mathsf{var}\ v : T$ [6]

11. $cx', \mathsf{var}\ v{:}T \vdash e : T'$ [induction hypothesis, 1, 9, 10]
12. $cx' \vdash \lambda v{:}T.\ e : T \to T'$ [exAbs, 11, 8]

If $v \in \mathcal{V}(cx')$, pick a fresh variable v':

9. $cx, \mathsf{var}\ v'{:}T \vdash e[v/v'] : T'$ [Theorem 4.56, 1]
10. $\vdash cx', \mathsf{var}\ v'{:}T : \textsc{context}$ [cxVdec, 5, 4]
11. $cx, \mathsf{var}\ v'{:}T \subseteq cx', \mathsf{var}\ v'{:}T$ [6]
12. $cx', \mathsf{var}\ v'{:}T \vdash e[v/v'] : T'$ [induction hypothesis, 9, 10, 11]
13. $cx' \vdash \lambda v'{:}T.\ e[v/v'] : T'$ [exAbs, 12, 8]
14. $OKsbs(e, v, v')$ [hypothesis v' fresh]
15. $v \neq v'$ [hypothesis v' fresh]
16. $v \notin \mathcal{FV}(e[v/v'])$ [Theorem 4.9, 14, 15]
17. $v' \notin \mathcal{FV}(e)$ [hypothesis v' fresh]
18. $\mathcal{CV}(e[v/v'], v') = \mathcal{CV}(e, v)$ [Theorem 4.15, 12, 15]
19. $v \notin \mathcal{CV}(e, v)$ [Theorem 4.2]
20. $v \notin \mathcal{CV}(e[v/v'], v')$ [18, 19]
21. $cx' \vdash \lambda v{:}T.\ e[v/v'][v'/v] : T \to T'$ [exAbsAlpha, 13, 16, 20]
22. $e[v/v'][v/v'] = e$ [Theorem 4.14, 14, 17]
23. $cx' \vdash \lambda v{:}T.\ e : T \to T'$ [21, 22]

Proof of Theorem 4.58

By induction on the derivation of the judgements in the antecedents of the implications:

tyBool)

1. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \textsc{context}$ [hypothesis]
2. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 1]
3. $\mathsf{Bool}[\overline{\alpha}/\overline{S}] = \mathsf{Bool}$
4. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \mathsf{Bool}[\overline{\alpha}/\overline{S}] : \textsc{type}$ [2, 3]

tyVar)

Let the instance of tyVar used to derive the judgement be

$$\frac{\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \textsc{context} \qquad \beta \in \mathcal{TV}(cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3)}{cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash \beta : \textsc{type}} \quad (0)$$

The judgement $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \beta[\overline{\alpha}/\overline{S}] : \textsc{type}$ is derived as follows:

1. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \textsc{context}$ [hypothesis]
2. $\beta \in \mathcal{TV}(cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3)$ [0]
3. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \textsc{context}$ [0]

If $\beta \in \mathcal{TV}(cx_1)$:

4. $\beta \notin \overline{\alpha}$ [Theorem 4.40, 3]
5. $\beta[\overline{\alpha}/\overline{S}] = \beta$ [Theorem 4.27, 4]
6. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \beta[\overline{\alpha}/\overline{S}] : \textsc{type}$ [tyVar, 1, hypothesis $\beta \in \mathcal{TV}(cx_1)$, 5]

If $\beta = \alpha_i$:

4. $\beta[\overline{\alpha}/\overline{S}] = S_i$
5. $cx_1, cx_2 \vdash S_i : \text{TYPE}$ [hypothesis]
6. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \beta[\overline{\alpha}/\overline{S}] : \text{TYPE}$ [Theorem 4.57, 5, 1, 4]

If $\beta \in \mathcal{TV}(cx_3)$:

4. $\beta \notin \overline{\alpha}$ [Theorem 4.40, 3]
5. $\beta \in cx_3[\overline{\alpha}/\overline{S}]$
6. $\beta[\overline{\alpha}/\overline{S}] = \beta$ [Theorem 4.27, 4]
7. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \beta[\overline{\alpha}/\overline{S}] : \text{TYPE}$ [tyVar, 1, 5, 6]

teDef)

Let the instance of teDef used to derive the judgement be

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \text{CONTEXT} \\ \mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \\ \forall i.\ \ cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T_i : \text{TYPE}\end{array}}{cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash \tau[\overline{T}] \approx T[\overline{\beta}/\overline{T}]} \quad (0)$$

The judgement $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \tau[\overline{T}][\overline{\alpha}/\overline{S}] \approx T[\overline{\beta}/\overline{T}][\overline{\alpha}/\overline{S}]$ is derived as follows:

1. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \text{CONTEXT}$ [hypothesis]
2. $\forall i.\ \ cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T_i : \text{TYPE}$ [0]
3. $\forall i.\ \ cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash T_i[\overline{\alpha}/\overline{S}] : \text{TYPE}$ [induction hypothesis, 2]
4. $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3$ [0]
5. $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1, cx_3$ [4]

If $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1$:

6. $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}]$ [5]
7. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \tau[\overline{T[\overline{\alpha}/\overline{S}]}] \approx T[\overline{\beta}/\overline{T[\overline{\alpha}/\overline{S}]}]$ [teDef, 1, 6, 3]
8. $\tau[\overline{T[\overline{\alpha}/\overline{S}]}] = \tau[\overline{T}][\overline{\alpha}/\overline{S}]$ [definition of _[_]]
9. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \text{CONTEXT}$ [0]
10. $\overline{\alpha} \cap \mathcal{TV}(cx_1) = \emptyset$ [Theorem 4.40, 9]
11. $\exists cx_1', cx_1''.\ \ cx_1 = (cx_1', \mathsf{def}\ \tau[\overline{\beta}] = T, cx_1'')$ [hypothesis $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_1$]
12. $\mathcal{FTV}(\mathsf{def}\ \tau[\overline{\beta}] = T) \subseteq \mathcal{TV}(cx_1')$ [Theorem 4.55, 1, 11]
13. $\mathcal{FTV}(\mathsf{def}\ \tau[\overline{\beta}] = T) \subseteq \mathcal{TV}(cx_1)$ [12, 11]
14. $\mathcal{FTV}(\mathsf{def}\ \tau[\overline{\beta}] = T) \cap \overline{\alpha} = \emptyset$ [10, 13]
15. $(\mathcal{FTV}(T) - \overline{\beta}) \cap \overline{\alpha} = \emptyset$ [14]
16. $T[\overline{\beta}/\overline{T}][\overline{\alpha}/\overline{S}] = T[\overline{\beta}/\overline{T[\overline{\alpha}/\overline{S}]}]$ [Theorem 4.29, 15]
17. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \tau[\overline{T}][\overline{\alpha}/\overline{S}] \approx T[\overline{\beta}/\overline{T}][\overline{\alpha}/\overline{S}]$ [7, 8, 16]

If $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_3$:

6. $\exists cx_3', cx_3''.\ \ cx_3 = (cx_3', \mathsf{def}\ \tau[\overline{\beta}] = T, cx_3'')$
7. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \text{CONTEXT}$ [0]
8. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3', \mathsf{tvar}\ \overline{\beta} \vdash T : \text{TYPE}$ [Theorem 4.43, 7, 6]
9. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3', \mathsf{tvar}\ \overline{\beta} : \text{CONTEXT}$ [Theorem 4.37, 8]

10. $\overline{\alpha} \cap \overline{\beta} = \emptyset$ [Theorem 4.40, 9]
11. $\mathsf{def}\ \tau[\overline{\beta}] = T[\overline{\alpha}/\overline{S}] \in cx_3[\overline{\alpha}/\overline{S}]$ [10, hypothesis $\mathsf{def}\ \tau[\overline{\beta}] = T \in cx_3$]
12. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \tau[\overline{T}[\overline{\alpha}/\overline{S}]] \approx T[\overline{\alpha}/\overline{S}][\overline{\beta}/\overline{T}[\overline{\alpha}/\overline{S}]]$ [TEDEF, 1, 11, 3]
13. $\tau[\overline{T}[\overline{\alpha}/\overline{S}]] = \tau[\overline{T}][\overline{\alpha}/\overline{S}]$ [definition of $\_[\_]$]
14. $OKsbs(cx_3, \overline{\alpha}, \overline{S})$ [hypothesis]
15. $\forall i.\ \mathcal{FTV}(S_i) \cap \mathcal{CTV}(cx_3, \alpha_i) = \emptyset$ [14, definition of $OKsbs$]
16. $\forall i.\ \mathcal{FTV}(S_i) \cap \mathcal{CTV}(\mathsf{def}\ \tau[\overline{\beta}] = T, \alpha_i) = \emptyset$ [15, 6]
17. $\forall i.\ \mathcal{FTV}(S_i) \cap \begin{cases} \overline{\beta} & \text{if} \quad \alpha_i \in \mathcal{FTV}(T) - \overline{\beta} \\ \emptyset & \text{otherwise} \end{cases} = \emptyset$ [16]
18. $\forall \alpha_i \in \mathcal{FTV}(T) - \overline{\beta}.\ \mathcal{FTV}(S_i) \cap \overline{\beta} = \emptyset$ [17]
19. $\forall \alpha_i \in \mathcal{FTV}(T).\ \mathcal{FTV}(S_i) \cap \overline{\beta} = \emptyset$ [18, 10]
20. $T[\overline{\beta}/\overline{T}][\overline{\alpha}/\overline{S}] = T[\overline{\alpha}/\overline{S}][\overline{\beta}/\overline{T}[\overline{\alpha}/\overline{S}]]$ [Theorem 4.30, 10, 19]
21. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \tau[\overline{T}][\overline{\alpha}/\overline{S}] \approx T[\overline{\beta}/\overline{T}][\overline{\alpha}/\overline{S}]$ [12, 13, 20]

EXVAR)

Let the instance of EXVAR used to derive the judgement be

$$\frac{\begin{array}{c}\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \text{CONTEXT} \\ \mathsf{var}\ v{:}T \in cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3\end{array}}{cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash v : T} \quad (0)$$

The judgement $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash v : T[\overline{\alpha}/\overline{S}]$ is derived as follows:

1. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \text{CONTEXT}$ [hypothesis]
2. $\mathsf{var}\ v{:}T \in cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3$ [0]
3. $\mathsf{var}\ v{:}T \in cx_1, cx_3$ [2]

If $\mathsf{var}\ v{:}T \in cx_1$:

4. $\mathsf{var}\ v{:}T \in cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}]$
5. $cx_1, cx_3, cx_3[\overline{\alpha}/\overline{S}] \vdash v : T$ [EXVAR, 1, 4]
6. $\exists cx_1', cx_1''.\ \ cx_1 = cx_1', \mathsf{var}\ v{:}T, cx_1''$ [hypothesis $\mathsf{var}\ v{:}T \in cx_1$]
7. $\mathcal{FTV}(T) \subseteq \mathcal{TV}(cx_1')$ [Theorem 4.55, 1, 6]
8. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 : \text{CONTEXT}$ [0]
9. $\overline{\alpha} \cap \mathcal{TV}(cx_1) = \emptyset$ [Theorem 4.40, 8]
10. $\mathcal{FTV}(T) \cap \overline{\alpha} = \emptyset$ [7, 6, 9]
11. $T[\overline{\alpha}/\overline{S}] = T$ [Theorem 4.27, 10]
12. $cx_1, cx_3, cx_3[\overline{\alpha}/\overline{S}] \vdash v : T[\overline{\alpha}/\overline{S}]$ [5, 11]

If $\mathsf{var}\ v{:}T \in cx_3$:

4. $\mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}] \in cx_3[\overline{\alpha}/\overline{S}]$
5. $\mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}] \in cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}]$ [4]
6. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash v : T[\overline{\alpha}/\overline{S}]$ [EXVAR, 1, 5]

EXOP, THAX)

Analogous to TEDEF.

exAbs)

Let the instance of exAbs used to defive the judgement be

$$\frac{\begin{array}{c} cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3, \mathsf{var}\ v{:}T \vdash e : T' \\ cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T' : \text{TYPE} \end{array}}{cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash \lambda v{:}T.\ e : T \to T'} \quad (0)$$

The judgement $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v{:}T.\ e)[\overline{\alpha}/\overline{S}] : (T \to T')[\overline{\alpha}/\overline{S}]$ is derived as follows:

1. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3, \mathsf{var}\ v{:}T \vdash e : T'$ [0]
2. $\vdash cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3, \mathsf{var}\ v{:}T : \text{CONTEXT}$ [Theorem 4.37, 1]
3. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T : \text{TYPE}$ [Theorem 4.46, 2]
4. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash T[\overline{\alpha}/\overline{S}] : \text{TYPE}$ [induction hypothesis, 3]
5. $v \notin cx_1, cx_3$ [Theorem 4.41, 2]
6. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash T' : \text{TYPE}$ [0]
7. $cx_1, cx_2, cx_2[\overline{\alpha}/\overline{S}] \vdash T' : \text{TYPE}$ [induction hypothesis, 6]

If $v \notin cx_2$:

8. $v \notin cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}]$ [5]
9. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] : \text{CONTEXT}$ [hypothesis]
10. $\vdash cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}], \mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}] : \text{CONTEXT}$ [cxVdec, 9, 8, 4]
11. $OKsbs(cx_3, \overline{\alpha}, \overline{S})$ [hypothesis]
12. $\overline{\alpha} \cap \mathcal{TV}(cx_3) = \emptyset$ [11]
13. $\overline{\alpha} \cap \mathcal{TV}(cx_3, \mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}]) = \emptyset$ [12]
14. $\forall i.\ \ \mathcal{FTV}(S_i) \cap \mathcal{CTV}(cx_3, \alpha_i) = \emptyset$ [11]
15. $\forall i.\ \ \mathcal{CTV}(\mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}], \alpha_i) = \emptyset$ [definition of $\mathcal{CTV}$]
16. $\forall i.\ \ \mathcal{FTV}(S_i) \cap \mathcal{CTV}((cx_3, \mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}]), \alpha_i) = \emptyset$ [14, 15]
17. $OKsbs((cx_3, \mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}]), \overline{\alpha}, \overline{S})$ [13, 16]
18. $cx_1, cx_3, cx_3[\overline{\alpha}/\overline{S}], \mathsf{var}\ v{:}T[\overline{\alpha}/\overline{S}] \vdash e[\overline{\alpha}/\overline{S}] : T'[\overline{\alpha}/\overline{S}]$ [induction hypothesis, 10, 17, 1]
19. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \lambda v{:}T[\overline{\alpha}/\overline{S}].\ e[\overline{\alpha}/\overline{S}] : T[\overline{\alpha}/\overline{S}] \to T'[\overline{\alpha}/\overline{S}]$ [exAbs, 18, 7]
20. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v{:}T.\ e)[\overline{\alpha}/\overline{S}] : (T \to T')[\overline{\alpha}/\overline{S}]$ [19]

If $v \in cx_2$, pick a fresh variable v':

8. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3, \mathsf{var}\ v'{:}T \vdash e[v/v'] : T'$ [Theorem 4.56, 1]
9. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \lambda v'{:}T[\overline{\alpha}/\overline{S}].\ e[v/v'][\overline{\alpha}/\overline{S}] : T[\overline{\alpha}/\overline{S}] \to T'[\overline{\alpha}/\overline{S}]$
[analogous derivation as 8–19 above for case $v \notin cx_2$]
10. v' fresh [hypothesis]
11. $OKsbs(e, v, v')$ [10]
12. $v \notin \mathcal{FV}(e[v/v'])$ [Theorem 4.9, 11, 10]
13. $v \notin \mathcal{FV}(e[v/v'][\overline{\alpha}/\overline{S}])$ [Theorem 4.18, 12]
14. $v \notin \mathcal{CV}(e, v)$ [Theorem 4.2]
15. $v \notin \mathcal{CV}(e[v/v'], v')$ [Theorem 4.15, 11, 10, 14]
16. $v \notin \mathcal{CV}(e[v/v'][\overline{\alpha}/\overline{S}], v')$ [Theorem 4.19, 15]
17. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \lambda v{:}T[\overline{\alpha}/\overline{S}].\ e[v/v'][\overline{\alpha}/\overline{S}][v'/v] : (T \to T')[\overline{\alpha}/\overline{S}]$
[exAbsAlpha, 9, 13, 16]

18. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \lambda v\!:\!T[\overline{\alpha}/\overline{S}].\ e[v/v'][v'/v][\overline{\alpha}/\overline{S}] : (T \to T')[\overline{\alpha}/\overline{S}]$ [Theorem 4.23, 17]
19. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash \lambda v\!:\!T[\overline{\alpha}/\overline{S}].\ e[\overline{\alpha}/\overline{S}] : (T \to T')[\overline{\alpha}/\overline{S}]$ [Theorem 4.14, 18, 11, 10]
20. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v\!:\!T.\ e)[\overline{\alpha}/\overline{S}] : (T \to T')[\overline{\alpha}/\overline{S}]$ [19]

thAbs)

Let the instance of thAbs used to derive the judgement be

$$\frac{\begin{array}{c} cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash (\lambda v\!:\!T.\ e)\ e' : \ldots \\ OKsbs(e, v, e') \end{array}}{cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash (\lambda v\!:\!T.\ e)\ e' \equiv e[v/e']} \quad (0)$$

The judgement $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash ((\lambda v\!:\!T.\ e)\ e' \equiv e[v/e'])[\overline{\alpha}/\overline{S}]$ is derived as follows:

1. $cx_1, \mathsf{tvar}\ \overline{\alpha}, cx_3 \vdash (\lambda v\!:\!T.\ e)\ e' : \ldots$ [0]
2. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash ((\lambda v\!:\!T.\ e)\ e')[\overline{\alpha}/\overline{S}] : \ldots$ [induction hypothesis, 1]
3. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v\!:\!T[\overline{\alpha}/\overline{S}].\ e[\overline{\alpha}/\overline{S}])\ e'[\overline{\alpha}/\overline{S}] : \ldots$ [2]
4. $OKsbs(e, v, e')$ [0]
5. $\mathcal{FV}(e') \cap \mathcal{CV}(e, v) = \emptyset$ [4]
6. $\mathcal{FV}(e'[\overline{\alpha}/\overline{S}]) \cap \mathcal{CV}(e[\overline{\alpha}/\overline{S}], v) = \emptyset$ [Theorem 4.18, Theorem 4.19]
7. $OKsbs(e[\overline{\alpha}/\overline{S}], v, e'[\overline{\alpha}/\overline{S}])$ [6]
8. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v\!:\!T[\overline{\alpha}/\overline{S}].\ e[\overline{\alpha}/\overline{S}])\ e'[\overline{\alpha}/\overline{S}] \equiv e[\overline{\alpha}/\overline{S}][v/e'[\overline{\alpha}/\overline{S}]]$ [thAbs, 3, 7]
9. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash (\lambda v\!:\!T[\overline{\alpha}/\overline{S}].\ e[\overline{\alpha}/\overline{S}])\ e'[\overline{\alpha}/\overline{S}] \equiv e[v/e'][\overline{\alpha}/\overline{S}]$ [Theorem 4.22, 8]
10. $cx_1, cx_2, cx_3[\overline{\alpha}/\overline{S}] \vdash ((\lambda v\!:\!T.\ e)\ e' \equiv e[v/e'])[\overline{\alpha}/\overline{S}]$ [9]

all other rules)

Analogous to tyBool: starting with the instance of the rule used to derive the judgement in the antecedent of the implication, we replace $\mathsf{tvar}\ \overline{\alpha}$ with cx_2 and $\overline{\alpha}$ with $\overline{S}$ in the premises of the rule instance and then we apply the rule to the transformed premises to obtain the conclusion with $\overline{\alpha}$ replaced with $\overline{S}$. We use Theorem 4.18 for tyRestr. We use Theorem 4.34 for stRefl, stArr, stRec, exIf, exThe, thIfSubst, thBool, thExt, thIf, thRec, and thProjSub. We use Theorem 4.18, Theorem 4.19, and Theorem 4.23 for exAbsAlpha.

Proof of Theorem 4.59

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 1]
3. $\vdash cx, \mathsf{var}\ v\!:\!\mathsf{Bool} : \textsc{context}$ [cxVdec, 1, hypothesis, 2]

Proof of Theorem 4.60

1. $cx \vdash e : \mathsf{Bool}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $\vdash cx, \mathsf{ax}\ e : \textsc{context}$ [cxAx, 2, 1]

Proof of Theorem 4.61

If $v \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $\vdash cx, \mathsf{var}\ v{:}\mathsf{Bool} : \textsc{context}$ [cxVdecBool, 1, hypothesis]
3. $cx, \mathsf{var}\ v{:}\mathsf{Bool} \vdash v : \mathsf{Bool}$ [exVar, 2]
4. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 1]
5. $cx \vdash \lambda v{:}\mathsf{Bool}.\ v : \mathsf{Bool} \to \mathsf{Bool}$ [exAbs, 3, 4]

If instead $v \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx \vdash \lambda v'{:}\mathsf{Bool}.\ v' : \mathsf{Bool} \to \mathsf{Bool}$, where $v' \neq v$ is a fresh variable not declared in cx, i.e. $v' \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v' to v.

Proof of Theorem 4.62

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $cx \vdash \lambda\gamma{:}\mathsf{Bool}.\ \gamma : \mathsf{Bool} \to \mathsf{Bool}$ [exIdBool, 1]
3. $cx \vdash (\lambda\gamma{:}\mathsf{Bool}.\ \gamma \equiv \lambda\gamma{:}\mathsf{Bool}.\ \gamma) : \mathsf{Bool}$ [exEq, 2, 2]

Proof of Theorem 4.63

If $v \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $cx \vdash T : \textsc{type}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $\vdash cx, \mathsf{var}\ v{:}T : \textsc{context}$ [cxVdec, 2, hypothesis, 1]
4. $cx, \mathsf{var}\ v{:}T \vdash \mathsf{true} : \mathsf{Bool}$ [exTrue, 3]
5. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 2]
6. $cx \vdash \lambda v{:}T.\ \mathsf{true} : T \to \mathsf{Bool}$ [exAbs, 4, 5]

If instead $v \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx \vdash \lambda v'{:}T.\ \mathsf{true} : T \to \mathsf{Bool}$, where $v' \neq v$ is a fresh variable not declared in cx, i.e. $v' \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v' to v.

Proof of Theorem 4.64

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $cx \vdash \lambda\gamma{:}\mathsf{Bool}.\ \gamma : \mathsf{Bool} \to \mathsf{Bool}$ [exIdBool, 1]
3. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 1]
4. $cx \vdash \lambda\gamma{:}\mathsf{Bool}.\ \mathsf{true} : \mathsf{Bool} \to \mathsf{Bool}$ [exConstTrue, 3]
5. $cx \vdash (\lambda\gamma{:}\mathsf{Bool}.\ \gamma \equiv \lambda\gamma{:}\mathsf{Bool}.\ \mathsf{true}) : \mathsf{Bool}$ [exEq, 2, 4]

Proof of Theorem 4.65

If $\gamma \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $\vdash cx, \mathsf{var}\ \gamma{:}\mathsf{Bool} : \textsc{context}$ [cxVdecBool, 1, hypothesis]
3. $cx, \mathsf{var}\ \gamma{:}\mathsf{Bool} \vdash \gamma : \mathsf{Bool}$ [exVar, 2]
4. $cx, \mathsf{var}\ \gamma{:}\mathsf{Bool} \vdash \mathsf{false} : \mathsf{Bool}$ [exFalse, 2]
5. $cx, \mathsf{var}\ \gamma{:}\mathsf{Bool} \vdash \mathsf{true} : \mathsf{Bool}$ [exTrue, 2]
6. $cx, \mathsf{var}\ \gamma{:}\mathsf{Bool} \vdash \mathsf{if}\ \gamma\ \mathsf{false}\ \mathsf{true} : \mathsf{Bool}$ [exIf0, 3, 4, 5]
7. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 1]
8. $cx \vdash \lambda\gamma{:}\mathsf{Bool}.\ (\mathsf{if}\ \gamma\ \mathsf{false}\ \mathsf{true}) : \mathsf{Bool} \rightarrow \mathsf{Bool}$ [exAbs, 6, 7]

If instead $\gamma \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx \vdash \lambda v{:}\mathsf{Bool}.\ (\mathsf{if}\ v\ \mathsf{false}\ \mathsf{true}) : \mathsf{Bool} \rightarrow \mathsf{Bool}$, where $v \neq \gamma$ is a fresh variable not declared in cx, i.e. $v \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v to γ.

Proof of Theorem 4.66

1. $cx \vdash e : \mathsf{Bool}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $cx \vdash \neg : \mathsf{Bool} \rightarrow \mathsf{Bool}$ [exNot, 2]
4. $cx \vdash \neg\ e : \mathsf{Bool}$ [exApp, 3, 1]

Proof of Theorem 4.67

1. $cx \vdash e_1 : \mathsf{Bool}$ [hypothesis]
2. $cx \vdash \neg\ e_1 : \mathsf{Bool}$ [exNeg, 1]
3. $\vdash cx, \mathsf{ax}\ \neg\ e_1 : \textsc{context}$ [cxAx0, 2]
4. $cx, \mathsf{ax}\ \neg\ e_1 \vdash \mathsf{false} : \mathsf{Bool}$ [exFalse, 3]
5. $cx, \mathsf{ax}\ e_1 \vdash e_2 : \mathsf{Bool}$ [hypothesis]
6. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
7. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 6]
8. $cx \vdash \mathsf{if}\ e_1\ e_2\ \mathsf{false} : \mathsf{Bool}$ [exIf, 1, 5, 4, 7]

Proof of Theorem 4.68

1. $cx \vdash e_1 : \mathsf{Bool}$ [hypothesis]
2. $cx \vdash e_2 : \mathsf{Bool}$ [hypothesis]
3. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
4. $cx \vdash \mathsf{false} : \mathsf{Bool}$ [exFalse, 3]
5. $cx \vdash \mathsf{if}\ e_1\ e_2\ \mathsf{false} : \mathsf{Bool}$ [exIf0, 1, 2, 4]

Proof of Theorem 4.69

1. $cx \vdash e_1 : \textsf{Bool}$ [hypothesis]
2. $\vdash cx, \textsf{ax}\ e_1 : \textsc{context}$ [cxAx0, 1]
3. $cx, \textsf{ax}\ e_1 \vdash \textsf{true} : \textsf{Bool}$ [exTrue, 2]
4. $cx, \textsf{ax}\ \neg\ e_1 \vdash e_2 : \textsf{Bool}$ [hypothesis]
5. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
6. $cx \vdash \textsf{Bool} : \textsc{type}$ [tyBool, 5]
7. $cx \vdash \textsf{if}\ e_1\ \textsf{true}\ e_2 : \textsf{Bool}$ [exIf, 1, 3, 4, 6]

Proof of Theorem 4.70

1. $cx \vdash e_1 : \textsf{Bool}$ [hypothesis]
2. $cx \vdash e_2 : \textsf{Bool}$ [hypothesis]
3. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
4. $cx \vdash \textsf{true} : \textsf{Bool}$ [exTrue, 3]
5. $cx \vdash \textsf{if}\ e_1\ \textsf{true}\ e_2 : \textsf{Bool}$ [exIf0, 1, 4, 2]

Proof of Theorem 4.71

1. $cx \vdash e_1 : \textsf{Bool}$ [hypothesis]
2. $cx \vdash \neg\ e_1 : \textsf{Bool}$ [exNeg, 1]
3. $\vdash cx, \textsf{ax}\ \neg\ e_1 : \textsc{context}$ [cxAx0, 2]
4. $cx, \textsf{ax}\ \neg\ e_1 \vdash \textsf{true} : \textsf{Bool}$ [exTrue, 3]
5. $cx, \textsf{ax}\ e_1 \vdash e_2 : \textsf{Bool}$ [hypothesis]
6. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
7. $cx \vdash \textsf{Bool} : \textsc{type}$ [tyBool, 6]
8. $cx \vdash \textsf{if}\ e_1\ e_2\ \textsf{true} : \textsf{Bool}$ [exIf, 1, 5, 4, 7]

Proof of Theorem 4.72

1. $cx \vdash e_1 : \textsf{Bool}$ [hypothesis]
2. $cx \vdash e_2 : \textsf{Bool}$ [hypothesis]
3. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
4. $cx \vdash \textsf{true} : \textsf{Bool}$ [exTrue, 3]
5. $cx \vdash \textsf{if}\ e_1\ e_2\ \textsf{true} : \textsf{Bool}$ [exIf0, 1, 2, 4]

Proof of Theorem 4.73

We first prove the derived rule

$$\frac{\begin{array}{c}\vdash cx, \mathsf{var}\ v\!:\!\mathsf{Bool} : \textsc{context} \\ v' \neq v\end{array}}{cx, \mathsf{var}\ v\!:\!\mathsf{Bool} \vdash \lambda v'\!:\!\mathsf{Bool}.\ v \equiv v' : \mathsf{Bool} \to \mathsf{Bool}} \quad (*)$$

If $v' \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $\vdash cx, \mathsf{var}\ v\!:\!\mathsf{Bool} : \textsc{context}$ [hypothesis]
2. $\vdash cx, \mathsf{var}\ v\!:\!\mathsf{Bool}, \mathsf{var}\ v'\!:\!\mathsf{Bool} : \textsc{context}$ [cxVdecBool, 1, hypothesis]
3. $cx, \mathsf{var}\ v\!:\!\mathsf{Bool}, \mathsf{var}\ v'\!:\!\mathsf{Bool} \vdash v : \mathsf{Bool}$ [exVar, 2]
4. $cx, \mathsf{var}\ v\!:\!\mathsf{Bool}, \mathsf{var}\ v'\!:\!\mathsf{Bool} \vdash v' : \mathsf{Bool}$ [exVar, 2]
5. $cx, \mathsf{var}\ v\!:\!\mathsf{Bool}, \mathsf{var}\ v'\!:\!\mathsf{Bool} \vdash v \equiv v' : \mathsf{Bool}$ [exEq, 3, 4]
6. $cx, \mathsf{var}\ v\!:\!\mathsf{Bool} \vdash \mathsf{Bool} : \textsc{type}$ [Theorem 4.46, 2]
7. $cx, \mathsf{var}\ v\!:\!\mathsf{Bool} \vdash \lambda v'\!:\!\mathsf{Bool}.\ v \equiv v' : \mathsf{Bool} \to \mathsf{Bool}$ [exAbs, 5, 6]

If instead $v' \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx, \mathsf{var}\ v\!:\!\mathsf{Bool} \vdash \lambda v''\!:\!\mathsf{Bool}.\ v \equiv v'' : \mathsf{Bool} \to \mathsf{Bool}$, where $v'' \neq v'$ is a fresh variable not declared in cx, i.e. $v'' \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v'' to v'. This concludes the proof of rule $*$.

We now prove exIff. If $\gamma \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $\vdash cx : \textsc{context}$ [hypothesis]
2. $\vdash cx, \mathsf{var}\ \gamma\!:\!\mathsf{Bool} : \textsc{context}$ [cxVdecBool, 1]
3. $cx, \mathsf{var}\ \gamma\!:\!\mathsf{Bool} \vdash \lambda \gamma'\!:\!\mathsf{Bool}.\ \gamma \equiv \gamma' : \mathsf{Bool} \to \mathsf{Bool}$ [$*$, 2, $\gamma \neq \gamma'$]
4. $cx \vdash \mathsf{Bool} : \textsc{type}$ [Theorem 4.46, 2]
5. $cx \vdash \lambda \gamma\!:\!\mathsf{Bool}.\ \lambda \gamma'\!:\!\mathsf{Bool}.\ \gamma \equiv \gamma' : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ [exAbs, 3, 4]

If instead $\gamma \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx \vdash \lambda v\!:\!\mathsf{Bool}.\ \lambda \gamma'\!:\!\mathsf{Bool}.\ v \equiv \gamma' : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$, where $v \neq \gamma$ is a fresh variable not declared in cx, i.e. $v \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v to γ.

Proof of Theorem 4.74

1. $cx \vdash e_1 : \mathsf{Bool}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $cx \vdash \Leftrightarrow : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}$ [exIff, 2]
4. $cx \vdash \Leftrightarrow\ e_1 : \mathsf{Bool} \to \mathsf{Bool}$ [exApp, 3, 1]
5. $cx \vdash e_2 : \mathsf{Bool}$ [hypothesis]
6. $cx \vdash \Leftrightarrow\ e_1\ e_2 : \mathsf{Bool}$ [exApp, 4, 5]

Proof of Theorem 4.75

1. $cx \vdash e_1 : T$ [hypothesis]
2. $cx \vdash e_2 : T$ [hypothesis]
3. $cx \vdash e_1 \equiv e_2 : \mathsf{Bool}$ [exEq, 1, 2]
4. $cx \vdash \neg\,(e_1 \equiv e_2) : \mathsf{Bool}$ [exNeg, 3]

Proof of Theorem 4.76

If $\psi \notin \mathcal{V}(cx)$, the rule is derived as follows:

1. $cx \vdash T : \textsc{type}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 2]
4. $cx \vdash T \to \mathsf{Bool} : \textsc{type}$ [tyArr, 1, 3]
5. $\vdash cx, \mathsf{var}\ \psi{:}T \to \mathsf{Bool} : \textsc{context}$ [cxVdec, 2, 4, hypothesis]
6. $cx, \mathsf{var}\ \psi{:}T \to \mathsf{Bool} \vdash \psi : T \to \mathsf{Bool}$ [exVar, 5]
7. $cx, \mathsf{var}\ \psi{:}T \to \mathsf{Bool} \vdash T : \textsc{type}$ [Theorem 4.57, 1, 5]
8. $cx, \mathsf{var}\ \psi{:}T \to \mathsf{Bool} \vdash \lambda\gamma{:}T.\ \mathsf{true} : T \to \mathsf{Bool}$ [exConstTrue, 7]
9. $cx, \mathsf{var}\ \psi{:}T \to \mathsf{Bool} \vdash \psi \equiv \lambda\gamma{:}T.\ \mathsf{true} : \mathsf{Bool}$ [exEq, 6, 8]
10. $cx \vdash \lambda\psi{:}T \to \mathsf{Bool}.\ (\psi \equiv \lambda\gamma{:}T.\ \mathsf{true}) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exAbs, 9, 3]

If instead $\psi \in \mathcal{V}(cx)$, we can use a very analogous derivation to derive $cx \vdash \lambda v{:}T \to \mathsf{Bool}.\ (v \equiv \lambda\gamma{:}T.\ \mathsf{true}) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$, where $v \neq \psi$ is a fresh variable not declared in cx, i.e. $v \notin \mathcal{V}(cx)$; then we use rule exAbsAlpha to rename v to ψ.

Proof of Theorem 4.77

1. $cx, \mathsf{var}\ v{:}T \vdash e : \mathsf{Bool}$ [hypothesis]
2. $\vdash cx, \mathsf{var}\ v{:}T : \textsc{context}$ [Theorem 4.37, 1]
3. $\vdash cx : \textsc{context}$ [Theorem 4.36, 2]
4. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 3]
5. $cx \vdash \lambda v{:}T.\ e : T \to \mathsf{Bool}$ [exAbs, 1, 4]
6. $cx \vdash T : \textsc{type}$ [Theorem 4.46, 2]
7. $cx \vdash \forall_T : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exFA, 6]
8. $cx \vdash \forall_T\ (\lambda v{:}T.\ e) : \mathsf{Bool}$ [exApp, 7, 5]

Proof of Theorem 4.78

Let $\widetilde{\psi}, \widetilde{\gamma} \in \mathcal{N}$ such that $\widetilde{\psi} \neq \widetilde{\gamma}$ and $\{\widetilde{\psi}, \widetilde{\gamma}\} \cap (\mathcal{V}(cx) \cup \{\psi, \gamma\}) = \emptyset$. The rule is derived as follows:

1. $cx \vdash T : \textsc{type}$ [hypothesis]
2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]
3. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 2]
4. $cx \vdash T \to \mathsf{Bool} : \textsc{type}$ [tyArr, 1, 3]
5. $\vdash cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} : \textsc{context}$ [cxVdec, 2, 4]
6. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash T : \textsc{type}$ [Theorem 4.57, 1, 5]
7. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \forall_T : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exFA, 6]

8. $\vdash cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T : \textsc{context}$ [cxVdec, 5, 6]

9. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\psi} : T \to \mathsf{Bool}$ [exVar, 8]

10. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\gamma} : T$ [exVar, 8]

11. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\psi}\ \widetilde{\gamma} : \mathsf{Bool}$ [exApp, 9, 10]

12. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \neg\ (\widetilde{\psi}\ \widetilde{\gamma}) : \mathsf{Bool}$ [exNeg, 11]

13. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 5]

14. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \lambda\widetilde{\gamma}{:}T.\ \neg\ (\widetilde{\psi}\ \widetilde{\gamma}) : T \to \mathsf{Bool}$ [exAbs, 12, 13]

15. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \lambda\gamma{:}T.\ \neg\ (\widetilde{\psi}\ \gamma) : T \to \mathsf{Bool}$ [exAbsAlpha, 14]

16. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \forall\gamma{:}T.\ \neg\ (\widetilde{\psi}\ \gamma) : \mathsf{Bool}$ [exApp, 7, 15]

17. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \neg\ (\forall\gamma{:}T.\ \neg\ (\widetilde{\psi}\ \gamma)) : \mathsf{Bool}$ [exNeg, 16]

18. $cx \vdash \lambda\widetilde{\psi}{:}T \to \mathsf{Bool}.\ \neg\ (\forall\gamma{:}T.\ \neg\ (\widetilde{\psi}\ \gamma)) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exAbs, 17, 3]

19. $cx \vdash \lambda\psi{:}T \to \mathsf{Bool}.\ \neg\ (\forall\gamma{:}T.\ \neg\ (\psi\ \gamma)) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exAbsAlpha, 18]

Proof of Theorem 4.79

Analogous to Theorem 4.77, using exEX instead of exFA.

Proof of Theorem 4.80

Let $\widetilde{\psi}, \widetilde{\gamma}, \widetilde{\gamma}' \in \mathcal{N}$ such that they are distinct and $\{\widetilde{\psi}, \widetilde{\gamma}, \widetilde{\gamma}'\} \cap (\mathcal{V}(cx) \cup \{\psi, \gamma, \gamma'\}) = \emptyset$. The rule is derived as follows:

1. $cx \vdash T : \textsc{type}$ [hypothesis]

2. $\vdash cx : \textsc{context}$ [Theorem 4.37, 1]

3. $cx \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 2]

4. $cx \vdash T \to \mathsf{Bool} : \textsc{type}$ [tyArr, 1, 3]

5. $\vdash cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} : \textsc{context}$ [cxVdec, 2, 4]

6. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash T : \textsc{type}$ [Theorem 4.57, 1, 5]

7. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \exists_T : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exEX, 6]

8. $\vdash cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T : \textsc{context}$ [cxVdec, 5, 6]

9. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\psi} : T \to \mathsf{Bool}$ [exVar, 8]

10. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\gamma} : T$ [exVar, 8]

11. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\psi}\ \widetilde{\gamma} : \mathsf{Bool}$ [exApp, 9, 10]

12. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash T : \textsc{type}$ [Theorem 4.57, 6, 8]

13. $\vdash cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T : \textsc{context}$ [cxVdec, 8, 12]

14. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\psi} : T \to \mathsf{Bool}$ [exVar, 13]

15. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\gamma} : T$ [exVar, 13]

16. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\gamma}' : T$ [exVar, 13]

17. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\gamma}' \equiv \widetilde{\gamma} : \mathsf{Bool}$ [exEq, 16, 15]

18. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\psi}\ \widetilde{\gamma}' : \mathsf{Bool}$ [exApp, 14, 16]

19. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T, \mathsf{var}\ \widetilde{\gamma}'{:}T \vdash \widetilde{\psi}\ \widetilde{\gamma}' \Rightarrow \widetilde{\gamma}' \equiv \widetilde{\gamma} : \mathsf{Bool}$ [exImpl0, 18, 17]

20. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \forall_T : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exFA, 12]

21. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 8]

22. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \lambda\widetilde{\gamma}'{:}T.\ (\widetilde{\psi}\ \widetilde{\gamma}' \Rightarrow \widetilde{\gamma}' \equiv \widetilde{\gamma}) : T \to \mathsf{Bool}$ [exAbs, 19, 21]

23. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \lambda\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \widetilde{\gamma}) : T \to \mathsf{Bool}$ [exAbsAlpha, 22]

24. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \widetilde{\gamma}) : \mathsf{Bool}$ [exApp, 20, 23]

25. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool}, \mathsf{var}\ \widetilde{\gamma}{:}T \vdash \widetilde{\psi}\ \widetilde{\gamma} \wedge \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \widetilde{\gamma}) : \mathsf{Bool}$ [exConj0, 11, 24]

26. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \exists_T : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exEX, 6]

27. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \mathsf{Bool} : \textsc{type}$ [tyBool, 5]

28. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \lambda\widetilde{\gamma}{:}T.\ (\widetilde{\psi}\ \widetilde{\gamma} \wedge \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \widetilde{\gamma})) : T \to \mathsf{Bool}$ [exAbs, 25, 27]

29. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \lambda\gamma{:}T.\ (\widetilde{\psi}\ \gamma \wedge \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \gamma)) : T \to \mathsf{Bool}$ [exAbsAlpha, 28]

30. $cx, \mathsf{var}\ \widetilde{\psi}{:}T \to \mathsf{Bool} \vdash \exists\gamma{:}T.\ (\widetilde{\psi}\ \gamma \wedge \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \gamma)) : \mathsf{Bool}$ [exApp, 26, 29]

31. $cx \vdash \lambda\widetilde{\psi}{:}T \to \mathsf{Bool}.\ \exists\gamma{:}T.\ (\widetilde{\psi}\ \gamma \wedge \forall\gamma'{:}T.\ (\widetilde{\psi}\ \gamma' \Rightarrow \gamma' \equiv \gamma)) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$ [exAbs, 30, 3]

32. $cx \vdash \lambda\psi{:}T \to \mathsf{Bool}.\ \exists\gamma{:}T.\ (\psi\ \gamma \wedge \forall\gamma'{:}T.\ (\psi\ \gamma' \Rightarrow \gamma' \equiv \gamma)) : (T \to \mathsf{Bool}) \to \mathsf{Bool}$
[exAbsAlpha, 31]

Proof of Theorem 4.81

Analogous to Theorem 4.77, using exEX1 instead of exFA.

Proof of Theorem 4.82

1. $cx \vdash \prod_i f_i\ T_i : \textsc{type}$ [hypothesis]
2. $cx \vdash \mathsf{proj}\ f_j : \prod_i f_i\ T_i \to T_j$ [exProj, 1]
3. $cx \vdash e : \prod_i f_i\ T_i$ [hypothesis]
4. $cx \vdash \mathsf{proj}\ f_j\ e : T_j$ [exApp, 2, 3]

Proof of Theorem 6.1

By straightforward calculation. Recall that a tuple $\langle\overline{e}\rangle$ is implicitly decorated by types $\overline{T}$.

Proof of Theorem 6.2

By straightforward calculation, using Theorem 4.1 where needed.

Acknowledgments

Peter Homeier has carefully reviewed this document and has provided very useful feedback and suggestions. He has also formalized the Metaslang logic in the HOL theorem prover and is using HOL to prove the (meta) theorems contained in this document.

References

[1] Kestrel Institute and Kestrel Technology LLC. *Specware 4.1 Language Manual*. Available at www.specware.org.

[2] Peter Andrews. *An Introduction to Mathematical Logic and Type Theory: To Thruth Through Proof*. Academic Press, 1986.

[3] Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical Report CSL–97–2R, SRI International, August 1997. Revised March 1999.

[4] *The HOL System Description*, July 1997.