8-2009

A Syntactic Account of Singleton Types via Hereditary Substitution

Karl Crary
Carnegie Mellon University

Published In

Proceedings of the Fourth international Workshop on Logical Frameworks and Meta-Languages: theory and Practice, LFMTP '09., 21-29.

A Syntactic Account of Singleton Types via Hereditary Substitution*

Karl Crary

Carnegie Mellon University

Abstract

We give a syntactic proof of decidability and consistency of equivalence for the singleton type calculus, which lies at the foundation of modern module systems such as that of ML. Unlike existing proofs, which work by constructing a model, our syntactic proof makes few demands on the underlying proof theory and mathematical foundation. Consequently, it can be — and has been — entirely formulated in the Twelf meta-logic, and provides an important piece of a Twelf-formalized type-safety proof for Standard ML.

The proof works by translation of the singleton type calculus into a canonical presentation, adapted from work on logical frameworks, in which equivalent terms are written identically. Canonical forms are not preserved under standard substitution, so we employ an alternative definition of substitution called *hereditary substitution,* which contracts redices that arise during substitution.

1 Introduction

Modern module systems provide means for controlled propagation of type information across module boundaries. For example, in Standard ML one can write the interface:

```
signature SIG =
   sig
     type s = int
     type t
     type u = s -> t
     ...
   end
```

A module matching this signature provides three types: `s` is known to be `int`, `t` is abstract, and `u` is known in terms of `s` and `t`. In writing such an interface, the programmer decides exactly how much type information to reveal, and conversely, how much to hold abstract.

Singleton kinds provide a direct and expressive type-theoretic foundation for expressing information about type information. The idea is for any type τ, one may form the singleton kind $\mathsf{S}(\tau)$, which contains just τ and any other types equivalent to it.

In the presence of singleton kinds, type equivalences may be given as ordinary kind ascriptions, such as in the pseudo-ML interface:

```
signature SIG =
   sig
     typeconstructor s : S(int)
     typeconstructor t : Type
     typeconstructor u : S(s -> t)
     ...
   end
```

In type-theoretic notation, `SIG` is rendered:

$$\Sigma s{:}\mathsf{S}(\texttt{int}).\,\Sigma t{:}\mathsf{T}.\,\Sigma u{:}\mathsf{S}(s \to t)\ldots$$

ML modules give rise not only to singleton kinds, but also to dependent kinds. As the example illustrates, dependent sums arise naturally from type constructor fields (or substructures, which are not shown in the example) that are referenced later. In ML dialects with applicative functors (particularly O'Caml [9] and Moscow ML [12]), dependent products arise naturally from functors. Dialects with only generative functors (particularly Standard ML [10]) do not use the full generality of dependent products. Even so, there does not appear to be any particularly compelling restriction of the singleton calculus that is still sufficient for such dialects.

Singleton and dependent kinds add interesting complications to the equational theory of types [13, 1]. The issue is, in the presence of singleton kinds, type equivalence is context-sensitive.

For example, the two constructors $\lambda\alpha{:}\mathsf{T}.\alpha$ and $\lambda\alpha{:}\mathsf{T}.\texttt{int}$ can be given either the kind $\mathsf{T} \to \mathsf{T}$ or (using subkinding $\mathsf{S}(\texttt{int}) \leq \mathsf{T}$ and the usual contravariance on for arrow kinds) $\mathsf{S}(\texttt{int}) \to \mathsf{T}$. When compared at the first kind, the two are inequal. However, when compared at the second kind, the two are equal, since α is guaranteed to be `int`.

There are (at least) two practical reasons why the equational theory of types is important:

- Most obviously, in order to perform type checking, we must have an effective algorithm to determine whether two types are equivalent.
- Just as importantly, in order to show that the language is type-safe, we must show that the equational theory of types is *consistent,* in the sense that two types with different outermost type operators are never equivalent. For example, if $\texttt{int} = \texttt{int} \to \texttt{int}$, then the non-function application 2(3) is well-typed, and the language is not type-safe.

Since type equivalence is context sensitive, the usual methods for determining equivalence and proving consistency using confluent reduction relations do not apply.

*The material is based on work supported in part by NSF grant 0716469 and by a grant from CyLab. Any opinions, findings, and conclusions of recommendations in this publication are those of the author and do not reflect the views of these agencies.

Stone and Harper [14] give an appropriate algorithm for type equivalence in the presence of singleton kinds and prove it sound and complete. Their completeness proof works by constructing a model based on a logical relation that coincides with the algorithm at the base kind (*i.e.,* T). They then show that constructors that are equivalent in the model are judged equivalent by the algorithm (this is non-trivial at higher kinds), and that all the type system's rules are valid in the model. Consistency can then be obtained as a corollary, by inspection of the algorithm.

For many purposes, Stone and Harper's result is canonical. Nevertheless, there are circumstances in which we prefer a direct, syntactic proof to one that employs a logical relation, in order to minimize the proof-theoretic strength and mathematical foundation on which the proof relies. The context of this work provides such a circumstance.

The motivation for this work is the machine-checked proof of type safety of Standard ML [8]. Consistency of type equivalence is essential to that proof (as noted above), but we also require a proof that can be formalized in the Twelf meta-logic [11]. Twelf is limited to proving sentences that can be expressed as Π-2 sentences (that is, sentences of the form $\forall \mathit{input\text{-}objects}.\exists \mathit{output\text{-}objects}.\mathsf{true}$). This limitation excludes logical relations proofs (such as Stone and Harper's), which require unlimited quantifier alternations.

In this paper, we give a syntactic proof of the metatheoretic results (decidable type equivalence and consistency) in the presence of singleton kinds. The proof is fully formalized in Twelf, and available on-line at:

`www.cs.cmu.edu/~crary/papers/2009/synsing.tgz`

The proof is based on cut elimination, using a device called *hereditary substitution* devised by Kevin Watkins [15]. In it, we refer to the singleton-kind language as the *external language* (EL), and we formulate an alternative presentation of the language that we call the *internal language* (IL).[1]

The internal language is a *canonical presentation* of the singleton-kind calculus, meaning that it has the important property that equivalent types can be written in only one way (up to alpha-equivalence). We give a sound, complete, and effective translation from the EL into the IL. Consequently, we can compare EL types for equivalence by translating them into the IL and comparing them syntactically. Consistency follows by inspection of the translation.

The canonical presentation is adapted from work on canonical presentations of logical frameworks [4, 15, 6]. It works by requiring that all type constructors be written in an eta-long[2] canonical form. Early efforts at canonical presentations [4] were complicated by the fact that canonical forms (indeed, beta-normal forms in general) are not preserved under substitution. Watkins, *et al.* [15] addresses the problem by devising an alternative notion of substitution, called hereditary substitution, that contracts redices arising from substitution. The termination of hereditary substitution is shown using an argument similar to cut elimination.

The broad structure of the proof is as follows: First we establish that the standard properties of substitution apply to hereditary substitution as well. (This is a substantial portion of the proof, but we will deal with it briefly in this paper.) Next we define subtyping (which is not primitive in the IL) and some related notions and establish their properties. We those preliminaries done, we are ready to define the EL-to-IL translation. We must show that translation commutes with substitution (in an appropriate sense), and then we are ready to prove that the translation is complete. Since our algorithm simply translates terms into the IL and compares them, it follows that the algorithm is complete.

To show soundness, we define an IL-to-EL relation called transliteration, and show that it inverts the EL-to-IL translation in an appropriate sense. Therefore, when two EL terms translate to the same IL term, that IL term transliterates to an EL term that is equivalent to both original EL terms, which are consequently equivalent to each other. It follows that the algorithm is sound.

Note that the ordinary terms of the programming language (as opposed to the types and kinds) play no role in its equational theory, and therefore no role in the development to come. Therefore, we will neglect terms in what follows. Furthermore, with terms taken out of the picture, we will simplify the terminology by lowering each syntactic level by one. Hence, types and type constructors will be referred to as "terms" and kinds will be referred to as "types." Accordingly, we will refer to *singleton types,* rather than singleton kinds.

In the remainder of the paper we lay out the details of the proof. In Section 2 we summarize the conventional singleton type calculus (that is, the EL). In Section 3 we present the canonical presentation (that is, the IL). In Section 4 we discuss the role of subtyping in the IL. In Section 5 we give the translation from the EL to IL and summarize its correctness proof. Concluding remarks appear in Section 6.

2 The Singleton Type Calculus

The syntax of the singleton type calculus is as follows:

$$\begin{array}{llll} \text{types} & \bar{A}, \bar{B} & ::= & \mathsf{T} \mid \mathsf{S}(\bar{M}) \mid \Pi x{:}\bar{A}.\bar{B} \mid \Sigma x{:}\bar{A}.\bar{B} \\ \text{terms} & \bar{M}, \bar{N} & ::= & c \mid x \mid \lambda x{:}\bar{A}.\bar{M} \mid \bar{M}\bar{N} \mid \langle \bar{M}, \bar{N} \rangle \\ & & \mid & \pi_1 \bar{M} \mid \pi_2 \bar{M} \\ \text{contexts} & \bar{\Gamma} & ::= & \epsilon \mid \bar{\Gamma}, x{:}\bar{A} \end{array}$$

We write all EL metavariables with an overbar, to distinguish them from the IL metavariables that come later. When necessary to avoid confusion, we will distinguish EL and IL judgements by writing them with marked turnstiles ($\vdash_E$, $\vdash_I$).

The sole base type is written T. (When lifted a level, this is the kind of types.) The singleton type is written $\mathsf{S}(\bar{M})$, for $\bar{M} : \mathsf{T}$. Dependent products and sums are written in the usual way. When x does not appear free in $\bar{B}$, we will often write $\Pi x{:}\bar{A}.\bar{B}$ as $\bar{A} \to \bar{B}$, and write $\Sigma x{:}\bar{A}.\bar{B}$ as $\bar{A} \times \bar{B}$.

The term-level forms are standard, where c ranges over a fixed collection of constants. As usual, we view alpha-equivalent types and terms as identical. We adopt the convention that a context is syntactically well-formed only when each variable it binds is distinct. We say that a type is *simple* when it contains no singleton types (equivalently, when it can be written using T, $\to$, and $\times$).

Although singleton types are supported primitively only at base type, we can construct higher-order singleton types using dependent types, as shown in Figure 1. For example, the singleton type for $\bar{M} : \mathsf{T} \to \mathsf{T}$ is the dependent

[1] To avoid confusion, note that the external language in this work corresponds to the internal language in Lee, et al. [8].

[2] That is: beta-normal, and eta-expanded as much as possible while remaining beta-normal.

$$
\begin{array}{lcll}
\mathsf{S}_{\mathsf{T}}(\bar{M}) & \stackrel{\text{def}}{=} & \mathsf{S}(\bar{M}) & \\
\mathsf{S}_{\mathsf{S}(\bar{N})}(\bar{M}) & \stackrel{\text{def}}{=} & \mathsf{S}(\bar{M}) & \\
\mathsf{S}_{\Pi x{:}\bar{A}.\bar{B}}(\bar{M}) & \stackrel{\text{def}}{=} & \Pi x{:}\bar{A}.\mathsf{S}_{\bar{B}}(\bar{M}x) & (x \text{ not free in } \bar{M}) \\
\mathsf{S}_{\Sigma x{:}\bar{A}.\bar{B}}(\bar{M}) & \stackrel{\text{def}}{=} & \mathsf{S}_{\bar{A}}(\pi_1\bar{M}) \times \mathsf{S}_{[\pi_1\bar{M}/x]\bar{B}}(\pi_2\bar{M}) &
\end{array}
$$

Figure 1: Higher-Order Singletons

$$
\frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M})}\ (1) \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \bar{M} : \mathsf{S}(\bar{M})}\ (2)
$$

$$
\frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{S}(\bar{M})}\ (3) \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M}) \equiv \mathsf{S}(\bar{N})}\ (4)
$$

$$
\frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{S}(\bar{N})}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}}\ (5) \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M}) \leq \mathsf{T}}\ (6)
$$

Figure 2: Singleton Rules

type $\Pi x{:}T.\mathsf{S}(\bar{M}x)$, which accepts elements of T and returns whatever $\bar{M}$ does on each argument.

The typing rules are given by five judgements: typing ($\bar{\Gamma} \vdash \bar{M} : \bar{A}$), term equivalence ($\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : A$), type formation ($\bar{\Gamma} \vdash \bar{A}$), type equivalence ($\bar{\Gamma} \vdash \bar{A} \equiv \bar{B}$), and subtyping ($\bar{\Gamma} \vdash \bar{A} \leq \bar{B}$).

The complete type system is given in Appendix A. Most of the rules are standard for a dependently typed language with subtyping. The types of constants are given by a *basis,*[3] which is a fixed mapping from constants to simple types, such as $\{\texttt{int} \mapsto \mathsf{T}, \texttt{arrow} \mapsto (\mathsf{T} \to \mathsf{T} \to \mathsf{T}), \texttt{list} \mapsto (\mathsf{T} \to \mathsf{T})\}$.

The rules for singleton types are repeated in Figure 2. Rule 1 is the formation rule for singleton types. Rules 2 and 3 are the introduction rules for singleton types. Rule 4 states that singleton types over equivalent terms are equivalent. Among other purposes, this allows the derivation of $\bar{M} : \mathsf{S}(\bar{N})$ when $\bar{M} \equiv \bar{N} : \mathsf{T}$. Rule 5 is the singleton elimination rule; it allows one to derive $\bar{M} \equiv \bar{N} : \mathsf{T}$ from $\bar{M} : \mathsf{S}(\bar{N})$. Rule 6 allows one to promote singletons to T, thereby forgetting equivalence information.

In order to make good use of singleton types, we must also include the four extensionality principles given in Figure 3. Rules 7 and 8 allow us to classify a term according to its extensional behavior (or equivalently, according to its eta-expansion). For example, assuming the sample basis given above, consider the constant `list`. With the singleton rules alone, we cannot obtain any better type than $\mathsf{T} \to \mathsf{T}$. However, using rule 7, we can retype `list` as belonging to $\mathsf{S}_{\mathsf{T}\to\mathsf{T}}(\texttt{list})$:

$$
\dfrac{\dfrac{}{\Gamma \vdash \texttt{list} : \mathsf{T} \to \mathsf{T}} \quad \dfrac{\dfrac{\vdots}{\Gamma, x{:}\mathsf{T} \vdash \texttt{list}\, x : \mathsf{T}}}{\Gamma, x{:}\mathsf{T} \vdash \texttt{list}\, x : \mathsf{S}(\texttt{list}\, x)}\ (2)}{\Gamma \vdash \texttt{list} : \Pi x{:}\mathsf{T}.\mathsf{S}(\texttt{list}\, x)}\ (7)
$$

[3]The basis mapping plays the same role as an LF signature [5], but we do not employ that term in order to avoid confusion with ML signatures.

$$
\frac{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M}x : \bar{B}}{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}}\ (7)
$$

$$
\frac{\bar{\Gamma} \vdash \pi_1\bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \pi_2\bar{M} : [\pi_1\bar{M}/x]\bar{B} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \bar{M} : \Sigma x{:}\bar{A}.\bar{B}}\ (8)
$$

$$
\frac{\begin{array}{c}\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}' \qquad \bar{\Gamma} \vdash \bar{N} : \Pi x{:}\bar{A}.\bar{B}'' \\ \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M}x \equiv \bar{N}x : \bar{B}\end{array}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Pi x{:}\bar{A}.\bar{B}}\ (9)
$$

$$
\frac{\begin{array}{c}\bar{\Gamma} \vdash \pi_1\bar{M} \equiv \pi_1\bar{N} : \bar{A} \\ \bar{\Gamma} \vdash \pi_2\bar{M} \equiv \pi_2\bar{N} : [\pi_1\bar{M}/x]\bar{B} \qquad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}\end{array}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Sigma x{:}\bar{A}.\bar{B}}\ (10)
$$

Figure 3: Extensionality Rules

Rule 8 is used similarly. For technical reasons, rule 7 (and rule 9, below) require an independent proof that the term(s) belong to a compatible function type.

Rules 9 and 10 allow us to derive equivalences of terms based on their extensional behavior. Consider the example terms from the introduction, $\lambda x{:}\mathsf{T}.x$ and $\lambda x{:}\mathsf{T}.\texttt{int}$. Without extensionality principles we cannot show that these terms are equal, since $x \not\equiv \texttt{int}$ under the assumption $x{:}\mathsf{T}$. However, using rule 9 we can derive:

$$
\dfrac{\dfrac{\dfrac{}{x{:}\mathsf{S}(\texttt{int}) \vdash x : \mathsf{S}(\texttt{int})}}{x{:}\mathsf{S}(\texttt{int}) \vdash x \equiv \texttt{int} : \mathsf{T}}\ (5)}{\vdots}}{\ldots \textit{elided} \ldots \quad x{:}\mathsf{S}(\texttt{int}) \vdash (\lambda x{:}\mathsf{T}.x)x \equiv (\lambda x{:}\mathsf{T}.\texttt{int})x : \mathsf{T}}{\vdash \lambda x{:}\mathsf{T}.x \equiv \lambda x{:}\mathsf{T}.\texttt{int} : \mathsf{S}(\texttt{int}) \to \mathsf{T}}\ (9)
$$

The extensionality rules are the principal source of complexity in the singleton calculus, because they allow us to determine term equivalence or inequivalence based on the classifying type. A more thorough discussion of the singleton type calculus is given in Stone and Harper [14] and in Crary [2].

3 The Canonical Presentation

The central idea to the canonical presentation is to arrange that equivalent terms can be written in only one way, up to alpha-equivalence [15]. The first step to doing so is to institute a syntactic division between atoms and terms:[4]

$$
\begin{array}{llll}
\text{types} & A, B & ::= & \mathsf{T} \mid \mathsf{S}(R) \mid \Pi x{:}A.B \mid \Sigma x{:}A.B \\
\text{atoms} & R, S & ::= & c \mid x \mid RM \mid \pi_1 R \mid \pi_2 R \\
\text{terms} & M, N & ::= & \mathsf{at}(R) \mid \lambda x.M \mid \langle M, N\rangle \\
\text{contexts} & \Gamma & ::= & \epsilon \mid \Gamma, x{:}A
\end{array}
$$

This syntactic structure ensures that only beta-normal forms can be written. Note that since singletons exist only at base type, and since the only facilities offered by terms and not atoms (*i.e.,* lambdas and pairs) are not useful at base type, it is appropriate to limit singleton types to atoms.

[4]As in the EL, we view alpha-equivalent types, atoms, and terms as identical, and adopt the convention that a context is syntactically well-formed only when each variable it binds is distinct.

$\boxed{\Gamma \vdash A}$

$$\frac{}{\Gamma \vdash \mathsf{T}} \qquad \frac{\Gamma \vdash R \Rightarrow \mathsf{T}}{\Gamma \vdash \mathsf{S}(R)} \qquad \frac{\Gamma \vdash A \quad \Gamma, x{:}A \vdash B}{\Gamma \vdash \Pi x{:}A.B} \qquad \frac{\Gamma \vdash A \quad \Gamma, x{:}A \vdash B}{\Gamma \vdash \Sigma x{:}A.B}$$

$\boxed{\Gamma \vdash R \Rightarrow A}$

$$\frac{\mathsf{Basis}(c) = A}{\Gamma \vdash c \Rightarrow A} \qquad \frac{\Gamma(x) = A}{\Gamma \vdash x \Rightarrow A} \qquad \frac{\Gamma \vdash R \Rightarrow \Pi x{:}A.B \quad \Gamma \vdash M \Leftarrow A \quad \Gamma \vdash [M/x]B}{\Gamma \vdash RM \Rightarrow [M/x]B}$$

$$\frac{\Gamma \vdash R \Rightarrow \Sigma x{:}A.B}{\Gamma \vdash \pi_1 R \Rightarrow A} \qquad \frac{\Gamma \vdash R \Rightarrow \Sigma x{:}A.B}{\Gamma \vdash \pi_2 R \Rightarrow \lceil \pi_1 R/x \rceil B}$$

$\boxed{\Gamma \vdash M \Leftarrow A}$

$$\frac{\Gamma \vdash R \Rightarrow \mathsf{T}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{T}} \qquad \frac{\Gamma \vdash R \Rightarrow \mathsf{T}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{S}(R)} \qquad \frac{\Gamma \vdash A \quad \Gamma, x{:}A \vdash M \Leftarrow B}{\Gamma \vdash \lambda x.M \Leftarrow \Pi x{:}A.B} \qquad \frac{\Gamma \vdash M \Leftarrow A \quad \Gamma \vdash N \Leftarrow [M/x]B \quad \Gamma, x{:}A \vdash B}{\Gamma \vdash \langle M, N \rangle \Leftarrow \Sigma x{:}A.B}$$

Figure 4: IL Rules

Terms and atoms are type-checked using a bidirectional system that synthesizes types for atoms (written $\Gamma \vdash R \Rightarrow A$) and checks types for terms (written $\Gamma \vdash M \Leftarrow A$).[5] No equivalence rules are employed.

To ensure that terms are not only beta-normal but eta-long, the type system provides that the shift from atoms to terms is permitted only at base type:

$$\frac{\Gamma \vdash R \Rightarrow \mathsf{T}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{T}} \qquad \frac{\Gamma \vdash R \Rightarrow \mathsf{T}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{S}(R)}$$

Note that these shift rules *do not* permit shifting from atoms to terms at singleton types. This point is critical, because it implies that singleton assumptions cannot be used to form terms. Thus, we are not faced with the problem of deciding, for example, whether the terms x and `int` are equal under the assumption $x{:}\mathsf{S}(\mathtt{int})$, because the former term is ill-typed. Singleton assumptions can be used to form atoms; we make no commitment that atoms are represented uniquely.

The complete set of typing rules is given in Figure 4. By an abuse of notation, we treat the EL basis as an IL basis as well. This is permissible since simple types are isomorphic in the EL and IL.

Hereditary substitution The typing rules employ two notions of substitution: atom substitution (written $\lceil R/x \rceil$) and term substitution (written $[M/x]$). Since variables are atoms, atom substitution is unproblematic and is defined in the usual (capture-avoiding) way. On the other hand, since variables are not terms, a naive definition of term substitution would produce expressions that are syntactically ill-formed. Put another way, our syntax only allows the expression of beta-normal expressions, but beta-normal expressions are not closed under substitution.

We therefore require a special definition for term substitution, called *hereditary substitution* [15]. Hereditary substitution automatically contracts any redices that arise in substitution, which can—in the case of functions—induce a secondary substitution. (Note that this is a stronger notion than finite developments [7], which does not contract secondary redices created by reduction.)

Hereditary substitution is defined in Figure 5 as a collection of judgements. The key observation is that substitution into an atom can return an atom or a term, depending on whether the substitution variable is the atom's head variable. If not, the atom's overall structure is preserved; substitution only intrudes into the terms along its spine. But if so, redices are often created and must be reduced.

Lemma 3.1 justifies the use of term substitution into terms or types (but not atoms) as a function:

Lemma 3.1 (Hereditary substitution is a function)

1. *Hereditary substitution is uniquely defined, where it is defined at all.*
2. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow B$. *Then either there exists* R' *such that* $[M/x]R$ is R', *or* N' *such that* $[M/x]R$ is N'. *Moreover, the definition constitutes an effective procedure for computing* R' *or* N'.
3. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash N \Leftarrow B$. *Then there exists* N' *such that* $[M/x]N$ is N'. *Moreover, the definition constitutes an effective procedure for computing* N'.
4. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash B$. *Then there exists* B' *such that* $[M/x]B$ is B'. *Moreover, the definition constitutes an effective procedure for computing* B'.

Proof Sketch

Part 1 is easy to show, since a variable cannot simultaneously be an atom's head variable and not. Parts 2 and 3 are proven in a similar fashion to the classic proof of cut elimination. For any type A, let A^* be the simple type obtained by replacing singletons by T. Then parts 2 and 3 are proven simultaneously by induction on the cut type A^*, with an inner induction on R and N. In the important case:

$$\frac{[M/x]R \text{ is } \lambda y.O \quad [M/x]N \text{ is } N' \quad [N'/y]O \text{ is } O'}{[M/x]RN \text{ is } O'}$$

[5] The intuition behind this notation is the types of atoms are uniquely determined, and thus can be viewed as an output of type checking, but the types of terms are not uniquely determined, and thus must be viewed as an input.

$\boxed{[M/x]R \text{ is } R'}$

$$\frac{x \neq y}{[M/x]y \text{ is } y} \qquad \frac{[M/x]R \text{ is } R' \quad [M/x]N \text{ is } N'}{[M/x]RN \text{ is } R'N'} \qquad \frac{[M/x]R \text{ is } R'}{[M/x]\pi_1 R \text{ is } \pi_1 R'} \qquad \frac{[M/x]R \text{ is } R'}{[M/x]\pi_2 R \text{ is } \pi_2 R'}$$

$\boxed{[M/x]R \text{ is } N}$

$$\frac{}{[M/x]x \text{ is } M} \qquad \frac{[M/x]R \text{ is } \lambda y.O \quad [M/x]N \text{ is } N' \quad [N'/y]O \text{ is } O'}{[M/x]RN \text{ is } O'} \qquad \frac{[M/x]R \text{ is } \langle N, O\rangle}{[M/x]\pi_1 R \text{ is } N} \qquad \frac{[M/x]R \text{ is } \langle N, O\rangle}{[M/x]\pi_2 R \text{ is } O}$$

$\boxed{[M/x]N \text{ is } N'}$

$$\frac{[M/x]R \text{ is } R'}{[M/x]\mathsf{at}(R) \text{ is } \mathsf{at}(R')} \qquad \frac{[M/x]R \text{ is } N}{[M/x]\mathsf{at}(R) \text{ is } N} \qquad \frac{x \neq y \quad y \text{ not free in } M \quad [M/x]N \text{ is } N'}{[M/x]\lambda y.N \text{ is } \lambda y.N'} \qquad \frac{[M/x]N \text{ is } N' \quad [M/x]O \text{ is } O'}{[M/x]\langle N, O\rangle \text{ is } \langle N', O'\rangle}$$

$\boxed{[M/x]A \text{ is } A'}$

$$\frac{}{[M/x]\mathsf{T} \text{ is } \mathsf{T}} \qquad \frac{[M/x]R \text{ is } R'}{[M/x]\mathsf{S}(R) \text{ is } \mathsf{S}(R')} \qquad \frac{[M/x]R \text{ is } \mathsf{at}(R')}{[M/x]\mathsf{S}(R) \text{ is } \mathsf{S}(R')}$$

$$\frac{x \neq y \quad y \text{ not free in } M \quad [M/x]A \text{ is } A' \quad [M/x]B \text{ is } B'}{[M/x]\Pi y{:}A.B \text{ is } \Pi y{:}A'.B'} \qquad \frac{x \neq y \quad y \text{ not free in } M \quad [M/x]A \text{ is } A' \quad [M/x]B \text{ is } B'}{[M/x]\Sigma y{:}A.B \text{ is } \Sigma y{:}A'.B'}$$

Figure 5: Hereditary Substitution

we are guaranteed that the cut type for $[N'/y]O$ (that is, the type of y) is smaller than A^* because x must be the head variable of R, and thus the simple type of $\lambda y.O$ (which is the same as the simple type of R) is either A^* or smaller. Once parts 2 and 3 are established, part 4 is proven by induction on B.

Parts 2 and 4 also rely on a lemma stating that substitution preserves simple types (the full substitution lemma—Lemma 3.2—is proven subsequently). That lemma is enough to dictate, where required, the coarse structure of terms returned by substitution (as in the above rule's premise $[M/x]R \text{ is } \lambda y.O$).

Once functionality is established, we can show that substitution preserves types:

Lemma 3.2 (Substitution)

1. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow B$, *and* $[M/x]R \text{ is } R'$. *Then* $\Gamma, [M/x]\Gamma' \vdash R \Rightarrow [M/x]B$.
2. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow B$, *and* $[M/x]R \text{ is } N$. *Then* $\Gamma, [M/x]\Gamma' \vdash N \Leftarrow [M/x]B$.
3. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash N \Leftarrow B$. *Then* $\Gamma, [M/x]\Gamma' \vdash [M/x]N \Leftarrow [M/x]B$.
4. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash B$. *Then* $\Gamma, [M/x]\Gamma' \vdash [M/x]B$.

Proof Sketch

By induction on derivations, using a collection of straightforward lemmas showing that substitutions can be permuted. Since there are several substitution judgements, each of which can interact with the others and with atomic substitution, there are fourteen such lemmas, making this a large proof.

Expansion One important remaining concept in the IL is eta-expansion, which expresses how to expand an atom into an "equivalent" term. Eta-expansion (written $\mathsf{E}(R : A)$) is defined as follows:

$$\begin{array}{lcl}
\mathsf{E}(R : \mathsf{T}) & \stackrel{\text{def}}{=} & \mathsf{at}(R) \\
\mathsf{E}(R : \mathsf{S}(R')) & \stackrel{\text{def}}{=} & \mathsf{at}(R') \\
\mathsf{E}(R : \Pi x{:}A.B) & \stackrel{\text{def}}{=} & \lambda x.\mathsf{E}(R\mathsf{E}(x : A) : B) \\
\mathsf{E}(R : \Sigma x{:}A.B) & \stackrel{\text{def}}{=} & \langle \mathsf{E}(\pi_1 R : A), \mathsf{E}(\pi_2 R : \lceil \pi_1 R/x \rceil B)\rangle
\end{array}$$

Note that eta-expansion at a singleton type discards the atom being expanded, and replaces it with the singleton's specification.

An eta-expanded atom is equivalent to the original term in the sense that eta-expansion commutes with substitution in various ways:

Lemma 3.3 (Expansion)

1. *Suppose* $\Gamma \vdash R \Rightarrow A$. *Then* $\Gamma \vdash \mathsf{E}(R : A) \Leftarrow A$.
2. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow B$ *and* $[M/x]R \text{ is } R'$. *Then* $[M/x]\mathsf{E}(R : B) = \mathsf{E}(R' : [M/x]B)$.
3. *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash R \Rightarrow B$ *and* $[M/x]R \text{ is } N$. *Then* $[M/x]\mathsf{E}(R : B) = N$.
4. *Suppose* $\Gamma \vdash R \Rightarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash S \Rightarrow B$ *and* $[\mathsf{E}(R : A)/x]S \text{ is } S'$. *Then* $S' = \lceil R/x \rceil S$.
5. *Suppose* $\Gamma \vdash R \Rightarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash S \Rightarrow B$ *and* $[\mathsf{E}(R : A)/x]S \text{ is } N$. *Then* $N = \mathsf{E}(\lceil R/x \rceil S : \lceil R/x \rceil B)$.
6. *Suppose* $\Gamma \vdash R \Rightarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash M \Leftarrow B$. *Then* $[\mathsf{E}(R : A)/x]M = \lceil R/x \rceil M$.

7. *Suppose* $\Gamma \vdash R \Rightarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash B$. *Then* $[\mathsf{E}(R : A)/x]B = \lceil R/x \rceil B$.

Proof Sketch

By simultaneous induction on derivations.

Part 1 states that expansion preserves types. Parts 2 and 3 relate to substitution *into* an eta-expansion, while parts 4 through 7 relate to substitution *using* an eta-expansion. Parts 3, 6, and 7 have an important corollary obtained by choosing $R = x$:

Corollary 3.4 (Variable expansion)

1. *Suppose* $\Gamma \vdash M : A$ *and* x *is fresh. Then* $[M/x]\mathsf{E}(x : A) = M$.
2. *Suppose* $\Gamma \vdash A$ *and* $\Gamma, x{:}A \vdash M \Leftarrow B$. *Then* $[\mathsf{E}(x : A)/x]M = M$.
3. *Suppose* $\Gamma \vdash A$ *and* $\Gamma, x{:}A \vdash B$. *Then* $[\mathsf{E}(x : A)/x]B = B$.

4 Subtyping

Subtyping in the EL is associated with a subsumption rule, as is typical for type systems with subtyping. However, the IL cannot support a subsumption rule. For example, the term $\lambda x.\mathsf{at}(x)$ belongs to the type $\mathsf{T} \to \mathsf{T}$ but it does not belong to $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$, despite that fact that $\mathsf{T} \to \mathsf{T}$ is a subtype of $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$.

Instead, subtyping in the IL is associated with a coercion from the subtype to the supertype. For example, the term $\lambda x.\mathsf{at}(x)$ when coerced from $\mathsf{T} \to \mathsf{T}$ to $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$ becomes $\lambda x.\mathsf{at}(\mathtt{int})$.

The subtyping coercion expresses the phenomenon that expresses itself in the EL as context sensitivity of equivalence. At the type $\mathsf{T} \to \mathsf{T}$ the terms $\lambda x.\mathsf{at}(x)$ and $\lambda x.\mathsf{at}(\mathtt{int})$ are distinct, but when coerced to $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$, they both become $\lambda x.\mathsf{at}(\mathtt{int})$.

The subtyping coercion judgement is written $A \leq B \rightsquigarrow x.M$ to mean that A is a subtype of B with coercion $x.M$. When N belongs to A, it is coerced to belong to B by (hereditarily) substituting N for x in M.

$$\frac{}{\mathsf{T} \leq \mathsf{T} \rightsquigarrow x.\mathsf{at}(x)}$$

$$\frac{x \text{ not free in } R}{\mathsf{S}(R) \leq \mathsf{T} \rightsquigarrow x.\mathsf{at}(R)} \qquad \frac{x \text{ not free in } R}{\mathsf{S}(R) \leq \mathsf{S}(R) \rightsquigarrow x.\mathsf{at}(R)}$$

$$\frac{A' \leq A \rightsquigarrow x'.M \quad [M/x]B \leq B' \rightsquigarrow y.N \quad z \text{ not free in } M, N}{\Pi x{:}A.B \leq \Pi x'{:}A'.B' \rightsquigarrow z.\,(\lambda x'.\lceil zM/y \rceil N)}$$

$$\frac{A \leq A' \rightsquigarrow x.M \quad B \leq [M/x']B' \rightsquigarrow y.N \quad z \text{ not free in } M, N}{\Sigma x{:}A.B \leq \Sigma x'{:}A'.B' \rightsquigarrow z.\,\langle \lceil \pi_1 z/x \rceil M, \lceil \pi_1 z, \pi_2 z/x, y \rceil N \rangle}$$

Subtyping preserves types, commutes with substitution, and is reflexive and transitive:

Lemma 4.1 (Subtyping)

1. *Suppose* $A \leq B \rightsquigarrow x.M$, *where* $\Gamma \vdash A, B$ *and* x *is not bound by* Γ. *Then* $\Gamma, x{:}A \vdash M \Leftarrow B$.
2. *Suppose* $A \leq B \rightsquigarrow y.N$, *where* $\Gamma \vdash M \Leftarrow C$ *and* $\Gamma, x{:}C, \Gamma' \vdash A, B$ *and* y *is not free in* M. *Then* $[M/x]A \leq [M/x]B \rightsquigarrow y.[M/x]N$.
3. *Suppose* $\Gamma \vdash A$, *where* x *is not free in* A. *Then* $A \leq A \rightsquigarrow x.\mathsf{E}(x : A)$.
4. *Suppose* $A \leq B \rightsquigarrow x.M$ *and* $B \leq C \rightsquigarrow y.N$, *where* $\Gamma \vdash A, B, C$ *and* x *is not free in* N. *Then* $A \leq C \rightsquigarrow x.[M/y]N$.

Transparency We say that a type is *transparent* if it is a singleton, or a Π with a transparent codomain, or a $\times$ with both constituents transparent. Transparent types are so named because they fully specify a term. That is, well-formed transparent types are inhabited by exactly one term. In Section 5 we will exploit this property to use transparent types as stand-ins for terms.

Lemma 4.2 (Transparency) *Suppose* A *is transparent and* $\Gamma \vdash M, N \Leftarrow A$. *Then* $M = N$.

Transparent types have an interesting property when used as a subtype. Suppose A is transparent and $A \leq B \rightsquigarrow x.M$. Since A fully specifies a term, the coercion M can be expressed without reference to x. Indeed, since terms can be expressed in only one way in the canonical system, M *must be* expressed without reference to x.

Lemma 4.3 (Transparent coercion) *Suppose* A *is transparent and* $A \leq B \rightsquigarrow x.M$. *Then* x *is not free in* M.

In this situation we will omit writing the unused binding occurrence, that is $A \leq B \rightsquigarrow M$.

Selfification Given a term M belonging to A, there exists a transparent subtype of A containing exactly the term M, written $\mathsf{Self}(M : A)$. This type, called the *selfification* of M at A, is reminiscent of the higher-order singletons from Section 2:

$$\begin{array}{lcl}
\mathsf{Self}(\mathsf{at}(R) : \mathsf{T}) & \stackrel{\text{def}}{=} & \mathsf{S}(R) \\
\mathsf{Self}(\mathsf{at}(R) : \mathsf{S}(R)) & \stackrel{\text{def}}{=} & \mathsf{S}(R) \\
\mathsf{Self}(\lambda x.M : \Pi x{:}A.B) & \stackrel{\text{def}}{=} & \Pi x{:}A.\mathsf{Self}(M : B) \\
\mathsf{Self}(\langle M, N \rangle : \Sigma x{:}A.B) & \stackrel{\text{def}}{=} & \mathsf{Self}(M : A) \\
 & & \quad \times \mathsf{Self}(N : [M/x]B)
\end{array}$$

The following lemmas state several important properties of selfification:

Lemma 4.4 (Selfification) *Suppose* $\Gamma \vdash M \Leftarrow A$. *Then* $\mathsf{Self}(M : A)$ *is defined, and:*

1. $\Gamma \vdash \mathsf{Self}(M : A)$, *and*
2. $\Gamma \vdash M \Leftarrow \mathsf{Self}(M : A)$, *and*
3. $\mathsf{Self}(M : A)$ *is transparent, and*
4. $\mathsf{Self}(M : A) \leq A \rightsquigarrow M$, *and*
5. *If* A *is transparent then* $\mathsf{Self}(M : A) = A$.

Lemma 4.5 (Self substitution) *Suppose* $\Gamma \vdash M \Leftarrow A$ *and* $\Gamma, x{:}A, \Gamma' \vdash N \Leftarrow B$. *Then* $[M/x]\mathsf{Self}(N : B) = \mathsf{Self}([M/x]N : [M/x]B)$.

Lemma 4.6 (Subtype strengthening) *Suppose* A *is transparent and* $A \leq B \rightsquigarrow M$, *where* $\Gamma \vdash A, B$. *Then* $A \leq \mathsf{Self}(M : B) \rightsquigarrow M$.

$\boxed{\Gamma \vdash \bar{A} \searrow A}$

$$\frac{}{\Gamma \vdash \mathsf{T} \searrow \mathsf{T}} \qquad \frac{\Gamma \vdash \bar{M} \searrow \mathsf{S}(R)}{\Gamma \vdash \mathsf{S}(\bar{M}) \searrow \mathsf{S}(R)} \qquad \frac{\Gamma \vdash \bar{A} \searrow A \quad \Gamma, x{:}A \vdash \bar{B} \searrow B}{\Gamma \vdash \Pi x{:}\bar{A}.\bar{B} \searrow \Pi x{:}A.B} \qquad \frac{\Gamma \vdash \bar{A} \searrow A \quad \Gamma, x{:}A \vdash \bar{B} \searrow B}{\Gamma \vdash \Sigma x{:}\bar{A}.\bar{B} \searrow \Sigma x{:}A.B}$$

$\boxed{\Gamma \vdash \bar{M} \searrow A}$

$$\frac{\mathsf{Basis}(c) = A}{\Gamma \vdash c \searrow \mathsf{Self}(\mathsf{E}(c : A) : A)} \qquad \frac{\Gamma(x) = A}{\Gamma \vdash x \searrow \mathsf{Self}(\mathsf{E}(x : A) : A)} \qquad \frac{\Gamma \vdash \bar{M} \searrow \Pi x{:}A.B \quad \Gamma \vdash \bar{N} \searrow C \quad C \leq A \rightsquigarrow N}{\Gamma \vdash \bar{M}\bar{N} \searrow [N/x]B}$$

$$\frac{\Gamma \vdash \bar{A} \searrow A \quad \Gamma, x{:}A \vdash \bar{M} \searrow B}{\Gamma \vdash \lambda x{:}\bar{A}.\bar{M} \searrow \Pi x{:}A.B} \qquad \frac{\Gamma \vdash \bar{M} \searrow A \times B}{\Gamma \vdash \pi_1 \bar{M} \searrow A} \qquad \frac{\Gamma \vdash \bar{M} \searrow A \times B}{\Gamma \vdash \pi_2 \bar{M} \searrow B} \qquad \frac{\Gamma \vdash \bar{M} \searrow A \quad \Gamma \vdash \bar{N} \searrow B}{\Gamma \vdash \langle \bar{M}, \bar{N} \rangle \searrow A \times B}$$

Figure 6: The Translation

5 The Translation

The translation takes EL types to IL types, and also takes EL terms to IL types (not terms!). The latter aspect, which may be surprising, stems from the absence of a subsumption rule in the IL. Were the term translation to return a term, it would have to return its type as well, in order to properly mediate subtyping in function application.

To see why, suppose the translation naively employed a term-to-term judgement $\bar{M} \searrow M$ and a type-free rule for application:

$$\frac{\bar{M} \searrow \lambda x.M \quad \bar{N} \searrow N}{\bar{M}\bar{N} \searrow [N/x]M} \ (wrong)$$

Then suppose we translated $\bar{F}(\lambda x{:}\mathsf{T}.x)$, where $\bar{F}$ translates to $F \Leftarrow (\mathsf{S}(\mathtt{int}) \to \mathsf{T}) \to \mathsf{T}$. By inversion, F must have the form $\lambda g.M$, and $g{:}(\mathsf{S}(\mathtt{int}) \to \mathsf{T}) \vdash_I M \Leftarrow \mathsf{T}$. Then we would obtain:

$$\frac{\bar{F} \searrow \lambda g.M \quad \lambda x{:}\mathsf{T}.x \searrow \lambda x.x}{\bar{F}(\lambda x{:}\mathsf{T}.x) \searrow [\lambda x.x/g]M} \ (wrong)$$

But the resulting substitution is not well-formed, because the substitutend $\lambda x.x$ cannot be assigned the required type $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$. We must coerce the substitutend from $\mathsf{T} \to \mathsf{T}$ to $\mathsf{S}(\mathtt{int}) \to \mathsf{T}$, and that requires that the types be made available to the translation.

On the other hand, it is actually *not* necessary that the translation generate a term explicitly. As we saw in the previous section, transparent types fully specify a term, so instead of returning a term explicitly, we return the transparent type containing that term. We thus give both pieces of information in a single syntactic object.

The complete translation is given in Figure 6. It consists of a type-to-type judgement, written $\Gamma \vdash \bar{A} \searrow A$, and term-to-transparent-type judgement, written $\Gamma \vdash \bar{M} \searrow A$. Both judgements are relative to an IL context.

Two rules are worthy of special attention. The variable rule (and the constant rule is similar) returns the transparent type containing just the eta-expansion of the variable:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \searrow \mathsf{Self}(\mathsf{E}(x : A) : A)}$$

The application rule first translates the function $\bar{M}$ to a function type $\Pi x{:}A.B$ and the argument $\bar{N}$ to a type C. Then it uses the subtyping judgement $C \leq A \rightsquigarrow N$ to obtain the translation of $\bar{N}$ as a term belonging to A, which is finally substituted for x in B:

$$\frac{\Gamma \vdash \bar{M} \searrow \Pi x{:}A.B \quad \Gamma \vdash \bar{N} \searrow C \quad C \leq A \rightsquigarrow N}{\Gamma \vdash \bar{M}\bar{N} \searrow [N/x]B}$$

Translation produces only well-formed types:

Lemma 5.1 (Translation regularity)

1. *If* $\Gamma \vdash \bar{A} \searrow A$ *then* $\Gamma \vdash_I A$.
2. *If* $\Gamma \vdash \bar{M} \searrow A$ *then* $\Gamma \vdash_I A$ *and* A *is transparent.*

Translation commutes with substitution, except that it can result in a subtype of the type one might expect. For example, consider the EL substitution $[\lambda x{:}\mathsf{T}.\mathtt{int}/f]f$, where f is assumed to have type $\mathsf{S}(\mathtt{int}) \to \mathsf{S}(\mathtt{int})$. Observe that $\lambda x{:}\mathsf{T}.\mathtt{int} \searrow \Pi x{:}\mathsf{T}.\mathsf{S}(\mathtt{int})$ and $f{:}\mathsf{S}(\mathtt{int}) \to \mathsf{S}(\mathtt{int}) \vdash f \searrow \Pi x{:}\mathsf{S}(\mathtt{int}).\mathsf{S}(\mathtt{int})$ and $\mathsf{T} \to \mathsf{S}(\mathtt{int}) \leq \mathsf{S}(\mathtt{int}) \to \mathsf{S}(\mathtt{int}) \rightsquigarrow \lambda x.\mathsf{at}(\mathtt{int})$.

When the substitution of $\lambda x{:}\mathsf{T}.\mathtt{int}$ for f in f is carried out in the EL before translation, we obtain $[\lambda x{:}\mathsf{T}.\mathtt{int}/f]f \searrow \Pi x{:}\mathsf{T}.\mathsf{S}(\mathtt{int})$. However, when we carry out the substitution in the IL, after translation, we get $[\lambda x.\mathsf{at}(\mathtt{int})/f](\Pi x{:}\mathsf{S}(\mathtt{int}).\mathsf{S}(\mathtt{int})) = \Pi x{:}\mathsf{S}(\mathtt{int}).\mathsf{S}(\mathtt{int})$, which is a supertype of $\Pi x{:}\mathsf{T}.\mathsf{S}(\mathtt{int})$. This phenomenon is generalized in Lemma 5.2:

Lemma 5.2 (Translation substitution)

1. *Suppose* $\Gamma \vdash \bar{M} \searrow C$ *and* $\Gamma, x{:}A, \Gamma' \vdash \bar{N} \searrow B$ *and* $C \leq A \rightsquigarrow M$, *where* $\Gamma \vdash_I A$. *Then* $\Gamma, [M/x]\Gamma' \vdash [\bar{M}/x]\bar{N} \searrow D$ *for some* $D \leq [M/x]B \rightsquigarrow O$.
2. *Suppose* $\Gamma \vdash \bar{M} \searrow C$ *and* $\Gamma, x{:}A, \Gamma' \vdash \bar{B} \searrow B$ *and* $C \leq A \rightsquigarrow M$, *where* $\Gamma \vdash_I A$. *Then* $\Gamma, [M/x]\Gamma' \vdash [\bar{M}/x]\bar{B} \searrow [M/x]B$.

Proof Sketch

By simultaneous induction on the derivations of $\Gamma, x{:}A, \Gamma' \vdash \bar{N} \searrow B$ and $\Gamma, x{:}A, \Gamma' \vdash \bar{B} \searrow B$.

In the case in which $\bar{N}$ is the substitution variable, we have $\bar{N} = x$ and $B = \mathsf{Self}(\mathsf{E}(x : A) : A)$. Consequently $[\bar{M}/x]\bar{N} = \bar{M}$, which translates to C. Thus let $D = C$ and also let $O = M$. It remains to show that $C \leq [M/x]B \rightsquigarrow M$. By self substitution and variable expansion, $[M/x]B = \mathsf{Self}(M : A)$. Then $C \leq \mathsf{Self}(M : A) \rightsquigarrow M$ by subtype strengthening.

Let the context translation $(\vdash \bar{\Gamma} \searrow \Gamma)$ be the pointwise extension of the type translation. Now we can prove completeness of the translation:

Theorem 5.3 (Completeness)

1. *Suppose* $\bar{\Gamma} \vdash_E \bar{A}$. *Then* $\Gamma \vdash \bar{A} \searrow A$, *where* $\bar{\Gamma} \searrow \Gamma$.
2. *Suppose* $\bar{\Gamma} \vdash_E \bar{M} : \bar{A}$. *Then* $\Gamma \vdash \bar{A} \searrow A$ *and* $\Gamma \vdash \bar{M} \searrow B$ *and* $B \leq A \rightsquigarrow M$, *where* $\bar{\Gamma} \searrow \Gamma$.
3. *Suppose* $\bar{\Gamma} \vdash_E \bar{A} \equiv \bar{B}$. *Then* $\Gamma \vdash \bar{A} \searrow A$ *and* $\Gamma \vdash \bar{B} \searrow A$, *where* $\bar{\Gamma} \searrow \Gamma$.
4. *Suppose* $\bar{\Gamma} \vdash_E \bar{A} \leq \bar{B}$. *Then* $\Gamma \vdash \bar{A} \searrow A$ *and* $\Gamma \vdash \bar{B} \searrow B$ *and* $A \leq B \rightsquigarrow M$, *where* $\bar{\Gamma} \searrow \Gamma$.
5. *Suppose* $\bar{\Gamma} \vdash_E \bar{M} \equiv \bar{N} : \bar{A}$. *Then* $\Gamma \vdash \bar{A} \searrow A$ *and* $\Gamma \vdash \bar{M} \searrow B$ *and* $\Gamma \vdash \bar{N} \searrow C$ *and* $B \leq A \rightsquigarrow M$ *and* $C \leq A \rightsquigarrow M$, *where* $\bar{\Gamma} \searrow \Gamma$.

5.1 Soundness

To prove soundness of the translation, we need one more definition. For any IL atom or term, we define its *transliteration* as the EL term obtained by rewriting it in the syntax of the EL, dropping `at` markers and adding appropriate type annotations to lambdas. Transliteration of IL types into EL types is defined similarly.

In order to determine appropriate type annotations for lambdas, transliteration needs access to the types of atom and terms. Thus, the transliteration judgements are written $\Gamma \vdash R \Rightarrow A \nearrow \bar{M}$, $\Gamma \vdash M \Leftarrow A \nearrow \bar{M}$, and $\Gamma \vdash A \nearrow \bar{A}$. Their precise definitions are given in Appendix B. Let context transliteration $(\vdash \bar{\Gamma} \nearrow \Gamma)$ be the pointwise extension of type transliteration.

Note that transliteration does not commute with substitution in the sense of hereditary substitution in the IL begetting substitution in the EL. Since hereditary substitution contracts redices and EL substitution does not, the best we can say is when M and N transliterate to $\bar{M}$ and $\bar{N}$, and $[M/x]N$ transliterates to $\bar{O}$, then $[\bar{M}/x]\bar{N}$ beta-reduces (in zero or more steps) to $\bar{O}$. It follows (under suitable well-formedness assumptions) that $[\bar{M}/x]\bar{N}$ and $\bar{O}$ are equivalent terms.

After establishing some basic properties of transliteration (which we omit for brevity), we are ready to prove soundness.

Lemma 5.4 (Subtype transliteration) *Suppose* $A \leq B \rightsquigarrow x.M$ *and* $\Gamma \vdash A \nearrow \bar{A}$ *and* $\Gamma \vdash B \nearrow \bar{B}$ *and* $\Gamma, x{:}A \vdash M \Leftarrow B \nearrow \bar{M}$. *Then* $\bar{\Gamma} \vdash_E \bar{A} \leq \bar{B}$ *and* $\bar{\Gamma}, x{:}\bar{A} \vdash_E x \equiv \bar{M} : \bar{B}$, *where* $\vdash \Gamma \nearrow \bar{\Gamma}$.

Theorem 5.5 (Translation inversion)

1. *Suppose* $\Gamma \vdash \bar{M} \searrow A$. *Then* $\Gamma \vdash A \nearrow \bar{A}$ *and* $\bar{\Gamma} \vdash_E \bar{M} : \bar{A}$, *where* $\vdash \Gamma \nearrow \bar{\Gamma}$.
2. *Suppose* $\Gamma \vdash \bar{A} \searrow A$. *Then* $\Gamma \vdash A \nearrow \bar{A}'$ *and* $\bar{\Gamma} \vdash_E \bar{A} \equiv \bar{A}'$, *where* $\vdash \Gamma \nearrow \bar{\Gamma}$.
3. *Suppose* $\vdash \bar{\Gamma} \searrow \Gamma$. *Then* $\vdash \Gamma \nearrow \bar{\Gamma}'$ *and* $\bar{\Gamma}$ *is pointwise equivalent to* $\bar{\Gamma}'$.

Corollary 5.6 (Soundness) *Suppose* $\Gamma \vdash \bar{M} \searrow A$ *and* $\Gamma \vdash \bar{N} \searrow B$ *and* $\Gamma \vdash \bar{C} \searrow C$ *and* $A \leq C \rightsquigarrow O$ *and* $B \leq C \rightsquigarrow O$, *where* $\vdash \bar{\Gamma} \searrow \Gamma$. *Then* $\bar{\Gamma} \vdash_E \bar{M} \equiv \bar{N} : \bar{C}$.

Proof

By translation inversion, A, B, C, and Γ transliterate to $\bar{A}$, $\bar{B}$, $\bar{C}'$, and $\bar{\Gamma}'$, with $\bar{\Gamma}' \vdash_E \bar{M} : \bar{A}$, $\bar{\Gamma}' \vdash_E \bar{N} : \bar{B}$, $\bar{\Gamma}' \vdash_E C \equiv C'$, and $\bar{\Gamma}$ pointwise equivalent to $\bar{\Gamma}'$. Let $\Gamma \vdash O \Leftarrow C \nearrow \bar{O}$. By subtype transliteration, $\bar{\Gamma}', x{:}\bar{A} \vdash_E x \equiv \bar{O} : \bar{C}'$. By substitution, $\bar{\Gamma}' \vdash_E \bar{M} \equiv \bar{O} : \bar{C}'$. Similarly $\bar{\Gamma}' \vdash_E \bar{N} \equiv \bar{O} : \bar{C}'$, so $\bar{\Gamma}' \vdash_E \bar{M} \equiv \bar{N} : \bar{C}$. It follows that $\bar{\Gamma} \vdash_E \bar{M} \equiv \bar{N} : \bar{C}$, since $\bar{\Gamma}$ and $\bar{\Gamma}'$ are pointwise equivalent.

5.2 Peroration

We can now prove decidability of equivalence:

Corollary 5.7 (Decidability) *It is decidable whether* $\bar{\Gamma} \vdash_E \bar{M} \equiv \bar{N} : \bar{A}$.

Proof

Immediate from soundness and completeness, since the definitions of translation and IL subtyping are syntax directed, and therefore define an effective procedure.

With only a little more effort, we can prove consistency of equivalence. For example:

Corollary 5.8 (Consistency, one case) *It is not the case that* $\bar{\Gamma} \vdash_E \mathtt{int} \equiv \mathtt{arrow}\,\bar{M}\,\bar{N} : \mathsf{T}$.

Proof

Observe that, in any context, $\mathtt{int} \searrow \mathsf{S}(\mathtt{int})$ and if $\mathtt{arrow}\,\bar{M}\,\bar{N} \searrow A$ then A must have the form $\mathsf{S}(\mathtt{arrow}\,M\,N)$.

Now suppose $\bar{\Gamma} \vdash_E \mathtt{int} \equiv \mathtt{arrow}\,\bar{M}\,\bar{N} : \mathsf{T}$. By consistency, $\mathsf{S}(\mathtt{int}) \leq \mathsf{T} \rightsquigarrow O$ and $\mathsf{S}(\mathtt{arrow}\,M\,N) \leq \mathsf{T} \rightsquigarrow O$. This is a contradiction, since O must simultaneously take the form $\mathtt{int}$ and $\mathtt{arrow}\,M\,N$.

We can prove one further result that is also necessary for proving the type safety of programming languages:

Corollary 5.9 (Injectivity, one case) *Suppose* $\bar{\Gamma} \vdash_E \mathtt{list}\,\bar{M} \equiv \mathtt{list}\,\bar{N} : \mathsf{T}$. *Then* $\bar{\Gamma} \vdash_E \bar{M} \equiv \bar{N} : \mathsf{T}$.

Proof

By consistency and inversion, $\mathsf{S}(\mathtt{list}\,M) \leq \mathsf{T} \rightsquigarrow O$ and $\mathsf{S}(\mathtt{list}\,N) \leq \mathsf{T} \rightsquigarrow O$, where $\Gamma \vdash \bar{M} \searrow A$ and $\Gamma \vdash \bar{N} \searrow B$ and $A \leq \mathsf{T} \rightsquigarrow M$ and $B \leq \mathsf{T} \rightsquigarrow N$ and $\vdash \bar{\Gamma} \searrow \Gamma$. Consequently, $\mathsf{at}(\mathtt{list}\,M) = O = \mathsf{at}(\mathtt{list}\,N)$, so $M = N$. By soundness, $\Gamma \vdash \bar{M} = \bar{N} : \mathsf{T}$.

6 Conclusion

The method of hereditary substitutions establishes the existence of a substitution in canonical form. Cast in the terminology of reduction, they provide a strategy for computing normal forms. Thus, like their roots in cut elimination, hereditary substitutions can be likened to a weak normalization argument. It does not provide a strong normalization argument. In fact, it is not even clear what an appropriate statement of strong normalization would be, since the usual treatment of singletons is to expand them, not reduce them.

Our proof makes contact with logical frameworks in two different ways. Not only is it formalized in a logical framework, but its main technical devices (canonical presentations and hereditary substitution) are borrowed from logical frameworks.

All the results of this paper are formalized in Twelf, and take up about 41 thousand lines of heavily commented Twelf code. The development also includes several extensions, such as bases that assign non-simple types to constants. About 40% of the development is spent establishing the properties of hereditary substitution and eta-expansion.

The main complication arising in the Twelf development but not on paper is in the IL substitution lemma. A naive transcription of the substitution lemma into Twelf requires the substitution variable to appear last in the context, but that invariant cannot be maintained without reordering assumptions, which is impossible in the presence of dependent types. We resolve this difficulty by using explicit contexts [3].

The Twelf development is included in a larger Twelf proof showing type safety of an internal language for Standard ML [8]. The singleton portion takes up roughly two-thirds of the type-safety proof. Thus, it seems that Standard ML's regime for controlled propagation of type information across module boundaries by itself contributes a large portion of the complexity of the entire language. Fortunately, none of the development in this paper (other than the choice of basis) needed to be tailored to Standard ML. The larger development was able to use the singleton calculus development off-the-shelf.

Acknowledgment

We are grateful to Kevin Watkins for his suggestion to apply the hereditary substitution technique to this problem, and for his assistance in formulating the internal language.

References

[1] David Aspinall. Subtyping with singleton types. In *Eighth International Workshop on Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 1–15, Kazimierz, Poland, September 1994. Springer.

[2] Karl Crary. Sound and complete elimination of singleton kinds. *ACM Transactions on Computational Logic*, 8(2), April 2007. An earlier version appeared in 2003 Workshop on Types in Compilation.

[3] Karl Crary. Explicit contexts in LF. In *Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, Pittsburgh, Pennsylvania, 2008.

[4] Amy Felty. Encoding dependent types in an intuitionistic logic. In *Logical Frameworks*, pages 214–251. Cambridge University Press, 1991.

[5] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[6] Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4–5), July 2007.

[7] R. Hindley. Reductions of residuals are finite. *Transactions of the American Mathematical Society*, 240, 1978.

[8] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of Standard ML. In *Thirty-Fourth ACM Symposium on Principles of Programming Languages*, Nice, France, January 2007.

[9] Xavier Leroy. *The Objective Caml System, Release 1.00*. Institut National de Recherche en Informatique et Automatique (INRIA), 1996.

[10] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. The MIT Press, Cambridge, Massachusetts, 1997.

[11] Frank Pfenning and Carsten Schürmann. *Twelf User's Guide, Version 1.4*, 2002. Available electronically at `http://www.cs.cmu.edu/~twelf`.

[12] Claudio V. Russo. *Types for Modules*. Number 60 in Electronic Notes in Theoretical Computer Science. Elsevier, January 2003.

[13] Christopher A. Stone and Robert Harper. Deciding type equivalence in a language with singleton kinds. In *Twenty-Seventh ACM Symposium on Principles of Programming Languages*, Boston, January 2000. Extended version published as CMU technical report CMU-CS-99-155.

[14] Christopher A. Stone and Robert Harper. Extensional equivalence and singleton types. *ACM Transactions on Computational Logic*, 7(4), October 2006. An earlier version appeared in the 2000 Symposium on Principles of Programming Languages.

[15] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Carnegie Mellon University, School of Computer Science, 2002. Revised May 2003.

A EL typing rules

$\boxed{\bar{\Gamma} \vdash \bar{A}}$

$$\frac{}{\bar{\Gamma} \vdash \mathsf{T}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M})} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \Sigma x{:}\bar{A}.\bar{B}}$$

$\boxed{\bar{\Gamma} \vdash \bar{M} : \bar{A}}$

$$\frac{\mathsf{Basis}(c) = \bar{A}}{\bar{\Gamma} \vdash c : \bar{A}} \qquad \frac{\bar{\Gamma}(x) = \bar{A}}{\bar{\Gamma} \vdash x : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M} : \bar{B}}{\bar{\Gamma} \vdash \lambda x{:}\bar{A}.\bar{M} : \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B} \quad \bar{\Gamma} \vdash \bar{N} : \bar{A}}{\bar{\Gamma} \vdash \bar{M}\bar{N} : [\bar{N}/x]\bar{B}}$$

$$\frac{\bar{\Gamma} \vdash \bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \bar{N} : [\bar{M}/x]\bar{B} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \langle \bar{M}, \bar{N} \rangle : \Sigma x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \Sigma x{:}\bar{A}.\bar{B}}{\bar{\Gamma} \vdash \pi_1 \bar{M} : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \Sigma x{:}\bar{A}.\bar{B}}{\bar{\Gamma} \vdash \pi_2 \bar{M} : [\pi_1 \bar{M}/x]\bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \bar{M} : \mathsf{S}(\bar{M})}$$

$$\frac{\bar{\Gamma} \vdash \bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \bar{A} \leq \bar{B}}{\bar{\Gamma} \vdash \bar{M} : \bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M}x : \bar{B}}{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \pi_1 \bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \pi_2 \bar{M} : [\pi_1 \bar{M}/x]\bar{B} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \bar{M} : \Sigma x{:}\bar{A}.\bar{B}}$$

$\boxed{\bar{\Gamma} \vdash \bar{A} \equiv \bar{B}}$

$$\frac{\bar{\Gamma} \vdash \bar{A}}{\bar{\Gamma} \vdash \bar{A} \equiv \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{B} \equiv \bar{A}}{\bar{\Gamma} \vdash \bar{A} \equiv \bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \equiv \bar{B} \quad \bar{\Gamma} \vdash \bar{B} \equiv \bar{C}}{\bar{\Gamma} \vdash \bar{A} \equiv \bar{C}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M}) \equiv \mathsf{S}(\bar{N})}$$

$$\frac{\bar{\Gamma} \vdash \bar{A} \equiv \bar{A}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B} \equiv \bar{B}'}{\bar{\Gamma} \vdash \Pi x{:}\bar{A}.\bar{B} \equiv \Pi x{:}\bar{A}'.\bar{B}'} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \equiv \bar{A}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B} \equiv \bar{B}'}{\bar{\Gamma} \vdash \Sigma x{:}\bar{A}.\bar{B} \equiv \Sigma x{:}\bar{A}'.\bar{B}'}$$

$\boxed{\bar{\Gamma} \vdash \bar{A} \leq \bar{B}}$

$$\frac{\bar{\Gamma} \vdash \bar{A} \equiv \bar{B}}{\bar{\Gamma} \vdash \bar{A} \leq \bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \leq \bar{B} \quad \bar{\Gamma} \vdash \bar{B} \leq \bar{C}}{\bar{\Gamma} \vdash \bar{A} \leq \bar{C}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{T}}{\bar{\Gamma} \vdash \mathsf{S}(\bar{M}) \leq \mathsf{T}}$$

$$\frac{\bar{\Gamma} \vdash \bar{A}' \leq \bar{A} \quad \bar{\Gamma}, x{:}\bar{A}' \vdash \bar{B} \leq \bar{B}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \Pi x{:}\bar{A}.\bar{B} \leq \Pi x{:}\bar{A}'.\bar{B}'} \qquad \frac{\bar{\Gamma} \vdash \bar{A} \leq \bar{A}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B} \leq \bar{B}' \quad \bar{\Gamma}, x{:}\bar{A}' \vdash \bar{B}'}{\bar{\Gamma} \vdash \Sigma x{:}\bar{A}.\bar{B} \leq \Sigma x{:}\bar{A}'.\bar{B}'}$$

$\boxed{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \bar{A}}$

$$\frac{\bar{\Gamma} \vdash \bar{M} : \bar{A}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{M} : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{N} \equiv \bar{M} : \bar{A}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \bar{A} \quad \bar{\Gamma} \vdash \bar{N} \equiv \bar{O} : \bar{A}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{O} : \bar{A}}$$

$$\frac{\bar{\Gamma} \vdash \bar{A} \equiv \bar{A}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M} \equiv \bar{M}' : \bar{B}}{\bar{\Gamma} \vdash \lambda x{:}\bar{A}.\bar{M} \equiv \lambda x{:}\bar{A}'.\bar{M}' : \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B} \quad \bar{\Gamma} \vdash \bar{N} : \bar{A}}{\bar{\Gamma} \vdash \bar{M}\bar{N} : [\bar{M}/x]\bar{B}}$$

$$\frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{M}' : \bar{A} \quad \bar{\Gamma} \vdash \bar{N} \equiv \bar{N}' : [\bar{M}/x]\bar{B} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \langle \bar{M}, \bar{N} \rangle \equiv \langle \bar{M}', \bar{N}' \rangle : \Sigma x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{M}' : \Sigma x{:}\bar{A}.\bar{B}}{\bar{\Gamma} \vdash \pi_1 \bar{M} \equiv \pi_1 \bar{M}' : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{M}' : \Sigma x{:}\bar{A}.\bar{B}}{\bar{\Gamma} \vdash \pi_2 \bar{M} \equiv \pi_2 \bar{M}' : [\pi_1 \bar{M}/x]\bar{B}}$$

$$\frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{S}(\bar{M})} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \mathsf{S}(\bar{N})}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \mathsf{T}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \bar{A} \quad \bar{\Gamma} \vdash \bar{A} \leq \bar{B}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \bar{B}}$$

$$\frac{\bar{\Gamma} \vdash \bar{M} : \Pi x{:}\bar{A}.\bar{B}' \quad \bar{\Gamma} \vdash \bar{N} : \Pi x{:}\bar{A}.\bar{B}'' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M}x \equiv \bar{N}x : \bar{B}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Pi x{:}\bar{A}.\bar{B}' \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{M}x \equiv \bar{N}x : \bar{B}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Pi x{:}\bar{A}.\bar{B}}$$

$$\frac{\bar{\Gamma} \vdash \pi_1 \bar{M} \equiv \pi_1 \bar{N} : \bar{A} \quad \bar{\Gamma} \vdash \pi_2 \bar{M} \equiv \pi_2 \bar{N} : [\pi_1 \bar{M}/x]\bar{B} \quad \bar{\Gamma}, x{:}\bar{A} \vdash \bar{B}}{\bar{\Gamma} \vdash \bar{M} \equiv \bar{N} : \Sigma x{:}\bar{A}.\bar{B}}$$

$$\frac{\bar{\Gamma}, x{:}\bar{A} \vdash \bar{M} : \bar{B} \quad \bar{\Gamma} \vdash \bar{N} : \bar{A}}{\bar{\Gamma} \vdash (\lambda x{:}\bar{A}.\bar{M})\bar{N} \equiv [\bar{N}/x]\bar{M} : [\bar{N}/x]\bar{B}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \bar{N} : \bar{B}}{\bar{\Gamma} \vdash \pi_1 \langle \bar{M}, \bar{N} \rangle \equiv \bar{M} : \bar{A}} \qquad \frac{\bar{\Gamma} \vdash \bar{M} : \bar{A} \quad \bar{\Gamma} \vdash \bar{N} : \bar{B}}{\bar{\Gamma} \vdash \pi_2 \langle \bar{M}, \bar{N} \rangle \equiv \bar{N} : \bar{A}}$$

B Transliteration

$\boxed{\Gamma \vdash A \nearrow \bar{A}}$

$$\frac{}{\Gamma \vdash \mathsf{T} \nearrow \mathsf{T}} \qquad \frac{\Gamma \vdash R \Rightarrow \mathsf{T} \nearrow \bar{M}}{\Gamma \vdash \mathsf{S}(R) \nearrow \mathsf{S}(\bar{M})} \qquad \frac{\Gamma \vdash A \nearrow \bar{A} \quad \Gamma, x{:}A \vdash B \nearrow \bar{B}}{\Gamma \vdash \Pi x{:}A.B \nearrow \Pi x{:}\bar{A}.\bar{B}} \qquad \frac{\Gamma \vdash A \nearrow \bar{A} \quad \Gamma, x{:}A \vdash B \nearrow \bar{B}}{\Gamma \vdash \Sigma x{:}A.B \nearrow \Sigma x{:}\bar{A}.\bar{B}}$$

$\boxed{\Gamma \vdash R \Rightarrow A \nearrow \bar{M}}$

$$\frac{\mathsf{Basis}(c) = A}{\Gamma \vdash c \Rightarrow A \nearrow c} \qquad \frac{\Gamma(x) = A}{\Gamma \vdash x \Rightarrow A \nearrow x} \qquad \frac{\Gamma \vdash R \Rightarrow \Pi x{:}A.B \nearrow \bar{N} \quad \Gamma \vdash M \Leftarrow A \nearrow \bar{M} \quad \Gamma \vdash [M/x]B}{\Gamma \vdash RM \Rightarrow [M/x]B \nearrow \bar{N}\bar{M}}$$

$$\frac{\Gamma \vdash R \Rightarrow \Sigma x{:}A.B \nearrow \bar{M}}{\Gamma \vdash \pi_1 R \Rightarrow A \nearrow \pi_1 \bar{M}} \qquad \frac{\Gamma \vdash R \Rightarrow \Sigma x{:}A.B \nearrow \bar{M}}{\Gamma \vdash \pi_2 R \Rightarrow \lceil \pi_1 R/x \rceil B \nearrow \pi_2 \bar{M}}$$

$\boxed{\Gamma \vdash M \Leftarrow A \nearrow \bar{M}}$

$$\frac{\Gamma \vdash R \Rightarrow \mathsf{T} \nearrow \bar{M}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{T} \nearrow \bar{M}} \qquad \frac{\Gamma \vdash R \Rightarrow \mathsf{T} \nearrow \bar{M}}{\Gamma \vdash \mathsf{at}(R) \Leftarrow \mathsf{S}(R) \nearrow \bar{M}} \qquad \frac{\Gamma \vdash A \nearrow \bar{A} \quad \Gamma, x{:}A \vdash M \Leftarrow B \nearrow \bar{M}}{\Gamma \vdash \lambda x.M \Leftarrow \Pi x{:}A.B \nearrow \lambda x{:}\bar{A}.\bar{M}}$$

$$\frac{\Gamma \vdash M \Leftarrow A \nearrow \bar{M} \quad \Gamma \vdash N \Leftarrow [M/x]B \nearrow \bar{N} \quad \Gamma, x{:}A \vdash B}{\Gamma \vdash \langle M, N \rangle \Leftarrow \Sigma x{:}A.B \nearrow \langle \bar{M}, \bar{N} \rangle}$$