Extensional Equivalence and Singleton Types

CHRISTOPHER A. STONE
Harvey Mudd College
and
ROBERT HARPER
Carnegie Mellon University

We study the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus, which contains singleton types $\mathcal{S}(M)$ classifying terms of base type provably equivalent to the term M. The system includes dependent types for pairs and functions (Σ and Π) and a subtyping relation induced by regarding singletons as subtypes of the base type. The decidability of type checking for this language is non-obvious, since to type check we must be able to determine equivalence of well-formed terms. But in the presence of singleton types, the provability of an equivalence judgment $\Gamma \vdash M_1 \equiv M_2 : A$ can depend both on the typing context Γ and on the particular type A at which M_1 and M_2 are compared.

We show how to prove decidability of term equivalence, hence of type checking, in $\lambda_{\leq}^{\Pi\Sigma S}$ by exhibiting a type-directed algorithm for directly computing normal forms. The correctness of normalization is shown using an unusual variant of Kripke logical relations organized around sets; rather than defining a logical equivalence relation, we work directly with (subsets of) the corresponding equivalence classes.

We then provide a more efficient algorithm for checking type equivalence without constructing normal forms. We also show that type checking, subtyping, and all other judgments of the system are decidable.

The $\lambda_{\leq}^{\Pi\Sigma S}$ calculus models type constructors and kinds in the intermediate language used by the TILT compiler for Standard ML to implement the SML module system. The decidability of $\lambda_{\leq}^{\Pi\Sigma S}$ term equivalence allows us to show decidability of type checking for TILT's intermediate language. We also obtain a consistency result that allows us to prove type safety for the intermediate language. The algorithms derived here form the core of the type checker used for internal type checking in TILT.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*applicative (functional) languages*; F.3.3 [**Logics and Meanings of Programs**]: Studies

---

Parts of this work occurred while C. A. Stone was at Carnegie Mellon University.
This research was supported in part by the US Army Research Office under Grant No. DAAH04-94-G-0289 and in part by the Advanced Research Projects Agency CSTO under the title "The Fox Project: Advanced Languages for Systems Software," ARPA Order No. C533, issued by ESC/ENS under Contract No. F19628-95-C-0050.
The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.
Authors' addresses: C. A. Stone, Harvey Mudd College, 301 Platt Boulevard, Claremont, CA 91711; email: stone@cs.hmc.edu; R. Harper, Computer Science Department, Carnegie Mellon University, 7113 Wean Hall, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890; email: rwh@cs.cmu.edu.

of Program Constructs—*type structure*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*lambda calculus and related systems*

General Terms: Languages, Theory

Additional Key Words and Phrases: Singleton types, logical relations, equivalence algorithms

1. INTRODUCTION

Lambda-calculus models of programming languages are of both theoretical and practical importance. On the theory side they support formal reasoning about the language, including type safety and equational properties. On the practical side they support certifying compilation through type-directed translations between typed intermediate languages [Tarditi et al. 1996; Morrisett et al. 1997].

Conventional type systems such as F^{ω} go a long way towards modeling programming language constructs such as polymorphism, but they do not provide an adequate account of type definitions. For example, although it is well known how to define local definitions (`let`) in terms of function application, this does not directly translate for definitions of type variables. If we have the polymorphic identity $\mathtt{id} : \forall\alpha.\alpha \rightharpoonup \alpha$, then the expression

$$\mathsf{let}\,\alpha = \mathtt{int}\times\mathtt{int}\ \mathsf{in}\ \mathtt{id}[\alpha](3,4)$$

seems unobjectionable, whereas the corresponding application

$$(\Lambda\alpha.\mathtt{id}[\alpha](3,4))[\mathtt{int}\times\mathtt{int}]$$

is ill-typed because its subterm $\Lambda\alpha.\mathtt{id}[\alpha](3,4)$ is not well formed.

More complex type definitions occur in the Standard ML module language, where we can have a structure `Set` with the signature

```
sig
   type elem = int
   type set
   type setpair = set * set

   val empty     : set
   val insert    : set * elem -> set
   val member    : set * elem -> bool
   val union     : setpair -> set
   val intersect : setpair -> set
end
```

specifying definitions for the types `Set.elem` and `Set.setpair`, where the latter has a dependency on the abstract type `Set.set`. Even if assume we have the rest of the program available, i.e., that we are not doing separate compilation, we cannot simply eliminate these definitions by syntactic substitution, replacing all occurrences of `Set.elem` by `int`. Type components in SML can be referenced indirectly. For example, if we were to define

```
structure Set2 = Set
```

then the type components of `Set2` are equal to those of `Set`. We must thereafter treat `Set2.elem` as equal to `Set.elem` (hence equal to `int`), `Set2.set` as equal to `Set.set`, and `Set2.setpair` as equal to `Set.setpair`. Functors (parameterized modules) return modules whose types depend on the types in their argument, and hence have even more complex type propagation behavior. The theory of any such language must be able to track the relations between types.

One can formally define languages allowing definitions of type variables [Severi and Poll 1994], or allowing type definitions in module interfaces [Harper and Lillibridge 1994], or even allowing type definitions within arguments of polymorphic abstractions [Minamide et al. 1996]. A more uniform approach is to allow definitions anywhere types are described, by making definitions part of the kind system of the language. A natural way of doing so is to add a kind $\mathcal{S}(\tau)$ to classify exactly those types equal to τ.

Singletons can then be used to describe and control the propagation of type definitions and sharing. The type τ has kind $\mathcal{S}(\sigma)$ if and only if the types τ and σ are provably equivalent, so the hypothesis that a variable α has kind $\mathcal{S}(\tau)$ expresses that α is a type variable with definition τ. This models open-scope (non-delimited) definitions in the source language.

Furthermore, singletons provide "partial" definitions for variables. If β is a pair of type constructors with kind $\mathcal{S}(\mathtt{int})\times\star$, then the first component of this pair, $\pi_1\beta$, is `int`. However, this kind tells us nothing about the identity of the $\pi_2\beta$, except that it has kind $\star$, the kind of ordinary types. As in the above example, partial definitions allow natural modeling of definitions in a modular system, where some components of a module have known definitions and others remain abstract.

The TILT compiler for Standard ML [Petersen et al. 2000] uses a typed intermediate language based on predicative F_ω extended with singleton kinds. Modules are represented in this language using a phase-splitting interpretation [Harper et al. 1990; Shao 1998]. The main idea is that modules can be split into type constructor and a term, while signatures split in a parallel way into a kind and a type. Singleton kinds are used to model definitions and type sharing specifications in module signatures, dependent record kinds model the type parts of structure signatures, dependent function kinds model the type parts of functor signatures, and subkinding models (noncoercive) signature matching.

A crucial property of typed intermediate languages is that type checking be decidable. For many languages this reduces to checking equivalence of types. We are therefore concerned with establishing the decidability of equivalence in the presence of singletons, ideally by showing the correctness of a practical algorithm.

1.1 From Singleton Kinds to Singleton Types

Since ordinary expressions in the TILT typed intermediate language have no effect on equivalence of type constructors, we abstract the problem to a simpler two-level calculus $\lambda_{\leq}^{\Pi\Sigma S}$ and study *term* equivalence in a language with singleton and dependent *types*. These levels correspond to the type constructors and kinds of TILT, but $\lambda_{\leq}^{\Pi\Sigma S}$ is of general interest in its own right.

In Section 2, we define the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus. The fact that any two terms having the same singleton type are equal is a form of extensionality, and so it seems natural for $\lambda_{\leq}^{\Pi\Sigma S}$ to define equivalence of functions and pairs extensionally as well.

This leads to a very interesting equational theory; for example, β-equivalence becomes admissible. More importantly, whether two terms are provably equivalent can depend on both the typing context and—less obviously—on the type at which the terms are compared. The identity function and a function always returning some constant c are naturally inequivalent, but if we consider them as functions of type $\mathcal{S}(c) \to \mathcal{S}(c)$, then extensionally the two functions are equal; they return the same result for any argument of type $\mathcal{S}(c)$, that is, for any argument equal to c. The common method of implementing equivalence via context-insensitive rewrite rules is thus not directly applicable for our calculus.

Section 3 contains proofs for many standard properties of the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus, such as preservation of well typedness under substitutions and the admissibility of useful rules. We show that although the definition of $\lambda_{\leq}^{\Pi\Sigma S}$ includes only restricted form of singleton type, more general singletons are definable.

In Section 4, we present an algorithm for computing normal forms of terms. This is broadly similar to the algorithm used in Typed Operational Semantics [Goguen 1994], but our algorithm is type-directed (normalization is guided by the type at which we are normalizing rather than by the shape of the term), and we compute what are essentially long normal forms.

We then show the correctness of this normalization algorithm. Our argument relies on a novel form of set-based Kripke logical relation that can handle the complications induced by singleton and dependent types. Intuitively, rather than define a logical relation for equivalence, we work directly with subsets of the corresponding equivalence classes.

Using the correctness of normalization, in Section 5 we present a more efficient and more direct binary equivalence algorithm.

In Section 6, we sketch algorithms for deciding the remaining type and term-level judgments (e.g., given a well formed context and a term M, determine whether there is a type A such that M is well formed with type A). All the algorithms are sound and complete with respect to the language definition, and are terminating. These algorithms form the core of the TILT compiler's type checking implementation.

Finally, we survey the related literature and conclude.

For space reasons, some proofs, have been omitted or abbreviated; further details can be found in the companion technical report [Stone and Harper 2004].

2. THE $\lambda_{\leq}^{\Pi\Sigma S}$ CALCULUS

2.1 Syntax of $\lambda_{\leq}^{\Pi\Sigma S}$

The abstract syntax of $\lambda_{\leq}^{\Pi\Sigma S}$ is shown in Figure 1. As usual, we work modulo renaming of bound variables. The meaning of each construct is explained in tandem with the static semantics (type system) of $\lambda_{\leq}^{\Pi\Sigma S}$ below.

Typing Contexts	$\Gamma, \Delta ::=$	$\bullet$	Empty context
	$\mid$	$\Gamma, x : A$	Context extension
Types	$A, B ::=$	b	Base type
	$\mid$	$\mathcal{S}(M)$	Singleton type
	$\mid$	$\Pi x{:}A'.\, A''$	Dependent function type
	$\mid$	$\Sigma x{:}A'.\, A''$	Dependent pair type
Terms	$M, N ::=$	$c_1 \mid c_2 \mid \ldots$	Base constants
	$\mid$	$x \mid y \mid \ldots$	Variables
	$\mid$	$\lambda x{:}A'.\, M$	Function
	$\mid$	$M\, M'$	Application
	$\mid$	$\langle M', M'' \rangle$	Pair
	$\mid$	$\pi_i M$	Projection

Fig. 1. Syntax of the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus.

$$
\begin{array}{ll}
\mathrm{FV}(b) & := \emptyset \\
\mathrm{FV}(\mathcal{S}(M)) & := \mathrm{FV}(M) \\
\mathrm{FV}(\Pi x{:}A'.\, A'') & := \mathrm{FV}(A') \cup (\mathrm{FV}(A'') \setminus \{x\}) \\
\mathrm{FV}(\Sigma x{:}A'.\, A'') & := \mathrm{FV}(A') \cup (\mathrm{FV}(A'') \setminus \{x\}) \\
\\
\mathrm{FV}(c) & := \emptyset \\
\mathrm{FV}(x) & := \{x\} \\
\mathrm{FV}(\lambda x{:}A.\, M) & := \mathrm{FV}(A) \cup (\mathrm{FV}(M) \setminus \{x\}) \\
\mathrm{FV}(M\, M') & := \mathrm{FV}(M) \cup \mathrm{FV}(M') \\
\mathrm{FV}(\langle M', M'' \rangle) & := \mathrm{FV}(M') \cup \mathrm{FV}(M'') \\
\mathrm{FV}(\pi_i M) & := \mathrm{FV}(M)
\end{array}
$$

Fig. 2. Free-variable sets for types and terms.

2.1.1 *Substitutions.* The notation FV(*phrase*) refers to the set of free variables in *phrase* and is defined in Figure 2 by induction on syntax.

We use the metavariable γ to stand for an arbitrary mapping from term variables to terms. The notation γ(*phrase*) is used to represent the result of applying γ to all free variables in the phrase *phrase*, avoiding variable capture. The substitution which sends x to M and leaves all other variables unchanged is written $[M/x]$. If γ is a substitution, then $\gamma[x \mapsto M]$ stands for the mapping which sends x to M and behaves like γ for all other variables.

2.1.2 *Typing Contexts.* A *typing context* Γ (or simply *context* when this is unambiguous) represents assumptions for the types of free term variables. It is represented as a finite sequence of variable/classifier associations. Typing contexts in $\lambda_{\leq}^{\Pi\Sigma S}$ are ordered sequences rather than sets because of the dependencies introduced by singletons: types can refer to earlier term variables in the context.

2.2 Judgments of $\lambda_{\leq}^{\Pi\Sigma S}$

The *context validity* judgment $\Gamma \vdash \mathrm{ok}$ holds when a typing context Γ is well formed; every type appearing in the context must be well formed with respect to the preceding segment of the context.

The side-condition in Rule 2 ensures that variables are not bound in a context more than once; dom(Γ) is the set of all term variables assigned types by Γ. It

Well-Formed Context

$$\frac{}{\bullet \vdash \text{ok}} \tag{1}$$

$$\frac{\Gamma \vdash A}{\Gamma, x : A \vdash \text{ok}} \quad (x \notin \text{dom}(\Gamma)) \tag{2}$$

Well-Formed Type

$$\frac{\Gamma \vdash \text{ok}}{\Gamma \vdash b} \tag{3}$$

$$\frac{\Gamma \vdash M : b}{\Gamma \vdash \mathcal{S}(M)} \tag{4}$$

$$\frac{\Gamma, x : A' \vdash A''}{\Gamma \vdash \Pi x{:}A'.\, A''} \tag{5}$$

$$\frac{\Gamma, x : A' \vdash A''}{\Gamma \vdash \Sigma x{:}A'.\, A''} \tag{6}$$

Fig. 3. Judgments of $\lambda_{\leq}^{\Pi\Sigma S}$, continued.

follows that well-formed typing contexts can be used as finite functions: $\Gamma(x)$ represents the type associated with in Γ. Because contexts are finite sequences, there is an obvious definition for appending any two contexts and the result of appending Γ_1 and Γ_2 is written Γ_1, Γ_2.

We define a (purely syntactic) inclusion order on contexts. The relation $\Gamma_1 \subseteq \Gamma_2$ on contexts holds if Γ_1 appears as a (not necessarily consecutive) subsequence of Γ_2. Thus, if $\Gamma_1 \subseteq \Gamma_2$, then $\text{dom}(\Gamma_1) \subseteq \text{dom}(\Gamma_2)$ and $\Gamma_1(x) = \Gamma_2(x)$ for every $x \in \text{dom}(\Gamma_1)$. We also write $\Gamma_2 \supseteq \Gamma_1$ to mean $\Gamma_1 \subseteq \Gamma_2$.

2.2.1 *Types.* The *type validity* judgment $\Gamma \vdash A$ specifies when a type A is well formed with respect to a given typing context Γ. It is defined in Figure 3.

The premise $\Gamma \vdash \text{ok}$ in Rule 3 for the base type b helps ensure that in any proof of a judgment $\Gamma \vdash A$ there is strict subderivation proving $\Gamma \vdash \text{ok}$. A similar property will hold for all of the judgments (Proposition 3.1).

Well-formed singleton types in $\lambda_{\leq}^{\Pi\Sigma S}$ are restricted to contain terms only of the base type b, as shown in Rule 4. However, more general singleton types $\mathcal{S}_A(M)$, classifying terms equivalent to M at type A, are definable (see Section 2.3).

Rules (5) and (6) for Π and Σ types (dependent types of functions of terms and pairs of terms respectively) are standard. $\Pi x{:}A'.\, A''$ is the type of all functions which map an argument x of type A' to a result of type A'', where A'' may depend on x. Similarly, $\Sigma x{:}A'.\, A''$ is the type of all pairs of terms whose first component x has type A' and whose second component has type A'', where A'' may refer to x. Both $\Pi x{:}A'.\, A''$ and $\Sigma x{:}A'.\, A''$ bind the term variable x in A''. We use the usual notation $A' \times A''$ for $\Sigma x{:}A'.\, A''$ and $A' \to A''$ for $\Pi x{:}A'.\, A''$ when x is not free in A''.

It is often convenient to be able to induct over types ignoring constituent terms. We therefore define the *size* of a type to be a strictly positive integer, Subtyping

$$\frac{\Gamma \vdash M : b}{\Gamma \vdash \mathcal{S}(M) \leq b} \tag{7}$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : b}{\Gamma \vdash \mathcal{S}(M_1) \leq \mathcal{S}(M_2)} \tag{8}$$

$$\frac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash b \leq b} \tag{9}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Pi x{:}A_1'.\,A_1'' \\ \Gamma \vdash A_2' \leq A_1' \qquad \Gamma, x : A_2' \vdash A_1'' \leq A_2''\end{array}}{\Gamma \vdash \Pi x{:}A_1'.\,A_1'' \leq \Pi x{:}A_2'.\,A_2''} \tag{10}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Sigma x{:}A_2'.\,A_2'' \\ \Gamma \vdash A_1' \leq A_2' \qquad \Gamma, x : A_1' \vdash A_1'' \leq A_2''\end{array}}{\Gamma \vdash \Sigma x{:}A_1'.\,A_1'' \leq \Sigma x{:}A_2'.\,A_2''} \tag{11}$$

Type Equivalence

$$\frac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash b \equiv b} \tag{12}$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : b}{\Gamma \vdash \mathcal{S}(M_1) \equiv \mathcal{S}(M_2)} \tag{13}$$

$$\frac{\Gamma \vdash A_1' \equiv A_2' \qquad \Gamma, x : A_1' \vdash A_1'' \equiv A_2''}{\Gamma \vdash \Pi x{:}A_1'.\,A_1'' \equiv \Pi x{:}A_2'.\,A_2''} \tag{14}$$

$$\frac{\Gamma \vdash A_1' \equiv A_2' \qquad \Gamma, x : A_1' \vdash A_1'' \equiv A_2''}{\Gamma \vdash \Sigma x{:}A_1'.\,A_1'' \equiv \Sigma x{:}A_2'.\,A_2''} \tag{15}$$

Fig. 4. Judgments of $\lambda_{\leq}^{\Pi\Sigma S}$, continued.

specified by induction on the structure of types:

$$\begin{array}{ll} size(b) & = 1 \\ size(\mathcal{S}(M)) & = 2 \\ size(\Pi x{:}A'.\,A'') & = size(A') + size(A'') + 1 \\ size(\Sigma x{:}A'.\,A'') & = size(A') + size(A'') + 1 \end{array}$$

The size of a type depends only on "shape" and is thus invariant under substitutions. The key properties of this measure are that $size(\mathcal{S}(M)) > size(b)$ and that the size of a Π or Σ is strictly greater than the sizes of all substitution instances of its constituent types.

The *subtyping* judgment $\Gamma \vdash A_1 \leq A_2$ shown in Figure 4 defines a preorder on types, which may be understood to say that A_1 is less general (exposes more information about a term) than A_2; a term of type A_1 will be acceptable in every context requiring a term of type A_2.

$\mathcal{S}(M)$ is the type of "all terms of type b equivalent to M", any such term should be acceptable where a term of type b is expected. Thus, the key subtyping rule is Rule (7) (where the premise of this rule helps ensure that $\mathcal{S}(M)$ is well formed).

Well-Formed Term

$$\frac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash c_i \,:\, b} \tag{16}$$

$$\frac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash x \,:\, \Gamma(x)} \quad (x \in \mathrm{dom}(\Gamma)) \tag{17}$$

$$\frac{\Gamma, x : A' \vdash M \,:\, A''}{\Gamma \vdash \lambda x{:}A'.\, M \,:\, \Pi x{:}A'.\, A''} \tag{18}$$

$$\frac{\Gamma \vdash M \,:\, \Pi x{:}A'.\, A'' \qquad \Gamma \vdash M' \,:\, A'}{\Gamma \vdash M\, M' \,:\, [M'/x]A''} \tag{19}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Sigma x{:}A'.\, A'' \\ \Gamma \vdash M' \,:\, A' \qquad \Gamma \vdash M'' \,:\, [M'/x]A''\end{array}}{\Gamma \vdash \langle M', M''\rangle \,:\, \Sigma x{:}A'.\, A''} \tag{20}$$

$$\frac{\Gamma \vdash M \,:\, \Sigma x{:}A'.\, A''}{\Gamma \vdash \pi_1 M \,:\, A'} \tag{21}$$

$$\frac{\Gamma \vdash M \,:\, \Sigma x{:}A'.\, A''}{\Gamma \vdash \pi_2 M \,:\, [\pi_1 M/x]A''} \tag{22}$$

$$\frac{\Gamma \vdash M \,:\, b}{\Gamma \vdash M \,:\, \mathcal{S}(M)} \tag{23}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Sigma x{:}A'.\, A'' \\ \Gamma \vdash \pi_1 M \,:\, A' \qquad \Gamma \vdash \pi_2 M \,:\, [\pi_1 M/x]A''\end{array}}{\Gamma \vdash M \,:\, \Sigma x{:}A'.\, A''} \tag{24}$$

$$\frac{\begin{array}{c}\Gamma, x : A' \vdash M\, x \,:\, A'' \\ \Gamma \vdash M \,:\, \Pi x{:}A'.\, B'' \qquad \Gamma \vdash \Pi x{:}A'.\, B''\end{array}}{\Gamma \vdash M \,:\, \Pi x{:}A'.\, A''} \tag{25}$$

$$\frac{\Gamma \vdash M \,:\, A_1 \qquad \Gamma \vdash A_1 \,\le\, A_2}{\Gamma \vdash M \,:\, A_2} \tag{26}$$

Fig. 5. Judgments of $\lambda_{\le}^{\Pi\Sigma S}$, continued.

Subtyping between two singleton types coincides with equivalence because a term of type b equivalent to M_1 should appear in a context expecting a term equivalent to M_2 if and only if M_1 and M_2 are equivalent. Rule (9) helps ensure that subtyping is reflexive for all types, including b. The remaining subtyping rules lift the relation to Π and Σ types, following the usual co- and contravariance properties. The topmost premises in Rules (10) and (11) ensure that $\Gamma \vdash A_1 \,\le\, A_2$ implies $\Gamma \vdash A_1$ and $\Gamma \vdash A_2$.

Type equivalence, denoted $\Gamma \vdash A_1 \equiv A_2$, is essentially a symmetrized version of subtyping. We show later that $\Gamma \vdash A_1 \equiv A_2$ if and only if $\Gamma \vdash A_1 \,\le\, A_2$ and $\Gamma \vdash A_2 \,\le\, A_1$.

2.2.2 *Terms.* The term-validity judgment $\Gamma \vdash M \,:\, A$, defined in Figure 5 specifies that the term M is well formed in context Γ with classifying type A. Rules (16)–(22) are the usual rules for a a dependently-typed λ-calculus with pairing, projections, and a base type.

Rule (23) is the obvious introduction form for singletons. Rules (24) and (25) are somewhat less familiar, but analogous rules often appear in literature studying Standard ML modules, including the nonstandard structure-typing rule of Harper et al. [1990], the VALUE rules of Harper and Lillibridge's translucent sums [Harper and Lillibridge 1994], the strengthening operation of Leroy's manifest type system [Leroy 1994], the "self" rule of Leroy's applicative functors [Leroy 1995], and the REFL rule of Aspinall [2000]. The two rules can be justified as reflexive instances of extensionality (Rules (35) and (36)) and ensure that a term has every type that its η-expansion does. In most dependently-typed calculi these rules are admissible. However, in $\lambda_{\leq}^{\Pi\Sigma S}$, they allow terms to be given strictly more precise types.

For example, assume that $x : b \times b$. In the absence of Rule (24), the most precise (and only) type of x is $b \times b$. Using Rule (24) though, we can show

$$x : b \times b \vdash x \; : \; \mathcal{S}(\pi_1 x) \times \mathcal{S}(\pi_2 x).$$

That is, x has "the type of pairs whose first component is equal to the first component of x and whose second component is equal to the second component of x". This type is much more precise and informative $b \times b$, and it is entirely reasonable that x itself ought to satisfy that type. (Extensionality will further ensure that x is the only *only* pair with this type.) Rules (24) and (25) are critical for encoding singleton types for arbitrary terms. The rules can be viewed as extending singleton introduction to higher types.

We conjecture that the lower two premises in Rule (25) could be replaced by the much simpler side-condition $x \notin \mathrm{FV}(M)$, but we are then unable to prove the existence of most-specific types (see Section 3.4). The formulation here makes explicit that Rule (25) yields more-precise Π types for terms only by making the codomain more precise, rather than by weakening the domain type.

The final term well-formedness rule is the standard subsumption rule, Rule (26).

Term equivalence, defined in Figure 6, provides a notion of equality (interchangeability) for terms. The judgment $\Gamma \vdash M_1 \equiv M_2 \; : \; A$ says that M_1 and M_2 are equivalent terms of type A under context Γ. Equivalence is highly context-sensitive, as whether $\Gamma \vdash M_1 \equiv M_2 \; : \; A$ is provable depends not only on M_1 and M_2, but also on the types in Γ and on the type A at which the terms are compared.

Equivalence is first defined to be a reflexive, symmetric, and transitive relation (Rules (27)–(29)) and a congruence (Rules (30)–(34)).

There are two extensionality rules, Rule (35) and (36). If two functions or two pairs cannot be distinguished by their uses, then they are considered equivalent. In particular, two pairs are equivalent if they have equivalent first and second components, and two functions are equivalent if they return equivalent results for all arguments. If Rule (25) were simplified as discussed above then the last two premises of these extensionality rules could be replaced with the side condition $x \notin (\mathrm{FV}(M_1) \cup \mathrm{FV}(M_2))$.

We also have subsumption for equivalence, Rule (37), paralleling Rule (26).

Term Equivalence

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M \equiv M : A} \quad (27)$$

$$\frac{\Gamma \vdash M_2 \equiv M_1 : A}{\Gamma \vdash M_1 \equiv M_2 : A} \quad (28)$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A \qquad \Gamma \vdash M_2 \equiv M_3 : A}{\Gamma \vdash M_1 \equiv M_3 : A} \quad (29)$$

$$\frac{\Gamma \vdash A_1' \equiv A_2' \qquad \Gamma, x : A_1' \vdash M_1 \equiv M_2 : A''}{\Gamma \vdash \lambda x{:}A_1'.\, M_1 \equiv \lambda x{:}A_2'.\, M_2 : \Pi x{:}A_1'.\, A''} \quad (30)$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : \Pi x{:}A'.\, A'' \qquad \Gamma \vdash M_1' \equiv M_2' : A'}{\Gamma \vdash M_1\, M_1' \equiv M_2\, M_2' : [M_1'/x]A''} \quad (31)$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : \Sigma x{:}A'.\, A''}{\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : A'} \quad (32)$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : \Sigma x{:}A'.\, A''}{\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''} \quad (33)$$

$$\frac{\begin{array}{c}\Gamma \vdash \Sigma x{:}A'.\, A'' \\ \Gamma \vdash M_1' \equiv M_2' : A' \\ \Gamma \vdash M_1'' \equiv M_2'' : [M_1'/x]A''\end{array}}{\Gamma \vdash \langle M_1', M_1''\rangle \equiv \langle M_2', M_2''\rangle : \Sigma x{:}A'.\, A''} \quad (34)$$

$$\frac{\begin{array}{c}\Gamma \vdash \Sigma x{:}A'.\, A'' \\ \Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : A' \\ \Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''\end{array}}{\Gamma \vdash M_1 \equiv M_2 : \Sigma x{:}A'.\, A''} \quad (35)$$

$$\frac{\begin{array}{c}\Gamma, x : A' \vdash M_1\, x \equiv M_2\, x : A'' \\ \Gamma \vdash M_1 : \Pi x{:}A'.\, B_1'' \qquad \Gamma \vdash M_2 : \Pi x{:}A'.\, B_2''\end{array}}{\Gamma \vdash M_1 \equiv M_2 : \Pi x{:}A'.\, A''} \quad (36)$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A_1 \qquad \Gamma \vdash A_1 \leq A_2}{\Gamma \vdash M_1 \equiv M_2 : A_2} \quad (37)$$

$$\frac{\Gamma \vdash M : \mathcal{S}(N)}{\Gamma \vdash M \equiv N : \mathcal{S}(N)} \quad (38)$$

Fig. 6. Judgments of $\lambda_{\leq}^{\Pi\Sigma S}$, continued.

Interestingly, an easy inductive argument shows that the rules given so far merely define term equivalence to be syntactic identity up to renaming of bound variables. However, adding Rule (38), the elimination rule for singleton types, makes equivalence nontrivial and justifies the presence of each of the above rules.

This completes the definition of term equivalence. It may be initially surprising that there are no built-in rules for reducing function applications or projections from pairs (i.e., β-like rules). It turns out that these are admissible in the presence of singleton types and Rule (38). Full details are in Section 2.3 and Section 3.3, but we sketch one example here. It is clear that

$$\vdash \langle c_1, c_2\rangle : \mathcal{S}(c_1) \times \mathcal{S}(c_2).$$

Then, by Rule (21), it follows

$$\vdash \pi_1\langle c_1, c_2\rangle \,:\, \mathcal{S}(c_1)$$

and by Rule (38) and subsumption since $\mathcal{S}(c_1) \preceq b$ we have

$$\vdash \pi_1\langle c_1, c_2\rangle \equiv c_1 \,:\, b.$$

This same argument can be generalized to projections from arbitrary pairs, and in an analogous fashion to applications of λ-abstractions.

Given the β-rules, then, the extensionality Rules (35) and (36) imply that the usual η-rules are admissible as well. It is well known that η-reduction is not confluent in the presence of terminal (unit) types. As unit is a special case of a singleton type, the same behavior appears here as well. For example:

$$x : b \to \mathcal{S}(c_1) \vdash x \equiv \lambda y{:}b.\, c_1 \,:\, b \to b$$

holds, as does

$$x : \mathcal{S}(c_1) \to b \vdash x \equiv \lambda y{:}\mathcal{S}(c_1).\, (x\, c_1) \,:\, \mathcal{S}(c_1) \to b.$$

All the terms in these judgments are normal with respect to $\beta\eta$-reduction; compare the right-hand term in the last judgment with $\lambda y{:}\mathcal{S}(c_1).\,(xy)$, the η-expansion of x.

A more obvious consequence of having singletons—and their original motivation—is that they can be used to express definitions for variables. For example, in the following judgment the context effectively defines x to be c_1:

$$x : \mathcal{S}(c_1) \vdash \langle x, c_1\rangle \equiv \langle c_1, x\rangle \,:\, b \times b.$$

But the system is not restricted merely to giving simple definitions to variables. In the provable judgment

$$x \,:\, b \times \mathcal{S}(c_1) \vdash \pi_2 x \equiv c_1 \,:\, b.$$

the context only partially defines x; it is known to be a pair and its second component is (equivalent to) c_1, but this does not give a definition for x as a whole. Alternatively, this could be thought of as giving $\pi_2 x$ the definition c_1 without giving a definition for $\pi_1 x$.

Similarly, in the two provable judgments

$$\begin{aligned} x &: (\Sigma y{:}b.\, \mathcal{S}(y)) \vdash \pi_1 x \equiv \pi_2 x \,:\, b \\ x &: (\Sigma y{:}b.\, \mathcal{S}(y)) \vdash x \equiv \langle \pi_1 x, \pi_1 x\rangle \,:\, b \times b. \end{aligned}$$

the assumption governing x requires that it be a pair whose first component y has type b and whose second component is equal to the first; that is, a pair with two equal components of type b. This gives a definition only to $\pi_2 x$, namely as being equal to $\pi_1 x$.

Now because of subtyping and subsumption, terms do not have unique types. The equational system presented here has the relatively unusual property (for a decidable system) that equivalence of two terms depends on the type at which they are compared. Two terms may be equivalent at one type but not at another; for example, one *cannot* prove

$$\vdash \lambda x{:}b.\, x \equiv \lambda x{:}b.\, c_1 \;:\; b \to b.$$

However, by subsumption these two functions both have type $\mathcal{S}(c_1) \to b$ and the judgment

$$\vdash \lambda x{:}b.\, x \equiv \lambda x{:}b.\, c_1 \;:\; \mathcal{S}(c_1) \to b$$

is provable; the proof uses extensionality and the fact that the two functions provably agree on all arguments of type $\mathcal{S}(c_1)$, that is, when applied to only the argument c_1.

The classifying type at which terms are compared may depend on the context of their occurrence. For example, it follows immediately from the previous equation that

$$y \;:\; (\mathcal{S}(c_1) \to b) \to b \vdash y\,(\lambda x{:}b.\, x) \equiv y\,(\lambda x{:}b.\, c_1) \;:\; b$$

is also provable. The type of y guarantees that it will apply its argument only to the term c_1, so it cannot matter whether y is given $\lambda x{:}b.\, x$ or $\lambda x{:}b.\, c_1$. In contrast, the judgment

$$y \;:\; (b \to b) \to b \vdash y\,(\lambda x{:}b.\, x) \equiv y\,(\lambda x{:}b.\, c_1) \;:\; b$$

is *not* provable because the context makes a weaker assumption about y.

2.3 Admissible Rules and Labeled Singletons

We next turn to a number of interesting and useful rules which are admissible in our system, shown in Figure 7. Rules (39)–(41) are variant introduction and elimination rules for singleton types.

Next, in $\lambda^{\Pi\Sigma S}_{\leq}$ the type $\mathcal{S}(M)$ is well formed if and only if M is of the base type b. This initially seems restrictive, as one might expect to find *labeled singleton types* of the form $\mathcal{S}_A(M)$ representing the type of all terms equivalent to M when compared at type A. These would be necessary, for example, to model definitions of term-level functions. However, such labeled singletons are already definable within $\lambda^{\Pi\Sigma S}_{\leq}$.

One possible definition,[1] defined by induction on the size of the type label, is

$$\begin{array}{lcll}
\mathcal{S}_b(M) & := & \mathcal{S}(M) & \\
\mathcal{S}_{\mathcal{S}(M')}(M) & := & \mathcal{S}(M) & \\
\mathcal{S}_{\Pi x:A_1.\,A_2}(M) & := & \Pi x{:}A_1.\,(\mathcal{S}_{A_2}(M\,x)) & \text{(where } x \notin \mathrm{FV}(M)\text{).} \\
\mathcal{S}_{\Sigma x:A_1.\,A_2}(M) & := & \mathcal{S}_{A_1}(\pi_1 M) \times \mathcal{S}_{[\pi_1 M/x]A_2}(\pi_2 M) &
\end{array}$$

For example, if y has type $b \to b$, then $\mathcal{S}_{b\to b}(y)$ is $\Pi x{:}b.\,\mathcal{S}(y\,x)$. This can be interpreted as "the type of all functions that, when applied, yield the same answer as y does", or "the type of all functions that agree pointwise with y". By extensionality, any such function is provably equivalent to y. Rule (25) is used to prove that y has this type.

[1]Since types only matter up to equivalence, these definitions are not unique. One could equally well define $\mathcal{S}_{\mathcal{S}(M')}(M)$ to be $\mathcal{S}(M')$, or define $\mathcal{S}_{\Sigma x:A_1.\,A_2}(M)$ to be $\Sigma x{:}\mathcal{S}_{A_1}(\pi_1 M).\,\mathcal{S}_{A_2}(\pi_2 M)$.

Rules for Singleton Introduction and Elimination

$$\frac{\Gamma \vdash M_1 \equiv M_2 : b}{\Gamma \vdash M_1 : \mathcal{S}(M_2)} \tag{39}$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : b}{\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}(M_2)} \tag{40}$$

$$\frac{\Gamma \vdash M_1 : \mathcal{S}(M_2)}{\Gamma \vdash M_1 \equiv M_2 : b} \tag{41}$$

Rules for Labeled Singletons

$$\frac{\Gamma \vdash M_2 : A \qquad \Gamma \vdash M_1 : \mathcal{S}_A(M_2)}{\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}_A(M_2)} \tag{42}$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A}{\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}_A(M_2)} \tag{43}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathcal{S}_A(M) \leq A} \tag{44}$$

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A_1 \qquad \Gamma \vdash A_1 \leq A_2}{\Gamma \vdash \mathcal{S}_{A_1}(M_1) \leq \mathcal{S}_{A_2}(M_2)} \tag{45}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathcal{S}_A(M)} \tag{46}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M : \mathcal{S}_A(M)} \tag{47}$$

β Rules

$$\frac{\Gamma, x : A' \vdash M : A'' \qquad \Gamma \vdash M' : A'}{\Gamma \vdash (\lambda x{:}A'.\, M)\, M' \equiv [M'/x]M : [M'/x]A''} \tag{48}$$

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \pi_1\langle M_1, M_2\rangle \equiv M_1 : A_1} \tag{49}$$

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \pi_2\langle M_1, M_2\rangle \equiv M_2 : A_2} \tag{50}$$

η Rules

$$\frac{\Gamma \vdash M : \Pi x{:}A'.\, A''}{\Gamma \vdash M \equiv \lambda x{:}A'.\,(M\, x) : \Pi x{:}A'.\, A''} \tag{51}$$

$$\frac{\Gamma \vdash M : \Sigma x{:}A'.\, A''}{\Gamma \vdash M \equiv \langle \pi_1 M, \pi_2 M\rangle : \Sigma x{:}A'.\, A''} \tag{52}$$

Fig. 7. Admissible rules.

Rules (42)–(47) are admissible, showing that the labeled singleton types do behave appropriately; proofs are deferred to Section 3.3.

Because these labeled singletons are defined rather than primitive, one must be careful to note that $\Gamma \vdash \mathcal{S}_A(M)$ does not imply $\Gamma \vdash M : A$. For example, if c_1 and c_2 are distinct constants, then, according to our definition, we have $\mathcal{S}_{\mathcal{S}(c_2)}(c_1) = \mathcal{S}(c_1)$, and therefore $\vdash \mathcal{S}_{\mathcal{S}(c_2)}(c_1)$ even though c_1 cannot be shown to have type $\mathcal{S}(c_2)$. This explains the premise $\Gamma \vdash M_2 : A$ in Rule (42).

Finally, as previously mentioned the β and η rules for pairs and functions are admissible, Rules (48)–(52).

3. DECLARATIVE PROPERTIES

In this section, we present several basic properties of the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus. From these, we derive the definability of generalized singleton types, and the admissibility of the rules given in Section 2.3, before moving on to the much more interesting results of Section 4.

3.1 Preliminaries

We start with a number of very simple properties, each of which follows easily by induction on derivations.

If we define typing-context-free judgment forms $\mathcal{J}$:

$$\mathcal{J} ::= \text{ok} \mid A \mid A_1 \leq A_2 \mid A_1 \equiv A_2 \mid M : A \mid M_1 \equiv M_2 : A,$$

then given a context Γ one can construct a $\lambda_{\leq}^{\Pi\Sigma S}$ judgment $\Gamma \vdash \mathcal{J}$. The substitution $\gamma\mathcal{J}$ is defined by applying the substitution to the individual types and terms appearing in $\mathcal{J}$, while the free variable computation FV($\mathcal{J}$) is similarly defined as the union of the free variables of the phrases in $\mathcal{J}$.

The first results all follow by induction on derivations.

PROPOSITION 3.1 (SUBDERIVATIONS)

(1) *Every proof of* $\Gamma \vdash \mathcal{J}$ *contains a subderivation* $\Gamma \vdash ok$.
(2) *Every proof of* $\Gamma_1, x{:}A, \Gamma_2 \vdash \mathcal{J}$ *contains a strict subderivation* $\Gamma_1 \vdash A$.
(3) *If* $\Gamma \vdash M\, M' : B$, *then there is a strict subderivation of the form* $\Gamma \vdash M : A$ *for some type* A.
(4) *If* $\Gamma \vdash \pi_i M : B$, *then there is a strict subderivation of the form* $\Gamma \vdash M : A$ *for some type* A.

PROPOSITION 3.2. *If* $\Gamma \vdash \mathcal{J}$, *then* $FV(\mathcal{J}) \subseteq \text{dom}(\Gamma)$.

PROPOSITION 3.3 (REFLEXIVITY)

(1) *If* $\Gamma \vdash A$, *then* $\Gamma \vdash A \equiv A$.
(2) *If* $\Gamma \vdash A$, *then* $\Gamma \vdash A \leq A$.

PROPOSITION 3.4 (RENAMING). *If* $\Gamma_1, x{:}A, \Gamma_2 \vdash \mathcal{J}$, *then there is a derivation, of equal size and structure, of* $\Gamma_1, x'{:}A, [x'/x]\Gamma_2 \vdash [x'/x]\mathcal{J}$ *for every* $x' \notin \text{dom}(\Gamma_1) \cup \text{dom}(\Gamma_2)$.

PROPOSITION 3.5 (WEAKENING)

(1) *If* $\Gamma_1 \vdash \mathcal{J}$ *and* $\Gamma_1 \subseteq \Gamma_2$ *and* $\Gamma_2 \vdash ok$, *then* $\Gamma_2 \vdash \mathcal{J}$.
(2) *If* $\Gamma_1, x : A_2, \Gamma_2 \vdash \mathcal{J}$ *and* $\Gamma_1 \vdash A_1 \leq A_2$ *and* $\Gamma_1 \vdash A_1$, *then* $\Gamma_1, x : A_1, \Gamma_2 \vdash \mathcal{J}$.

Later we show that the assumption $\Gamma_1 \vdash A_1$ in the statement of Weakening is redundant, being already implied by $\Gamma_1 \vdash A_1 \leq A_2$.

Definition 3.6. The judgment $\theta \vdash \gamma : \Gamma$ holds if and only if the following conditions all hold:

(1) $\theta \vdash \mathrm{ok}$
(2) $\forall x \in \mathrm{dom}(\Gamma).\ \theta \vdash \gamma x : \gamma(\Gamma(x))$

PROPOSITION 3.7 (SUBSTITUTION)

(1) *If* $\Gamma \vdash \mathcal{J}$ *and* $\theta \vdash \gamma : \Gamma$*, then* $\theta \vdash \gamma(\mathcal{J})$*.*
(2) *If* $\Gamma_1, x : A, \Gamma_2 \vdash ok$ *and* $\Gamma_1 \vdash M : A$*, then* $\Gamma_1, [M/x]\Gamma_2 \vdash ok$*.*

3.2 Validity and Functionality

We next show two highly useful properties of the calculus. *Validity* is the property that any phrase appearing within a provable judgment is well formed (e.g., if $\Gamma \vdash M_1 \equiv M_2 : A$ then $\Gamma \vdash \mathrm{ok}$ and $\Gamma \vdash A$ and $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$). *Functionality* states that applying equivalent substitutions to phrases related by equivalence or subtyping yields similarly related phrases.

The rules have been structured to assume validity for premises and guarantee and preserve validity for conclusions. A simple proof, however, is hindered by the presence of dependencies in types. The direct approach by induction on derivations fails because of cases such as Rule (33):

$$\frac{\Gamma \vdash M_1 \equiv M_2 : \Sigma x{:}A'.\, A''}{\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''}.$$

Here we need to show $\Gamma \vdash \pi_2 M_2 : [\pi_1 M_1/x]A''$ but from the inductive hypothesis and Rule (22) we have only $\Gamma \vdash \pi_2 M_2 : [\pi_1 M_2/x]A''$. The desired result would follow if we knew that $\Gamma \vdash [\pi_1 M_2/x]A'' \leq [\pi_1 M_1/x]A''$. Since $\Gamma \vdash \pi_1 M_2 \equiv \pi_1 M_1 : A'$, the subtyping judgment required follows from functionality.

This suggests one should first prove functionality. The most general form of functionality cannot be directly proved in the absence of validity, but the proof does go through for the restricted case of equivalent substitutions being applied to a single phrase to obtain related results. This suffices to show validity, and together these allow a simple proof of general functionality.

Definition 3.8. The judgment $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$ holds if and only if the following conditions all hold:

(1) $\theta \vdash \gamma_1 : \Gamma$ and $\theta \vdash \gamma_2 : \Gamma$
(2) $\forall x \in \mathrm{dom}(\Gamma).\ \theta \vdash \gamma_1 x \equiv \gamma_2 x : \gamma_1(\Gamma(x))$

PROPOSITION 3.9 (SIMPLE FUNCTIONALITY)

(1) *If* $\Gamma \vdash ok$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$*, then* $\theta \vdash \gamma_2 \equiv \gamma_1 : \Gamma$*.*
(2) *If* $\Gamma \vdash A$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$*, then* $\theta \vdash \gamma_1 A \equiv \gamma_2 A$*.*
(3) *If* $\Gamma \vdash A$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$*, then* $\theta \vdash \gamma_1 A \leq \gamma_2 A$*.*
(4) *If* $\Gamma \vdash M : A$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$*, then* $\theta \vdash \gamma_1 M \equiv \gamma_2 M : \gamma_1 A$*.*

PROOF. By induction on the proofs of the first premise. □

PROPOSITION 3.10 (VALIDITY)

(1) *If* $\Gamma \vdash A_1 \le A_2$, *then* $\Gamma \vdash A_1$ *and* $\Gamma \vdash A_2$.
(2) *If* $\Gamma \vdash A_1 \equiv A_2$, *then* $\Gamma \vdash A_1$, $\Gamma \vdash A_2$, $\Gamma \vdash A_1 \le A_2$, *and* $\Gamma \vdash A_2 \le A_1$
(3) *If* $\Gamma \vdash M : A$, *then* $\Gamma \vdash A$.
(4) *If* $\Gamma \vdash M_1 \equiv M_2 : A$, *then* $\Gamma \vdash M_1 : A$, $\Gamma \vdash M_2 : A$, *and* $\Gamma \vdash A$.

PROOF. By induction on derivations. The only interesting cases are equivalence rules which treat the left and right sides assymetrically. The assymetric choice of substitutions in Rule (31) (where conclusion substitutes M_1' for x rather than the provably equivalent term M_2') and Rule (33) are handled by Proposition 3.9. The subtyping conclusions in Part (2) are a strengthening of the inductive hypothesis to ensure that the cases for Rules (14), (15), and (30) (which all put A_1' in the typing context rather than A_2') go through. □

PROPOSITION 3.11 (ANTISYMMETRY OF SUBTYPING). $\Gamma \vdash A_1 \le A_2$ *and* $\Gamma \vdash A_2 \le A_1$ *if and only if* $\Gamma \vdash A_1 \equiv A_2$.

PROOF. The "if" direction was proved in Part (2). The "only if" direction follows by induction on $size(A_1) + size(A_2)$. □

PROPOSITION 3.12 (SYMMETRY AND TRANSITIVITY OF TYPE EQUIVALENCE).

(1) *If* $\Gamma \vdash A_1 \equiv A_2$, *then* $\Gamma \vdash A_2 \equiv A_1$
(2) *If* $\Gamma \vdash A_1 \equiv A_2$, *and* $\Gamma \vdash A_2 \equiv A_3$ *then* $\Gamma \vdash A_1 \equiv A_3$.

PROOF. By induction on derivations. □

PROPOSITION 3.13 (TRANSITIVITY OF SUBTYPING). *If* $\Gamma \vdash A_1 \le A_2$ *and* $\Gamma \vdash A_2 \le A_3$, *then* $\Gamma \vdash A_1 \le A_3$.

PROOF. By induction on $size(A_1) + size(A_2) + size(A_3)$ □

PROPOSITION 3.14 (FULL FUNCTIONALITY)

(1) *If* $\Gamma \vdash M_1 \equiv M_2 : A$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$, *then* $\theta \vdash \gamma_1 M_1 \equiv \gamma_2 M_2 : \gamma_1 A$.
(2) *If* $\Gamma \vdash A_1 \equiv A_2$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$, *then* $\theta \vdash \gamma_1 A_1 \equiv \gamma_2 A_2$.
(3) *If* $\Gamma \vdash A_1 \le A_2$ *and* $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$, *then* $\theta \vdash \gamma_1 A_1 \le \gamma_2 A_2$.

PROOF. We show the proof for just the first part; the last two parts follow similarly. Assume $\Gamma \vdash M_1 \equiv M_2 : A$ and $\theta \vdash \gamma_1 \equiv \gamma_2 : \Gamma$. By Proposition 3.7, $\theta \vdash \gamma_1 M_1 \equiv \gamma_1 M_2 : \gamma_1 A$. By Proposition 3.10, we have $\Gamma \vdash M_2 : A$, and so by Proposition 3.9, $\theta \vdash \gamma_1 M_2 \equiv \gamma_2 M_2 : \gamma_1 A$. By Proposition 3.12, $\theta \vdash \gamma_1 M_1 \equiv \gamma_2 M_2 : \gamma_1 A$. □

3.3 Proofs of Admissibility

We now have enough technical machinery to prove the admissibility of Rules (39)–(52).

LEMMA 3.15. $\gamma(\mathcal{S}_A(M)) = \mathcal{S}_{\gamma A}(\gamma M)$.

PROPOSITION 3.16. *The rules from Section* 2.3 *are all admissible*

PROOF.

—Rules (39)–(41). By Proposition 3.10, and subsumption.
—Rule (42).

$$\frac{\Gamma \vdash M_2 : A \qquad \Gamma \vdash M_1 : \mathcal{S}_A(M_2)}{\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}_A(M_2)}$$

By induction on the size of A.

—Case: $A = b$ and $\mathcal{S}_A(M_2) = \mathcal{S}(M_2)$. By Rule (38), $\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}(M_2)$.

—Case: $A = \mathcal{S}(N)$ and $\mathcal{S}_A(M_2) = \mathcal{S}(M_2)$. By Rule (38), $\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}(M_2)$.

—Case: $A = \Pi x{:}A'.\, A''$ and $\mathcal{S}_A(M_2) = \Pi x{:}A'.\, \mathcal{S}_{A''}(M_2\, x)$. By Rule (19) and Proposition 3.5 and Lemma 3.15 we have $\Gamma, x : A' \vdash M_1\, x : \mathcal{S}_{A''}(M_2\, x)$ and $\Gamma, x : A' \vdash M_2\, x : A''$. By the inductive hypothesis, $\Gamma, x : A' \vdash M_1\, x \equiv M_2\, x : \mathcal{S}_{A''}(M_2\, x)$. Therefore, by Rule (36), we have $\Gamma \vdash M_1 \equiv M_2 : \Pi x{:}A'.\, \mathcal{S}_{A''}(M_2\, x)$.

—Case: $A = \Sigma x{:}A'.\, A$ and $\mathcal{S}_A(M_2) = (\mathcal{S}_{A'}(\pi_1 M_2)) \times (\mathcal{S}_{[\pi_1 M_2/x]A''}(\pi_2 M_2))$. Then $\Gamma \vdash \pi_1 M_1 : \mathcal{S}_{A'}(\pi_1 M_2)$ and $\Gamma \vdash \pi_2 M_1 : \mathcal{S}_{[\pi_1 M_1/x]A''}(\pi_2 M_2)$. $\Gamma \vdash \pi_1 M_2 : A'$ and $\Gamma \vdash \pi_2 M_2 : [\pi_1 M_2/x]A'$, so by the inductive hypothesis, $\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : \mathcal{S}_{A'}(\pi_1 M_2)$ and $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : \mathcal{S}_{[\pi_1 M_1/x]A''}(\pi_2 M_2)$. By Proposition 3.10, and Rule (35) we have $\Gamma \vdash M_1 \equiv M_2 : (\mathcal{S}_{A'}(\pi_1 M_2)) \times (\mathcal{S}_{[\pi_1 M_2/x]A''}(\pi_2 M_2))$.

—Rule (43).

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A}{\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}_A(M_2)}$$

By induction on the size of A.

—Case: $A = b$ and $\mathcal{S}_A(M_2) = \mathcal{S}(M_2)$. $\Gamma \vdash M_1 : \mathcal{S}(M_2)$ by Rule (39), so $\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}(M_2)$ by Rule (38).

—Case: $A = \mathcal{S}(N)$ and $\mathcal{S}_A(M_2) = \mathcal{S}(M_2)$. $\Gamma \vdash N : b$ by Proposition 3.10, and inversion of Rule (4), so $\Gamma \vdash \mathcal{S}(N) \le b$. Then, $\Gamma \vdash M_1 \equiv M_2 : b$ by subsumption, so $\Gamma \vdash M_1 : \mathcal{S}(M_2)$ by Rule (39). Thus, $\Gamma \vdash M_1 \equiv M_2 : \mathcal{S}(M_2)$ by Rule (38).

—Case: $A = \Pi x{:}A'.\, A''$ and $\mathcal{S}_A(M_2) = \Pi x{:}A'.\, \mathcal{S}_{A''}(M_2\, x)$. By Proposition 3.5, and Rule (31), $\Gamma, x : A' \vdash M_1\, x \equiv M_2\, x : A''$. By the inductive hypothesis, $\Gamma, x : A' \vdash M_1\, x \equiv M_2\, x : \mathcal{S}_{A''}(M_2\, x)$. By Proposition 3.10, we have $\Gamma \vdash M_1 : \Pi x{:}A'.\, A''$ and $\Gamma \vdash M_2 : \Pi x{:}A'.\, A''$. Therefore, by Rule (36), $\Gamma \vdash M_1 \equiv M_2 : \Pi x{:}A'.\, \mathcal{S}_{A''}(M_2\, x)$.

—Case: $A = \Sigma x{:}A'.\, A''$ and $\mathcal{S}_A(M_2) = (\mathcal{S}_{A'}(\pi_1 M_2)) \times (\mathcal{S}_{[\pi_1 M_2/x]A''}(\pi_2 M_2))$. Then $\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : A'$ and $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''$. By Proposition 3.9, and Rule (37), $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_2/x]A''$. By the inductive hypothesis, $\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : \mathcal{S}_{A'}(\pi_1 M_2)$ and $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : \mathcal{S}_{[\pi_1 M_2/x]A''}(\pi_2 M_2)$. (Note that *size*$([\pi_1 M_2/x]A'')$

$= size(A'') < size(A)$.) Therefore by Proposition 3.10, and Rule (35), we have $\Gamma \vdash M_1 \equiv M_2 : (\mathcal{S}_{A'}(\pi_1 M_2)) \times (\mathcal{S}_{[\pi_1 M_2/x]A''}(\pi_2 M_2))$.

—Rule (44).

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathcal{S}_A(M) \le A}$$

By induction on the size of A

—Case: $A = b$ and $\mathcal{S}_A(M) = \mathcal{S}(M)$. By Rule (7), we have $\Gamma \vdash \mathcal{S}(M) \le b$.

—Case: $A = \mathcal{S}(N)$ and $\mathcal{S}_A(M) = \mathcal{S}(M)$. Then, $\Gamma \vdash M \equiv N : b$ so $\Gamma \vdash \mathcal{S}(M) \le \mathcal{S}(N)$.

—Case: $A = \Pi x{:}A_1.\, A_2$ and $\mathcal{S}_A(M) = \Pi x{:}A_1.\, \mathcal{S}_{A_2}(M\, x)$. Then $\Gamma \vdash A_1$ and $\Gamma, x : A_1 \vdash M\, x : A_2$. By the inductive hypothesis, $\Gamma, x : A_1 \vdash \mathcal{S}_{A_2}(M\, x) \le A_2$. Therefore, $\Gamma \vdash \Pi x{:}A_1.\, \mathcal{S}_{A_2}(M\, x) \le \Pi x{:}A_1.\, A_2$.

—Case: $A = \Sigma x{:}A'.\, A''$ and $\mathcal{S}_A(M) = (\mathcal{S}_{A'}(\pi_1 M)) \times (\mathcal{S}_{[\pi_1 M/x]A''}(\pi_2 M))$. By Proposition 3.10, and inversion of Rule (6) we have $\Gamma, x{:}A' \vdash A''$. Then $\Gamma \vdash \pi_1 M : A'$ so by the inductive hypothesis, $\Gamma \vdash \mathcal{S}_{A'}(\pi_1 M) \le A'$. Furthermore, $\Gamma \vdash \pi_2 M : [\pi_1 M/x]A''$. By the inductive hypothesis, $\Gamma \vdash \mathcal{S}_{[\pi_1 M/x]A''}(\pi_2 M) \le [\pi_1 M/x]A''$. Also, by Proposition 3.1, and Proposition 3.5, $\Gamma, x : \mathcal{S}_{A'}(\pi_1 M) \vdash A'' \le A''$. By Rule (42), we have $\Gamma, x : \mathcal{S}_{A'}(\pi_1 M) \vdash x \equiv \pi_1 M : \mathcal{S}_{A'}(\pi_1 M)$ so by Functionality, we have $\Gamma, x : \mathcal{S}_{A'}(\pi_1 M) \vdash [\pi_1 M/x]A'' \le A''$. Therefore, using Proposition 3.13, $\Gamma \vdash (\mathcal{S}_{A'}(\pi_1 M)) \times (\mathcal{S}_{[\pi_1 M/x]A''}(\pi_2 M)) \le \Sigma x{:}A'.\, A''$.

—Rule (45).

$$\frac{\Gamma \vdash M_1 \equiv M_2 : A_1 \qquad \Gamma \vdash A_1 \le A_2}{\Gamma \vdash \mathcal{S}_{A_1}(M_1) \le \mathcal{S}_{A_2}(M_2)}$$

By induction on $size(A_1)$.

—Case: $A_1 = b$ or $\mathcal{S}(M_1)$ and $A_2 = b$ or $\mathcal{S}(M_2)$. Then $\mathcal{S}_{A_1}(M_1) = \mathcal{S}(M_1)$, $\mathcal{S}_{A_2}(M_2) = \mathcal{S}(M_2)$, and the desired conclusion follows by Rule (8).

—Case: $A_1 = \Pi x{:}A'_1.\, A''_1$ and $A_2 = \Pi x{:}A'_2.\, A''_2$. $\mathcal{S}_{A_i}(M_i) = \Pi x{:}A'_i.\, \mathcal{S}_{A''_i}(M_i\, x)$. By inversion $\Gamma \vdash A'_2 \le A'_1$ and $\Gamma, x : A'_2 \vdash A''_1 \le A''_2$. Now $\Gamma, x : A'_2 \vdash M_1\, x \equiv M_2\, x : A''_1$. By the inductive hypothesis, $\Gamma, x : A'_2 \vdash \mathcal{S}_{A''_1}(M_1\, x) \le \mathcal{S}_{A''_2}(M_2\, x)$. The conclusion follows by Rule (10).

—Case: $A_1 = \Sigma x{:}A'_1.\, A''_1$ and $A_2 = \Sigma x{:}A'_2.\, A''_2$. Then $\mathcal{S}_{A_1}(M_1) = \Sigma x : \mathcal{S}_{A'_1}(\pi_1 M_1).\mathcal{S}_{[\pi_1 M_1/x]A''_1}(\pi_2 M_1)$ and $\mathcal{S}_{A_2}(M_2) = \Sigma x{:}\mathcal{S}_{A'_2}(\pi_1 M_2).\, \mathcal{S}_{[\pi_1 M_2/x]A''_2}(\pi_2 M_2)$. Now $\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : A'_1$ and $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''_1$. By the inductive hypothesis, $\Gamma \vdash \mathcal{S}_{A'_1}(\pi_1 M_1) \le \mathcal{S}_{A'_2}(\pi_1 M_2)$. Since $\Gamma \vdash [\pi_1 M_1/x]A''_1 \le [\pi_1 M_2/x]A''_2$, the inductive hypothesis applies, yielding $\Gamma \vdash \mathcal{S}_{[\pi_1 M_1/x]A''_1}(\pi_2 M_1) \le \mathcal{S}_{[\pi_1 M_2/x]A''_2}(\pi_2 M_2)$. (Here it is important that the induction is on the size of A_1 and not by induction on the proof $\Gamma \vdash A_1 \le A_2$.) The desired result follows by Proposition 3.5, and Rule (11).

—Rules (46) and (47).

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathcal{S}_A(M)} \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \mathcal{S}_A(M)}$$

$$
\begin{array}{ll}
\Gamma \vdash c \Uparrow \mathcal{S}(c) & \\
\Gamma \vdash x \Uparrow \mathcal{S}_{\Gamma(x)}(x) & \\
\Gamma \vdash \lambda x{:}A'.\, M \Uparrow \Pi x{:}A'.\, A'' & \text{if } \Gamma \vdash A' \text{ and } x \notin \mathrm{dom}(\Gamma) \text{ and } \Gamma, x : A' \vdash M \Uparrow A'' \\
\Gamma \vdash M\, M' \Uparrow [M'/x]A'' & \text{if } \Gamma \vdash M \Uparrow \Pi x{:}A'.\, A'' \text{ and } \Gamma \vdash M' \Uparrow A'_1 \text{ and } \Gamma \vdash A'_1 \leq A' \\
\Gamma \vdash \langle M', M'' \rangle \Uparrow A' \times A'' & \text{if } \Gamma \vdash M' \Uparrow A' \text{ and } \Gamma \vdash M'' \Uparrow A''. \\
\Gamma \vdash \pi_1 M \Uparrow A' & \text{if } \Gamma \vdash M \Uparrow A' \times A'' \\
\Gamma \vdash \pi_2 M \Uparrow A'' & \text{if } \Gamma \vdash M \Uparrow A' \times A''
\end{array}
$$

Fig. 8. Rules for principal types.

Assume $\Gamma \vdash M : A$. By Rule (27), $\Gamma \vdash M \equiv M : A$. By Rule (43), $\Gamma \vdash M \equiv M : \mathcal{S}_A(M)$. By Proposition 3.10, $\Gamma \vdash \mathcal{S}_A(M)$ and $\Gamma \vdash M : \mathcal{S}_A(M)$.

—Rule (48).

$$\frac{\Gamma, x : A' \vdash M : A'' \qquad \Gamma \vdash M' : A'}{\Gamma \vdash (\lambda x{:}A'.\, M)\, M' \equiv [M'/x]M : [M'/x]A''}$$

Assume $\Gamma, x : A_2 \vdash M : A$ and $\Gamma \vdash M_2 : A_2$. Then, $\Gamma, x : A_2 \vdash M : \mathcal{S}_A(M)$, so $\Gamma \vdash \lambda x{:}A_2.\, M : \Pi x{:}A_2.\, \mathcal{S}_A(M)$. By Rule (19) and Lemma 3.15 we have $\Gamma \vdash (\lambda x{:}A_2.\, M)\, M_2 : \mathcal{S}_{[M_2/x]A}([M_2/x]M)$. By Proposition 3.7, $\Gamma \vdash [M_2/x]M : [M_2/x]A$. Thus $\Gamma \vdash (\lambda x{:}A_2.\, M)\, M_2 \equiv [M_2/x]M : [M_2/x]A$ by Rule (42).

—Rule (49).

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \pi_1 \langle M_1, M_2 \rangle \equiv M_1 : A_1}$$

Assume $\Gamma \vdash M_1 : A_1$ and $\Gamma \vdash M_2 : A_2$. Then, $\Gamma \vdash M_1 : \mathcal{S}_{A_1}(M_1)$, so $\Gamma \vdash \langle M_1, M_2 \rangle : \mathcal{S}_{A_1}(M_1) \times A_2$. Thus, $\Gamma \vdash \pi_1 \langle M_1, M_2 \rangle : \mathcal{S}_{A_1}(M_1)$ and $\Gamma \vdash \pi_1 \langle M_1, M_2 \rangle \equiv M_1 : A_1$.

—Rule (50). Analogous to Rule (49).

—Rules (51)–(52). By the β-rules and extensionality. □

It follows that a variable with a labeled singleton type is interchangeable with the term appearing in the singleton:

Lemma 3.17.

(1) *If* $\Gamma \vdash M : A$ *and* $\Gamma, x : \mathcal{S}_A(M) \vdash N : B$, *then* $\Gamma, x : \mathcal{S}_A(M) \vdash N \equiv [M/x]N : B$.

(2) *If* $\Gamma \vdash M : A$ *and* $\Gamma, x : \mathcal{S}_A(M) \vdash B$, *then* $\Gamma, x : \mathcal{S}_A(M) \vdash B \equiv [M/x]B$.

Proof. By admissible Rule (42) and Proposition 3.9. □

3.4 Principal Types

It is useful to have an alternate, more syntax-directed characterization of the typing judgment that avoids the subsumption and singleton rules by directly computing most-specific types. (We use this in proving soundness for our algorithms.) The definition appears in Figure 8. Although there is one rule for each possible form of term, it does not define an algorithm because it still refers to the subtyping relation, which in turn is defined in terms of term equivalence and well formedness. A fully algorithmic version of well formedness appears later in Section 6, once we have a correct algorithm for term equivalence in hand.

We will also refer to a term's most-precise type as its *principal* type, since all other types for the term can be derived from the principal type by subsumption. Formally, A is principal for M in Γ if and only if $\Gamma \vdash M : A$ and whenever $\Gamma \vdash M : B$ we have $\Gamma \vdash A \leq B$. By the antisymmetry of subtyping, principal types are unique up to equivalence.

THEOREM 3.18 (PRINCIPAL TYPE SOUNDNESS). *If* $\Gamma \vdash ok$ *and* $\Gamma \vdash M \Uparrow B$, *then* $\Gamma \vdash M : B$.

PROOF. By induction on the proof of $\Gamma \vdash M \Uparrow B$. □

LEMMA 3.19 (PRINCIPAL TYPE WEAKENING AND DETERMINISM). *If* $\Gamma \vdash M \Uparrow A$ *and* $\Gamma' \supseteq \Gamma$, *then* $\Gamma' \vdash M \Uparrow B$ *if and only if* $A = B$.

THEOREM 3.20 (PRINCIPAL TYPE COMPLETENESS). *If* $\Gamma \vdash M : B$, *then there exists* A *(determined by* Γ *and* M*) such that* $\Gamma \vdash M \Uparrow A$ *and* $\Gamma \vdash A \leq \mathcal{S}_B(M)$ *(so that* $\Gamma \vdash A \leq B$*).*

PROOF. By induction on the proof of the assumption and cases on the last rule used. The idea of replacing the general subsumption rule with a single use of subtyping within applications is very common, so we show only a few cases involving rules specific to $\lambda_{\leq}^{\Pi\Sigma S}$.

—Case: Rule (23).

$$\frac{\Gamma \vdash M : b}{\Gamma \vdash M : \mathcal{S}(M)}.$$

By the inductive hypothesis, noting that $\mathcal{S}_{\mathcal{S}(M)}(M) = \mathcal{S}(M)$. It is important here that the induction hypothesis guarantees $A \leq \mathcal{S}_B(M)$ rather than just $A \leq B$.

—Case: Rule (24).

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

By the inductive hypothesis, $\Gamma \vdash \pi_1 M \Uparrow A'$and $\Gamma \vdash A' \leq \mathcal{S}_{B'}(\pi_1 M)$. Similarly, $\Gamma \vdash \pi_2 M \Uparrow A''$ and $\Gamma \vdash A'' \leq \mathcal{S}_{[\pi_1 M/x]B''}(\pi_2 M)$. By inversion of the principal type rules, and the observation that they cannot produce a dependent Σ type, it must be that $\Gamma \vdash M \Uparrow A' \times A''$. Since $\mathcal{S}_{\Sigma x:B'.\,B''}(M) = \mathcal{S}_{B'}(\pi_1 M) \times \mathcal{S}_{[\pi_1 M/x]B''}(\pi_2 M)$, by Rule (11) and Proposition 3.5 we have $\Gamma \vdash A' \times A'' \leq \mathcal{S}_{\Sigma x:B'.\,B''}(M)$.

—Case: Rule (25).

$$\frac{\Gamma, x : B' \vdash M\,x : B'' \quad \Gamma \vdash M : \Pi x{:}B'.\,B_2{''} \qquad \Gamma \vdash \Pi x{:}B'.\,B_2{''}}{\Gamma \vdash M : \Pi x{:}B'.\,B''}$$

By the inductive hypothesis, $\Gamma \vdash M \Uparrow A$, $\Gamma \vdash M : A$, and $\Gamma \vdash A \leq \mathcal{S}_{\Pi x:B'.\,B_2''}(M)$. Now $\mathcal{S}_{\Pi x:B'.\,B_2''}(M) = \Pi x{:}B'.\,\mathcal{S}_{B_2''}(M\,x)$ so by inversion of

Natural Types

$\Gamma \rhd c_i \uparrow b$	
$\Gamma \rhd x \uparrow \Gamma(x)$	
$\Gamma \rhd \pi_1 p \uparrow A'$	if $\Gamma \rhd p \uparrow \Sigma y{:}A'.\, A''$
$\Gamma \rhd \pi_2 p \uparrow [\pi_1 p/y]A''$	if $\Gamma \rhd p \uparrow \Sigma y{:}A'.\, A''$
$\Gamma \rhd p\, M \uparrow [M/y]A''$	if $\Gamma \rhd p \uparrow \Pi y{:}A'.\, A''$

Head Reduction

$\Gamma \rhd \mathcal{E}[(\lambda x{:}A.\, M)\, M'] \rightsquigarrow \mathcal{E}[[M'/x]M]$	
$\Gamma \rhd \mathcal{E}[\pi_1 \langle M_1, M_2 \rangle] \rightsquigarrow \mathcal{E}[M_1]$	
$\Gamma \rhd \mathcal{E}[\pi_2 \langle M_1, M_2 \rangle] \rightsquigarrow \mathcal{E}[M_2]$	
$\Gamma \rhd \mathcal{E}[p] \rightsquigarrow \mathcal{E}[N]$	if $\Gamma \rhd p \uparrow \mathcal{S}(N)$

Head Normalization

$\Gamma \rhd M \Downarrow N$	if $\Gamma \rhd M \rightsquigarrow M'$ and $\Gamma \rhd M' \Downarrow N$
$\Gamma \rhd M \Downarrow M$	otherwise

Term Normalization

$\Gamma \rhd M : b \Longrightarrow M''$	if $\Gamma \rhd M \Downarrow M'$ and $\Gamma \rhd M' \longrightarrow M'' \uparrow b$
$\Gamma \rhd M : \mathcal{S}(N) \Longrightarrow M''$	if $\Gamma \rhd M \Downarrow M'$ and $\Gamma \rhd M' \longrightarrow M'' \uparrow b$
$\Gamma \rhd M : \Pi x{:}A'.\, A'' \Longrightarrow \lambda x{:}B'.\, N$	if $\Gamma \rhd A' \Longrightarrow B'$ and $\Gamma, x : A' \rhd (M\, x) : A'' \Longrightarrow N$
$\Gamma \rhd M : \Sigma x{:}A'.\, A'' \Longrightarrow \langle N', N'' \rangle$	if $\Gamma \rhd \pi_1 M : A' \Longrightarrow N'$ and $\Gamma \rhd \pi_2 M : [\pi_1 M/x]A'' \Longrightarrow N''$.

Path Normalization

$\Gamma \rhd c \longrightarrow c \uparrow b$	
$\Gamma \rhd x \longrightarrow x \uparrow \Gamma(x)$	
$\Gamma \rhd p\, M \longrightarrow p'\, M' \uparrow [M/x]A''$	if $\Gamma \rhd p \longrightarrow p' \uparrow \Pi x{:}A'.\, A''$ and $\Gamma \rhd M : A' \Longrightarrow M'$
$\Gamma \rhd \pi_1 p \longrightarrow \pi_1 p' \uparrow A'$	if $\Gamma \rhd p \longrightarrow p' \uparrow \Sigma x{:}A'.\, A''$
$\Gamma \rhd \pi_2 p \longrightarrow \pi_2 p' \uparrow [\pi_1 p/x]A'$	if $\Gamma \rhd p \longrightarrow p' \uparrow \Sigma x{:}A'.\, A''$

Type Normalization

$\Gamma \rhd b \Longrightarrow b$	
$\Gamma \rhd \mathcal{S}(M) \Longrightarrow \mathcal{S}(M')$	if $\Gamma \rhd M : b \Longrightarrow M'$
$\Gamma \rhd \Pi x{:}A'.\, A'' \Longrightarrow \Pi x{:}B'.\, B''$	if $\Gamma \rhd A' \Longrightarrow B'$ and $\Gamma, x : A' \rhd A'' \Longrightarrow B''$
$\Gamma \rhd \Sigma x{:}A'.\, A'' \Longrightarrow \Sigma x{:}B'.\, B''$	if $\Gamma \rhd A' \Longrightarrow B'$ and $\Gamma, x : A' \rhd A'' \Longrightarrow B''$

Fig. 9. Normalization algorithm.

Rule (10), $A = \Pi x{:}A'.\, A''$ and $\Gamma \vdash B' \le A'$. Also by the inductive hypothesis, $\Gamma, x : B' \vdash M\, x \Uparrow A''_2$, $\Gamma, x : B' \vdash M\, x : A''_2$, and $\Gamma, x : B' \vdash A''_2 \le \mathcal{S}_{B''}(M\, x)$. But by Lemma 3.19, we have $A''_2 = [x/x]A'' = A''$. Now $\mathcal{S}_{\Pi x:B'.\, B''}(M) = \Pi x{:}B'.\, \mathcal{S}_{B''}(M\, x)$. Therefore, $\Gamma \vdash \Pi x{:}A'.\, A'' \le \mathcal{S}_{\Pi x:B'.\, B''}(M)$. □

4. NORMALIZATION OF TERMS AND TYPES

4.1 Introduction

Determining whether types and types are well formed is straightforward once we have a method for checking equivalence of well-formed terms. The fact that equivalence is sensitive both to the typing context and to the classifying type makes it difficult to use context-insensitive rewrite rules such as β-reduction. We therefore introduce a complete algorithm for computing the normal form of a term given a context and a type; two terms are provably equivalent if and only if they have the same normal form. (In Section 6, we show a more efficient method for determining type equivalence.)

4.2 Normalization Algorithm

The components of the normalization algorithm are defined in Figure 9. The algorithm uses the concepts of *paths* and *elimination contexts*. An elimination context is a series of applications to and projections from "$\diamond$", which is called the context's hole.

$$\begin{aligned} \mathcal{E} ::= \; & \diamond \\ \mid \; & \mathcal{E}\, M \\ \mid \; & \pi_1 \mathcal{E} \\ \mid \; & \pi_2 \mathcal{E} \end{aligned}$$

If $\mathcal{E}$ is such a context, then $\mathcal{E}[M]$ represents the term resulting by replacing the hole in $\mathcal{E}$ with M. If a term is of the form $\mathcal{E}[x]$ or $\mathcal{E}[c]$, then it is called a *path*, denoted by p. Note that $\mathcal{E}[p]$ will also be a path.

The definitions in Figure 9 are "algorithmic" inference rules; they have been carefully designed to be syntax-directed, so that proof search is deterministic and no backtracking is required. To distinguish algorithmic rules from the declarative rules of $\lambda^{\Pi\Sigma S}_{\leq}$, we use the symbol $\triangleright$ to separate the typing assumptions from the conclusion.

It seems reasonable to say that a variable $x : b$ has no definition, but that a variable $x : \mathcal{S}(c_1)$ has the definition c_1. Similarly, if $y : b \times \mathcal{S}(c_1)$ then y as a whole has no definition, nor does $\pi_1 y$, yet $\pi_2 y$ has the definition c_1. This intuition is formalized through the concept of a *natural type*. This is the most precise type that can be assigned with standard typing rules, ignoring the singleton-introducing Rules (23), (24), and (25), as these provide no "new" information about the value of a term.

The *natural type* algorithmic relation is written

$$\Gamma \triangleright p \uparrow A.$$

Given a well formed context Γ and a path p that is well formed in this context, the natural type algorithm attempts to determine a type for the path by taking the type of the head variable or constant and doing appropriate substitutions and projections. A path is said to *have a definition* if its natural type is a singleton type $\mathcal{S}(N)$; in this case N is said to be the definition of the path.

The natural type is not always the most-precise type. For example, $x : b \triangleright x \uparrow b$ although the principal type of x in this context would be $\mathcal{S}(x)$. We show later that $\mathcal{S}_A(p)$ is principal for p, if A is the natural type of p.

The *head reduction* relation

$$\Gamma \triangleright M \rightsquigarrow N$$

takes Γ and M and returns the result of applying one step of head reduction if M has such a redex. If the head of M is a path that has a definition then the definition is returned. Otherwise, there is no head reduct.

The *head normalization* relation

$$\Gamma \triangleright M \Downarrow N$$

takes Γ and M and repeatedly applies head reduction to M until a head normal form is found. Head reduction and head normalization are deterministic, since the head β-redex is always unique if one exists, and a path can yield at most one definition. Because head reduction includes expansion of definitions, it is possible to have paths—including single variables—that are not head normal.

It is easy to check that normalization is deterministic (up to renaming of bound variables) and satisfies a weakening property.

LEMMA 4.1 (DETERMINACY)

(1) *If* $\Gamma \rhd p \uparrow N_1$ *and* $\Gamma \rhd p \uparrow N_2$, *then* $N_1 = N_2$.
(2) *If* $\Gamma \rhd M : A \Longrightarrow N_1$ *and* $\Gamma \rhd M : A \Longrightarrow N_2$, *then* $N_1 = N_2$.
(3) *If* $\Gamma \rhd p \longrightarrow p'_1 \uparrow A_1$ *and* $\Gamma \rhd p \longrightarrow p'_2 \uparrow A_2$, *then* $p'_1 = p'_2$ *and* $A_1 = A_2$.
(4) *If* $\Gamma \rhd A \Longrightarrow B_1$ *and* $\Gamma \rhd A \Longrightarrow B_2$, *then* $B_1 = B_2$.

LEMMA 4.2 (WEAKENING FOR TYPE RECONSTRUCTION). *If* $\Gamma_1 \rhd p \uparrow A$ *and* $\Gamma_1 \subseteq \Gamma_2$, *then* $\Gamma_2 \rhd p \uparrow A$.

4.3 Soundness

LEMMA 4.3. *If* $\Gamma \vdash p : A$, *then there exists* B, *determined by* Γ *and* p, *such that* $\Gamma \rhd p \uparrow B$ *and* $\Gamma \vdash p : B$ *and* $\Gamma \vdash \mathcal{S}_B(p) \leq A$.

PROOF. By induction on the proof of the assumption, and cases on the last rule used. □

COROLLARY 4.4. *If* $\Gamma \vdash \mathcal{E}[p] : A$ *and* $\Gamma \rhd p \uparrow \mathcal{S}(M)$, *then* $\Gamma \vdash \mathcal{E}[p] \equiv \mathcal{E}[M] : A$.

PROOF. By Lemma 4.3, $\Gamma \rhd \mathcal{E}[p] \uparrow B$, $\Gamma \vdash \mathcal{E}[p] : B$, and $\Gamma \vdash \mathcal{S}_B(\mathcal{E}[p]) \leq A$. By the determinacy of natural types, the first of these can be reconciled with $\Gamma \rhd p \uparrow \mathcal{S}(M)$ only if $\mathcal{E} = \diamond$ and $B = \mathcal{S}(M)$. Thus, $\Gamma \vdash p \equiv M : b$. and $\mathcal{S}_B(\mathcal{E}[p]) = \mathcal{S}(p)$. By inversion of subtyping, either $A = b$ or $A = \mathcal{S}(M')$ with $\Gamma \vdash p \equiv M' : b$. In either case, $\Gamma \vdash p \equiv M : A$. □

PROPOSITION 4.5. *If* $\Gamma \vdash \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] : A$ *then* $\Gamma \vdash \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] \equiv \mathcal{E}[[M'/x]M] : A$.

PROOF. By simultaneous induction on the proof of the assumption, and cases on the last rule used.

—Case: Rule (19).

$$\frac{\Gamma \vdash (\lambda x{:}A'.\, M) : \Pi x{:}A'_1.\, A''_1 \qquad \Gamma \vdash M' : A'_1}{\Gamma \vdash (\lambda x{:}A'.\, M)\, M' : [M'/x]A''},$$

where $A = [M'/x]A''_1$ and $\mathcal{E} = \diamond$. By Theorem 3.20 and inversion, we have $\Gamma \vdash A'$, $\Gamma, x : A' \vdash M \Uparrow B''$, $\Gamma \vdash \Pi x{:}A'.\, B'' \leq \Pi x{:}A'_1.\, A''_1$, $\Gamma \vdash M' \Uparrow B'$, and $\Gamma \vdash B' \leq A'_1$. By inversion of Rule (10) we have $\Gamma \vdash A'_1 \leq A'$ and $\Gamma, x : A'_1 \vdash B'' \leq A''_1$. By Theorem 3.18, we have $\Gamma, x : A' \vdash M : B''$ and by subsumption $\Gamma \vdash M' : A'$, so by Rule (48) we have $\Gamma \vdash (\lambda x{:}A'.\, M)\, M' \equiv [M'/x]M : [M'/x]B''$. By Proposition 3.7, $\Gamma \vdash [M'/x]B'' \leq [M'/x]A''_1$, so by subsumption we have $\Gamma \vdash (\lambda x{:}A'.\, M)\, M' \equiv [M'/x]M : [M'/x]A''_1$

—Case: Rule (23)

$$\frac{\Gamma \vdash \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] : b}{\Gamma \vdash \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] : \mathcal{S}(\mathcal{E}[(\lambda x{:}A'.\, M)\, M'])}.$$

By induction we have $\Gamma \vdash \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] \equiv \mathcal{E}[[M'/x]M] : b$. By Rules (28) and (40), we have $\Gamma \vdash \mathcal{E}[[M'/x]M] \equiv \mathcal{E}[(\lambda x{:}A'.\, M)\, M'] : \mathcal{S}(\mathcal{E}[(\lambda x{:}A'.\, M)\, M'])$, so the desired result follows by another application of Rule (28).

—The remaining cases follow similarly by induction. □

PROPOSITION 4.6.

(1) *If* $\Gamma \vdash \mathcal{E}[\pi_1\langle M', M''\rangle] : A$*, then* $\Gamma \vdash \mathcal{E}[\pi_1\langle M', M''\rangle] \equiv \mathcal{E}[M'] : A$.

(2) *If* $\Gamma \vdash \mathcal{E}[\pi_2\langle M', M''\rangle] : A$*, then* $\Gamma \vdash \mathcal{E}[\pi_2\langle M', M''\rangle] \equiv \mathcal{E}[M''] : A$.

PROOF. Generally similar to the previous proposition, using the principal type rules and Rules (49) and (50). □

COROLLARY 4.7. *If* $\Gamma \vdash M : A$ *and* $\Gamma \rhd M \Downarrow N$*, then* $\Gamma \vdash M \equiv N : A$.

PROOF. By Propositions 3.3, and 3.12, it suffices to show that if $\Gamma \vdash M : A$ and $\Gamma \rhd M \rightsquigarrow N$, then $\Gamma \vdash M \equiv N : A$. But all possibilities for the reduction step are covered by Corollary 4.4, Proposition 4.5, and Proposition 4.6. □

PROPOSITION 4.8 (SOUNDNESS OF NORMALIZATION)

(1) *If* $\Gamma \vdash M : A$ *and* $\Gamma \rhd M : A \Longrightarrow N$*, then* $\Gamma \vdash M \equiv N : A$.

(2) *If* $\Gamma \vdash p : A$ *and* $\Gamma \rhd p \longrightarrow p' \uparrow B$*, then* $\Gamma \vdash p \equiv p' : A$.

(3) *If* $\Gamma \vdash A$ *and* $\Gamma \rhd A \Longrightarrow B$*, then* $\Gamma \vdash A \equiv B$.

PROOF. By induction on algorithmic derivations. □

COROLLARY 4.9.

(1) *If* $\Gamma \vdash M_1 : A$*,* $\Gamma \vdash M_2 : A$*,* $\Gamma \rhd M_1 : A \Longrightarrow N$*, and* $\Gamma \rhd M_2 : A \Longrightarrow N$*, then* $\Gamma \vdash M_1 \equiv M_2 : A$.

(2) *If* $\Gamma \vdash A_1$*,* $\Gamma \vdash A_2$*,* $\Gamma \rhd A_1 \Longrightarrow B$*, and* $\Gamma \rhd A_2 \Longrightarrow B$*, then* $\Gamma \vdash A_1 \equiv A_2$.

4.4 Completeness of Normalization

The much more interesting question is whether two provably equivalent terms are guaranteed to produce the same normal form.

It is instructive to see why the direct approach of proving completeness by induction on the derivation of $\Gamma \vdash M_1 \equiv M_2 : A$ fails. We immediately run into trouble with such rules as Rule (31):

$$\frac{\Gamma \vdash M_1 \equiv M_2 : \Pi x{:}A'.\, A'' \qquad \Gamma \vdash M_1' \equiv M_2' : A'}{\Gamma \vdash M_1\, M_1' \equiv M_2\, M_2' : [M_1'/x]A''}$$

Here we would have by the induction hypothesis that M_1 and M_2 have a common normal form, as well as M_1' and M_2'. However, there is no obvious way to conclude that $M_1\, M_1'$ and $M_2\, M_2'$ have equal normal forms because normalization of an application is not defined in terms of the normal forms of the components.

Coquand [1991] proves the completeness of an equivalence algorithm for a lambda calculus with Π types using a form of Kripke logical relation. The key idea is to prove completeness by defining a stronger "logical" relation that implies algorithmic equivalence. For example, if two functions are logically related then their application to logically related arguments must yield logically related applications. By proving inductively that declarative equivalence implies not just algorithmic equivalence but logical equivalence, we have strengthened the induction hypothesis enough to allow cases such as Rule (31) to go through.

To show the completeness and termination for the algorithm we use a modified Kripke-style logical relations argument. The primary difficulty is the context-sensitive nature of normalization, which makes it difficult to define a logical equivalence relation that guarantees common normal forms. For example, if we have two terms M and N of type $\Sigma x{:}A'.\,A''$ then a natural definition for the logical equivalence relation would require both that $\pi_1 M$ and $\pi_1 N$ are logically equivalent at type A', and that $\pi_2 M$ and $\pi_2 N$ are logically equivalent. But at what type should the latter pair be compared? The most obvious choices are either $[\pi_1 M/x]A''$ or $[\pi_1 N/x]A''$. But even if we were to require that $\pi_2 M$ and $\pi_2 N$ be logically equivalent at *both* types, this appears insufficient to guarantee that M and N have equal normal forms: normalizing M and N at the type $\Sigma x{:}A'.\,A''$ will involve normalizing $\pi_2 M$ at type $[\pi_1 M/x]A''$ and normalizing $\pi_2 N$ at type $[\pi_1 N/x]A''$. If the logical relation guarantees only that at each common type the two projections have the same normal form, this is too weak.[2]

We therefore move to a formulation that allows us to express the fact that multiple terms considered at multiple types (and in multiple typing contexts) should all have a single common normal form. Our Kripke world Δ is a nonempty set of contexts. The preorder $\preceq$ is defined as follow:

$$\Delta_1 \preceq \Delta_2 \quad :\Longleftrightarrow \quad \forall \theta_2 \in \Delta_2.\exists \theta_1 \in \Delta_1.\theta_1 \subseteq \theta_2.$$

where $\subseteq$ is the inclusion ordering on contexts. That is, $\Delta_1 \preceq \Delta_2$ if and only if every context in Δ_2 extends some context in Δ_1.

We will use $\mathcal{A}$ and $\mathcal{B}$ to range over finite, nonempty sets of types, $\mathcal{M}$ and $\mathcal{B}$ to range over finite, nonempty sets of terms, and $\mathcal{G}$ to range over finite, nonempty sets of substitutions.

It turns out to be very convenient to define notation using sets of types where one would normally use a single type, and sets of type terms where one would normally use a single term, with the result being a set computed pointwise (and, where it makes sense, unioned):

$$\begin{array}{lcl}
[\mathcal{M}/x]A & := & \{[M/x]A \mid M \in \mathcal{M}\} \\
[\mathcal{M}/x]\mathcal{A} & := & \{[M/x]A \mid M \in \mathcal{M}, A \in \mathcal{A}\} \\
\mathcal{M}\,\mathcal{M}' & := & \{M\,M' \mid M \in \mathcal{M}, M' \in \mathcal{M}'\} \\
\pi_i \mathcal{M} & := & \{\pi_i M \mid M \in \mathcal{M}\} \\
\mathcal{G}(M) & := & \{\gamma(M) \mid \gamma \in \mathcal{G}\} \\
\mathcal{G}(\mathcal{A}) & := & \{\gamma(A) \mid \gamma \in \mathcal{G}, A \in \mathcal{A}\} \\
\mathrm{dom}(\mathcal{G}) & := & \bigcup\{\mathrm{dom}(\gamma) \mid \gamma \in \mathcal{G}\} \\
\mathrm{rng}(\mathcal{G}) & := & \bigcup\{\mathrm{rng}(\gamma) \mid \gamma \in \mathcal{G}\} \\
\mathcal{S}(\mathcal{M}) & := & \{\mathcal{S}(M) \mid M \in \mathcal{M}\}.
\end{array}$$

[2]Modifying the algorithm to substitute in the normal form of $\pi_1 M$ would resolve this problem, but then later we must show that a term and its fully normalized form are logically equivalent, which seems nonobvious.

—$\mathcal{A}$ ok $[\Delta]$ if
  —$\mathcal{A} = \{b\}$
  —Or, $\mathcal{A} = \mathcal{S}(\mathcal{N})$, and $\mathcal{N}$ in $\{b\}$ $[\Delta]$.
  —Or, $\mathcal{A} = \Pi x{:}\mathcal{A}'.\mathcal{A}''$, and $\mathcal{A}'$ ok $[\Delta]$, and for all $\Delta' \succeq \Delta$ if $\mathcal{M}$ in $\mathcal{A}$ $[\Delta']$ then $([\mathcal{M}/x]\mathcal{A}'')$ ok $[\Delta']$.
  —Or, $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\mathcal{A}''$, and $\mathcal{A}'$ ok $[\Delta]$, and for all $\Delta' \succeq \Delta$ if $\mathcal{M}$ in $\mathcal{A}'$ $[\Delta']$ then $([\mathcal{M}/x]\mathcal{A}'')$ ok $[\Delta']$.

—$\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$ if
  (1) $\mathcal{A}$ ok $[\Delta]$.
  (2) —$\mathcal{A} = \{b\}$ and $\exists N.\ \forall \theta \in \Delta, M \in \mathcal{M}.\ \ \theta \rhd M : b \Longrightarrow N$.
      (That is, the types all normalize to the same answer in all of the contexts.)
    —Or, $\mathcal{A} = \mathcal{S}(\mathcal{N})$ and $(\mathcal{M} \cup \mathcal{N})$ in $\{b\}$ $[\Delta]$.
    —Or, $\mathcal{A} = \Pi x{:}\mathcal{A}'.\mathcal{A}''$ and for all $\Delta' \succeq \Delta$ if $\mathcal{M}'$ in $\mathcal{A}'$ $[\Delta']$ then $(\mathcal{M}\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$.
    —Or, $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\mathcal{A}''$ and $\pi_1\mathcal{M}$ in $\mathcal{A}'$ $[\Delta]$, and $\pi_2\mathcal{M}$ in $([\pi_1\mathcal{M}/x]\mathcal{A}'')$ $[\Delta]$.

—$\mathcal{G}$ in Γ $[\Delta]$ if for every $x \in \mathrm{dom}(\Gamma)$ we have $\mathcal{G}(x)$ in $\mathcal{G}(\Gamma(x))$ $[\Delta]$

Fig. 10. Logical relations.

If $\mathcal{A}$ is a set $\{\Pi x{:}A'_i.\,A''_i \mid i \in I\}$, it is also useful to use a form of pattern-matching: we write $\mathcal{A} = \Pi x{:}\mathcal{A}'.\,\mathcal{A}''$ to mean that $\mathcal{A}' = \{A'_i \mid i \in I\}$ and $\mathcal{A}'' = \{A''_i \mid i \in I\}$. The notation $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$ when $\mathcal{A}$ is a set of Σ types is defined analogously.

The logical relations are then defined in Figure 10. Logically related sets of types, written $\mathcal{A}$ ok $[\Delta]$, are those which can index our logical relation for sets of terms. All elements of a logically related set of types must have the same "shape" (and the same size). In the base case, a set just containing the base type b is logically related, while a set of singleton types are logically related if they have the same normal form in all contexts in the world. A set of Π kinds is logically related if their domains form a logically related set, and if substitution instances of their codomains do too. The condition for a set of Σ kinds is similar.

Logical relatedness for a set of terms is defined inductively on the common size of the elements in a logically related set of types. In the base case, a set of terms of the base type must have the same normal form under all contexts in the world. Similarly, a set of terms is logically related with respect to a set of singleton types if the terms in the set and those in the singletons all have a common normal form. The definition for terms at a set of Π types is the usual Kripke logical relations definition lifted to sets: in all future worlds (a condition required to obtain monotonicity [Crary 2005]), related arguments should yield related results. Finally, a set of terms is logically related at a set of Σ types if their first and second projections are each logically related.

The first property to be checked is that the logical relations are monotone (preserved when passing to future worlds), which corresponds to weakening in the algorithmic relations.

LEMMA 4.10 (ALGORITHMIC WEAKENING)

(1) *If* $\Gamma \rhd M \rightsquigarrow N$ *and* $\Gamma' \succeq \Gamma$, *then* $\Gamma' \rhd M \rightsquigarrow N$
(2) *If* $\Gamma \rhd M \uparrow A$ *and* $\Gamma' \succeq \Gamma$, *then* $\Gamma' \rhd M \uparrow A$.
(3) *If* $\Gamma \rhd M \Downarrow p$ *and* $\Gamma' \succeq \Gamma$, *then* $\Gamma' \rhd M \Downarrow p$.

(4) *If* $\Gamma \rhd M \longrightarrow A \uparrow N$ *and* $\Gamma' \succeq \Gamma$*, then* $\Gamma' \rhd M \longrightarrow A \uparrow N$.
(5) *If* $\Gamma \rhd M : A \Longrightarrow N$ *and* $\Gamma' \succeq \Gamma$*, then* $\Gamma' \rhd M : A \Longrightarrow N$.
(6) *If* $\Gamma \rhd A \Longrightarrow B$ *and* $\Gamma' \succeq \Gamma$*, then* $\Gamma' \rhd A \Longrightarrow B$.

PROOF. By induction on algorithmic derivations. □

LEMMA 4.11 (MONOTONICITY)

(1) *If* $\mathcal{A}$ ok $[\Delta]$ *and* $\Delta' \succeq \Delta$*, then* $\mathcal{A}$ ok $[\Delta']$.
(2) *If* $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$ *and* $\Delta' \succeq \Delta$*, then* $\mathcal{M}$ in $\mathcal{A}$ $[\Delta']$.
(3) *If* $\mathcal{G}$ in Γ $[\Delta]$ *and* $\Delta' \succeq \Delta$*, then* $\mathcal{G}$ in Γ $[\Delta']$.

PROOF. By induction on the size of types, using Lemma 4.10. It is important here that the preorder on worlds is *not* merely a subset relation on sets of contexts. □

Next, we show that logical relatedness for sets acts like the property of "being a subset of an equivalence class". Subsets of a logically related set are logically related, and any two overlapping logically related sets (i.e., "two subsets of the same equivalence class") have a union that is logically related ("have a union that is a subset of a single equivalence class").

LEMMA 4.12.

(1) *If* $\mathcal{A}_2$ ok $[\Delta]$ *and* $\mathcal{A}_1 \subseteq \mathcal{A}_2$*, then* $\mathcal{A}_1$ ok $[\Delta]$.
(2) *If* $\mathcal{M}_2$ in $\mathcal{A}_2$ $[\Delta]$ *and* $\mathcal{M}_1 \subseteq \mathcal{M}_2$ *and* $\mathcal{A}_1 \subseteq \mathcal{A}_2$*, then* $\mathcal{M}_1$ in $\mathcal{A}_1$ $[\Delta]$.
(3) *If* $\mathcal{A}_1$ ok $[\Delta]$ *and* $\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset$ *and* $\mathcal{A}_2$ ok $[\Delta]$*, then* $(\mathcal{A}_1 \cup \mathcal{A}_2)$ ok $[\Delta]$.
(4) *If* $\mathcal{M}$ in $\mathcal{A}_1$ $[\Delta]$ *and* $\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset$ *and* $\mathcal{A}_2$ ok $[\Delta]$*, then* $\mathcal{M}$ in $(\mathcal{A}_1 \cup \mathcal{A}_2)$ $[\Delta]$. (*In particular, if* $\mathcal{A}_1 \subseteq \mathcal{A}_2$ *then* $\mathcal{M}$ in $\mathcal{A}_2$ $[\Delta]$.)
(5) *If* $\mathcal{M}_1$ in $\mathcal{A}$ $[\Delta]$ *and* $\mathcal{M}_1 \cap \mathcal{M}_2 \neq \emptyset$ *and* $\mathcal{M}_2$ in $\mathcal{A}$ $[\Delta]$*, then* $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}$ $[\Delta]$.

PROOF. By simultaneous induction on the sizes of the types involved.

(1) Assume $\mathcal{A}_2$ ok $[\Delta]$ and $\mathcal{A}_1 \subseteq \mathcal{A}_2$.
—Case: $\mathcal{A}_2 = \{b\}$. Since $\mathcal{A}_1$ is non-empty we have $\mathcal{A}_1 = \{b\} = \mathcal{A}_2$.
—Case: $\mathcal{A}_2 = \mathcal{S}(\mathcal{M}_2)$ and $\mathcal{A}_1 = \mathcal{S}(\mathcal{M}_1)$ with $\mathcal{M}_1 \subseteq \mathcal{M}_2$. Then $\mathcal{M}_2$ in $\{b\}$ $[\Delta]$, so inductively by Part (2), we have $\mathcal{M}_1$ in $\{b\}$ $[\Delta]$. Therefore, $\mathcal{A}_1$ ok $[\Delta]$.
—Case: $\mathcal{A}_2 = \Pi x{:}\mathcal{A}'_2.\, \mathcal{A}''_2$ and $\mathcal{A}_1 = \Pi x{:}\mathcal{A}'_1.\, \mathcal{A}''_1$ with $\mathcal{A}'_1 \subseteq \mathcal{A}'_2$ and $\mathcal{A}''_1 \subseteq \mathcal{A}''_2$. Then $\mathcal{A}'_2$ ok $[\Delta]$ so inductively by Part (1), we have $\mathcal{A}'_1$ ok $[\Delta]$. Now assume $\Delta' \succeq \Delta$ and $\mathcal{M}'$ in $\mathcal{A}_1$ $[\Delta']$. Since $\mathcal{A}_1 \neq \emptyset$, inductively by Part (1), we have $\mathcal{M}'$ in $\mathcal{A}'_2$ $[\Delta']$, so $[\mathcal{M}'/x]\mathcal{A}''_2$ ok $[\Delta']$. Inductively by Part (1) again, we have $[\mathcal{M}'/x]\mathcal{A}''_1$ ok $[\Delta']$. Therefore, $\mathcal{A}_1$ ok $[\Delta]$.
—Case: Case: $\mathcal{A}_2 = \Sigma x{:}\mathcal{A}'_2.\, \mathcal{A}''_2$ and $\mathcal{A}_1 = \Sigma x{:}\mathcal{A}'_1.\, \mathcal{A}''_1$ with $\mathcal{A}'_1 \subseteq \mathcal{A}'_2$ and $\mathcal{A}''_1 \subseteq \mathcal{A}''_2$. Same argument as for the previous case.

(2) Assume $\mathcal{M}_2$ in $\mathcal{A}_2$ $[\Delta]$ and $\mathcal{M}_1 \subseteq \mathcal{M}_2$ and $\mathcal{A}_1 \subseteq \mathcal{A}_2$
—Case: $\mathcal{A}_2 = \{b\}$. Again, $\mathcal{A}_1$ must be nonempty and hence $\mathcal{A}_1 = \{b\}$. Since all the types in $\mathcal{M}_2$ have a common normal form, then so do all the types in the subset $\mathcal{M}_1$. Therefore, $\mathcal{M}_1$ in $\{b\}$ $[\Delta]$.

—Case: $\mathcal{A}_2 = **\mathcal{S}(\mathcal{B}_2)$ and $\mathcal{A}_1 = \mathcal{S}(\mathcal{B}_1)$ for some $\mathcal{B}_1 \subseteq \mathcal{B}_2$. $(\mathcal{M}_2 \cup \mathcal{B}_2)$ in $\{b\}$ $[\Delta]$, so inductively, by Part (2), we have $(\mathcal{M}_1 \cup \mathcal{B}_1)$ in $\{b\}$ $[\Delta]$. Therefore, $\mathcal{M}_1$ in $\mathcal{A}_1$ $[\Delta]$.

—Case: $\mathcal{A}_2 = \Pi x{:}\mathcal{A}_2'.\, \mathcal{A}_2''$ and $\mathcal{A}_1 = \Pi x{:}\mathcal{A}_1'.\, \mathcal{A}_1''$ with $\mathcal{A}_1' \subseteq \mathcal{A}_2'$ and $\mathcal{A}_1'' \subseteq \mathcal{A}_2''$. Assume $\Delta' \succeq \Delta$ and $\mathcal{M}'$ in $\mathcal{A}_1'$ $[\Delta']$. Now $\mathcal{A}_2'$ ok $[\Delta']$, so inductively, by Part (4), we have $\mathcal{M}'$ in $\mathcal{A}_2'$ $[\Delta']$, and hence $(\mathcal{M}_2\, \mathcal{M}')$ in $[\mathcal{M}'/x]\mathcal{A}_2''$ $[\Delta']$. Inductively, by Part (2), we have $(\mathcal{M}_1\, \mathcal{M}')$ in $[\mathcal{M}'/x]\mathcal{A}_1''$ $[\Delta']$. Therefore, $\mathcal{M}_1$ in $\mathcal{A}_1$ $[\Delta]$.

—Case: Case: $\mathcal{A}_2 = \Sigma x{:}\mathcal{A}_2'.\, \mathcal{A}_2''$ and $\mathcal{A}_1 = \Sigma x{:}\mathcal{A}_1'.\, \mathcal{A}_1''$ with $\mathcal{A}_1' \subseteq \mathcal{A}_2'$ and $\mathcal{A}_1'' \subseteq \mathcal{A}_2''$. Then, $\pi_1\mathcal{M}_2$ in $\mathcal{A}_2'$ $[\Delta]$ so inductively by Part (2), we have $\pi_1\mathcal{M}_1$ in $\mathcal{A}_1'$ $[\Delta]$. Similarly, $\pi_2\mathcal{M}_2$ in $([\pi_1\mathcal{M}_2/x]\mathcal{A}_2'')$ $[\Delta]$ so inductively, by Part (2), we have $\pi_2\mathcal{M}_1$ in $([\pi_1\mathcal{M}_1/x]\mathcal{A}_1'')$ $[\Delta]$. Therefore, $\mathcal{M}_1$ in $\mathcal{A}_1$ $[\Delta]$.

(3) Assume $\mathcal{A}_1$ ok $[\Delta]$ and $\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset$ and $\mathcal{A}_2$ ok $[\Delta]$

—Case: $\mathcal{A}_1 = \{b\} = \mathcal{A}_2$. Then, $\{b\}$ ok $[\Delta]$ by definition.

—Case: Case: $\mathcal{A}_1 = \mathcal{S}(\mathcal{B}_1)$ and $\mathcal{A}_2 = \mathcal{S}(\mathcal{B}_2)$ with $\mathcal{B}_1 \cap \mathcal{B}_2 \neq \emptyset$. Then, all elements of $\mathcal{B}_1$ and $\mathcal{B}_2$ must have the same common normal form and so $(\{\mathcal{S}(M)\} \cup \mathcal{S}(\mathcal{B}_1) \cup \mathcal{S}(\mathcal{B}_2))$ ok $[\Delta]$.

—Case: $\mathcal{A}_1 = \Pi x{:}\mathcal{B}_1'.\, \mathcal{B}_1''$ and $\mathcal{A}_2 = \Pi x{:}\mathcal{B}_2'.\, \mathcal{B}_2''$ where $\mathcal{B}_1' \cap \mathcal{B}_2' \neq \emptyset$ and $\mathcal{B}_1'' \cap \mathcal{B}_2'' \neq \emptyset$. Inductively by Part (3), we have $(\mathcal{B}_1' \cup \mathcal{B}_2')$ ok $[\Delta]$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $(\mathcal{B}_1' \cup \mathcal{B}_2')$ $[\Delta']$. Inductively by Part (1), we have $\mathcal{M}'$ in $\mathcal{B}_1'$ $[\Delta']$ and $\mathcal{M}'$ in $\mathcal{B}_2'$ $[\Delta']$, and hence $([\mathcal{M}'/x]\mathcal{B}_1'')$ ok $[\Delta']$ and $([\mathcal{M}'/x]\mathcal{B}_2'')$ ok $[\Delta']$. Inductively by Part (3) again, we have $([\mathcal{M}'/x](\mathcal{B}_1'' \cup \mathcal{B}_2''))$ ok $[\Delta']$. Therefore, $(\mathcal{A}_1 \cup \mathcal{A}_2)$ ok $[\Delta]$.

—Case: $\mathcal{A}_1 = \Sigma x{:}\mathcal{B}_1'.\, \mathcal{B}_1''$ and $\mathcal{A}_2 = \Sigma x{:}\mathcal{B}_2'.\, \mathcal{B}_2''$ where $\mathcal{B}_1' \cap \mathcal{B}_2' \neq \emptyset$ and $\mathcal{B}_1'' \cap \mathcal{B}_2'' \neq \emptyset$. Same argument as for the previous case.

(4) Assume $\mathcal{M}$ in $\mathcal{A}_1$ $[\Delta]$ and $\mathcal{A}_1 \cap \mathcal{A}_2 \neq \emptyset$ and $\mathcal{A}_2$ ok $[\Delta]$. By the previous part (noninductively), we know that $(\mathcal{A}_1 \cup \mathcal{A}_2)$ ok $[\Delta]$.

—Case: $\mathcal{A}_1 = \{b\} = \mathcal{A}_2$. Then $\mathcal{M}$ in $\{b\}$ $[\Delta]$ holds by assumption.

—Case: $\mathcal{A}_1 = \mathcal{S}(\mathcal{B}_1)$ and $\mathcal{A}_2 = \mathcal{S}(\mathcal{B}_2)$ with $\mathcal{B}_1 \cap \mathcal{B}_2 \neq \emptyset$. Then, all the elements of $\mathcal{M}$ and $\mathcal{B}_1$ and $\mathcal{B}_2$ have a common unique normal form, so $\mathcal{M}$ in $(\mathcal{S}(\mathcal{B}_1) \cup \mathcal{S}(\mathcal{B}_2))$ $[\Delta]$.

—Case: $\mathcal{A}_1 = \Pi x{:}\mathcal{B}_1'.\, \mathcal{B}_1''$ and $\mathcal{A}_2 = \Pi x{:}\mathcal{B}_2'.\, \mathcal{B}_2''$ where $\mathcal{B}_1' \cap \mathcal{B}_2' \neq \emptyset$ and $\mathcal{B}_1'' \cap \mathcal{B}_2'' \neq \emptyset$. Since $(\mathcal{A}_1 \cup \mathcal{A}_2)$ ok $[\Delta]$ by the previous part, we have $(\mathcal{B}_1' \cup \mathcal{B}_2')$ ok $[\Delta]$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $(\mathcal{B}_1' \cup \mathcal{B}_2')$ $[\Delta']$. Inductively by Part (2), we have $\mathcal{M}'$ in $\mathcal{B}_1'$ $[\Delta']$ and have $\mathcal{M}'$ in $\mathcal{B}_2'$ $[\Delta']$. Using the assumptions, $(\mathcal{M}\, \mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{B}_1'')$ $[\Delta']$ and $([\mathcal{M}'/x]\mathcal{B}_2'')$ ok $[\Delta']$. Inductively by Part (4), we have $(\mathcal{M}\, \mathcal{M}')$ in $([\mathcal{M}'/x](\mathcal{B}_1'' \cup \mathcal{B}_2''))$ $[\Delta']$. Therefore, $\mathcal{M}$ in $(\mathcal{A}_1 \cup \mathcal{A}_2)$ $[\Delta]$.

—Case: $\mathcal{A}_1 = \Sigma x{:}\mathcal{B}_1'.\, \mathcal{B}_1''$ and $\mathcal{A}_2 = \Sigma x{:}\mathcal{B}_2'.\, \mathcal{B}_2''$ where $\mathcal{B}_1' \cap \mathcal{B}_2' \neq \emptyset$ and $\mathcal{B}_1'' \cap \mathcal{B}_2'' \neq \emptyset$. Since $(\mathcal{A}_1 \cup \mathcal{A}_2)$ ok $[\Delta]$, we have $(\mathcal{B}_1' \cup \mathcal{B}_2')$ ok $[\Delta]$. Then since $\pi_1\mathcal{M}$ in $\mathcal{B}_1'$ $[\Delta]$, inductively by Part (4), we have $\pi_1\mathcal{M}$ in $(\mathcal{B}_1' \cup \mathcal{B}_2')$ $[\Delta]$. Similarly, $\pi_2\mathcal{M}$ in $([\pi_1\mathcal{M}/x]\mathcal{B}_1'')$ $[\Delta]$, and $([\pi_1\mathcal{M}/x](\mathcal{B}_1'' \cup \mathcal{B}_2''))$ ok $[\Delta]$ and so inductively by Part (4), we have $\pi_2\mathcal{M}$ in $([\pi_1\mathcal{M}/x](\mathcal{B}_1'' \cup \mathcal{B}_2''))$ $[\Delta]$. Therefore, $\mathcal{M}$ in $(\mathcal{A}_1 \cup \mathcal{A}_2)$ $[\Delta]$.

(5) Assume $\mathcal{M}_1$ in $\mathcal{A}$ $[\Delta]$ and $\mathcal{M}_1 \cap \mathcal{M}_2 \neq \emptyset$ and $\mathcal{M}_2$ in $\mathcal{A}$ $[\Delta]$. By definition of the logical relations, $\mathcal{A}$ ok $[\Delta]$.

—Case: $\mathcal{A} = \{b\}$. Then the elements of C_1 have a common normal form, and the elements of C_2 have a common normal form, and by Lemma 4.1 these normal forms must be identical. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\{b\}$ $[\Delta]$.

—Case: $\mathcal{A} = \mathcal{S}(\mathcal{N})$. Then $\mathcal{M}_1$ and $\mathcal{N}$ have a common normal form as do $\mathcal{M}_2$ and $\mathcal{N}$. Again these must be equal, so $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{S}(\mathcal{N})$ $[\Delta]$.

—Case: $\mathcal{A} = \Pi x{:}\mathcal{A}'.\mathcal{A}''$. Assume $\Delta' \succeq \Delta$ and $\mathcal{M}'$ in $\mathcal{A}'$ $[\Delta']$. Then $(\mathcal{M}_1\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$ and $(\mathcal{M}_2\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. Since $\mathcal{M}' \neq \emptyset$, $(\mathcal{M}_1\,\mathcal{M}') \cap (\mathcal{M}_2\,\mathcal{M}') \neq \emptyset$ and hence inductively by Part (5), we have $((\mathcal{M}_1 \cup \mathcal{M}_2)\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}$ $[\Delta]$.

—Case: $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\mathcal{A}''$. Then $\pi_1\mathcal{M}_1$ in $\mathcal{A}'$ $[\Delta]$ and $\pi_1\mathcal{M}_2$ in $\mathcal{A}'$ $[\Delta]$ and these two sets of terms overlap, so inductively by Part (5), we have $\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}'$ $[\Delta]$. Next, we have $\pi_2\mathcal{M}_1$ in $([\pi_1\mathcal{M}_1/x]\mathcal{A}'')$ $[\Delta]$ and $\pi_2\mathcal{M}_2$ in $([\pi_1\mathcal{M}_2/x]\mathcal{A}'')$ $[\Delta]$ and $([\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)/x]\mathcal{A}'')$ ok $[\Delta]$, so, inductively by Parts (4) and (5), we have $\pi_2(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $([\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)/x]\mathcal{A}'')$ $[\Delta]$. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}$ $[\Delta]$. □

We next show that the logical relation for terms is preserved under head expansion.

LEMMA 4.13 (HEAD EXPANSION)

(1) *If* $\Gamma \rhd M' \rightsquigarrow M$, *then* $\Gamma \rhd \mathcal{E}[M'] \rightsquigarrow \mathcal{E}[M]$.
(2) *If* $\mathcal{M}_2$ in $\mathcal{A}$ $[\Delta]$ *and* $\forall\theta \in \Delta, M_1 \in \mathcal{M}_1.\ \exists M_2 \in \mathcal{M}_2.\ \ \theta \rhd M_1 \rightsquigarrow M_2$ *(i.e., if in all contexts in* Δ *everything in* $\mathcal{M}_1$ *head-reduces to something in* $\mathcal{M}_2$*) then* $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}$ $[\Delta]$.

PROOF.

(1) Obvious by definition of $\Gamma \rhd M \rightsquigarrow N$.
(2) By induction on the size of $\mathcal{A}$. Assume $\mathcal{M}_2$ in $\mathcal{A}$ $[\Delta]$ and $\forall\theta \in \Delta, M_1 \in \mathcal{M}_1.\ \exists M_2 \in \mathcal{M}_2.\ \ \theta \rhd M_1 \rightsquigarrow M_2$. By definition of the logical relation, $\mathcal{A}$ ok $[\Delta]$.

—Case: $\mathcal{A} = \{b\}$. Then, there is a type N such that $\forall\theta \in \Delta, M \in \mathcal{M}_2.\ \ \theta \rhd M : b \Longrightarrow N$. Let $\theta \in \Delta$ and $M_1 \in \mathcal{M}_1$ be given. By assumption, we may choose $M_2 \in \mathcal{M}_2$ such that $\theta \rhd M_1 \rightsquigarrow M_2$. Since $\theta \rhd M_2 : b \Longrightarrow N$, by definition of normalization at type b we know that $\theta \rhd M_1 : b \Longrightarrow N$ as well. As θ and M_1 were arbitrary, we have that $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\{b\}$ $[\Delta]$.

—Case: $\mathcal{A} = \mathcal{S}(\mathcal{N})$ and there exists a type N such that $\forall\theta \in \Delta, M_2 \in (\mathcal{M}_2 \cup \mathcal{N}).\ \ \theta \rhd M_2 : b \Longrightarrow N$. By exactly the same argument as for the previous case, $\theta \rhd M_1 : b \Longrightarrow N$ for every $\theta \in \Delta$ and every $M_1 \in \mathcal{M}_1$. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{S}(\mathcal{N})$ $[\Delta]$.

—Case: $\mathcal{A} = \Pi x{:}\mathcal{A}'.\mathcal{A}''$. Assume $\Delta' \succeq \Delta$ and $\mathcal{M}'$ in $\mathcal{A}'$ $[\Delta']$. Then $(\mathcal{M}_2\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. Now by Part (1), $\forall\theta \in \Delta, M_1 \in \mathcal{M}_1, M' \in \mathcal{M}'.\ \exists M_2 \in \mathcal{M}_2.\ \ \theta \rhd (M_1\,M') \rightsquigarrow (M_2\,M')$. That is, $\forall\theta \in \Delta, M_3 \in (\mathcal{M}_1\,\mathcal{M}').\ \exists M_4 \in (\mathcal{M}_2\,\mathcal{M}').\ \ \theta \rhd M_3 \rightsquigarrow M_4$. Thus, by the induction hypothesis, we have $(\mathcal{M}_1\,\mathcal{M}' \cup \mathcal{M}_2\,\mathcal{M}')$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. That is, $(\mathcal{M}_1 \cup \mathcal{M}_2)\,\mathcal{M}'$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\Pi x{:}\mathcal{A}'.\,\mathcal{A}''$ $[\Delta']$.

—Case: $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$. Then $\pi_1\mathcal{M}_2$ in $\mathcal{A}'$ $[\Delta]$ and, by Part (1), we have $\forall\theta \in \Delta.\forall M_3 \in \pi_1\mathcal{M}_1.\ \exists M_4 \in \pi_1\mathcal{M}_2.\ \ \theta \rhd M_3 \rightsquigarrow M_4$. Thus, by the induction hypothesis $\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\mathcal{A}'$ $[\Delta]$. An analogous argument starting with $\pi_2\mathcal{M}_2$ in $([\pi_1\mathcal{M}_2/x]\mathcal{A}'')$ $[\Delta]$ gives us $\pi_2(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $([\pi_1\mathcal{M}_2/x]\mathcal{A}'')$ $[\Delta]$. Now since $\mathcal{A}$ ok $[\Delta]$, we know that $([\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)/x]\mathcal{A}'')$ ok $[\Delta]$. Thus, by Lemma 4.12 Part (4), we have $\pi_2(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $([\pi_1(\mathcal{M}_1 \cup \mathcal{M}_2)/x]\mathcal{A}'')$ $[\Delta]$. Therefore, $(\mathcal{M}_1 \cup \mathcal{M}_2)$ in $\Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$ $[\Delta]$. □

Following all this preliminary work, we can now show that equivalence under the logical relations implies equality of normal forms. This requires a strengthened induction hypothesis: under suitable conditions variables (and more generally paths) are logically related.

LEMMA 4.14.

(1) *If* $\mathcal{A}$ ok $[\Delta]$*, then there exists* B *such that for all* $\theta \in \Delta$ *and all* $A \in \mathcal{A}$*, we have* $\theta \rhd A \Longrightarrow B$.

(2) *If* $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$*, then there exists* N *such that for all* $\theta \in \Delta$ *and all* $A \in \mathcal{A}$ *and all* $M \in \mathcal{M}$ *we have* $\theta \rhd M : A \Longrightarrow N$.

(3) *Assume* $\mathcal{M}$ *is a set of paths, that* $\mathcal{A}$ ok $[\Delta]$ *and there exists* N *such that*

$$\forall\theta \in \Delta, p \in \mathcal{M}.\ \exists A \in \mathcal{A}.\ \ \theta \rhd p \longrightarrow N \uparrow A,$$

(*i.e., the paths all have a common normal form and logically equivalent natural types*)*. Then,* $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$.

PROOF. By simultaneous induction on the sizes of the types in $\mathcal{A}$.

(1) Assume $\mathcal{A}$ ok $[\Delta]$

—Case: $\mathcal{A} = \{b\}$ Then, we can take $B := b$.

—Case: $\mathcal{A} = \mathcal{S}(\mathcal{N})$ and there exists a type N such that $\forall\theta \in \Delta, M \in \mathcal{N}.\ \ \theta \rhd M : b \Longrightarrow N$. Put $B := \mathcal{S}(N)$.

—Case: $\mathcal{A} = \Pi x{:}\mathcal{A}'.\,\mathcal{A}''$. Then, $\mathcal{A}'$ ok $[\Delta]$, so inductively by Part (1) these types have a common normal form B'. Now put $\Delta' := \{\,\theta, x : A' \mid \theta \in \Delta, A' \in \mathcal{A}'\,\}$. Inductively by Part (3), we have that $\{x\}$ in $\mathcal{A}'$ $[\Delta']$. Thus, by definition of the logical relation we have $([x/x]\mathcal{A}'')$ ok $[\Delta']$, that is, $\mathcal{A}''$ ok $[\Delta']$. Inductively by Part (1) again, the elements of $\mathcal{A}''$ have a common normal form B'' in all of the contexts in Δ'. But these contexts are a superset of the contexts that the algorithm would use in normalizing these elements when normalizing $\mathcal{A}$, so we know that the elements of $\mathcal{A}$ therefore all have the common normal form $\Pi x{:}B'.\,B''$.

—Case: $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$. Same argument as for the previous case.

(2) Assume $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$

—Case: $\mathcal{A} = \{b\}$. The desired result is exactly the definition of the logical relation.

—Case: $\mathcal{A} = \mathcal{S}(\mathcal{N})$. By definition of the logical relation, there exists N' such that for all $\theta \in \Delta$ and all $A \in \mathcal{A}$ and all $M \in \mathcal{M}$, we have $\theta \rhd M : b \Longrightarrow N'$.

But by definition of the algorithm, normalization at a singleton type is the same as normalization at type b, and so this last judgment is equivalent to $\theta \triangleright M : \mathcal{S}(N) \Longrightarrow N'$ for every $b \in \mathcal{N}$.

—Case: $\mathcal{A} = \Pi x{:}\mathcal{A}'.\,\mathcal{A}''$. then $\mathcal{A}'$ ok $[\Delta]$, so inductively by Part (1) these types have a common normal form B'. Now put $\Delta' := \{\theta, x : A' \mid \theta \in \Delta, A' \in \mathcal{A}'\}$. Inductively by Part (3), we have that $\{x\}$ in $\mathcal{A}'$ $[\Delta']$. Thus, by definition of the logical relation we have $\mathcal{M}\,\{x\}$ in $\mathcal{A}''$ $[\Delta']$. Inductively by Part (2) these applications have a common normal form N''. Therefore, by definition of the algorithm we have that for all $\theta \in \Delta$ and all $A \in \mathcal{A}$ and all $M \in \mathcal{M}$, we have $\theta \triangleright M : A \Longrightarrow \lambda x{:}B'.\,N''$.

—Case: $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$. Then, $\pi_1\mathcal{M}$ in $\mathcal{A}'$ $[\Delta]$ so inductively by Part (2) these types have a common normal form N'. Similarly, $\pi_2\mathcal{M}$ in $([\pi_1\mathcal{M}/x]\mathcal{A}')$ $[\Delta]$ so inductively by Part (2) again these types have a common normal form N''. Therefore, for all $\theta \in \Delta$ and all $A \in \mathcal{A}$ and all $M \in \mathcal{M}$, we have $\theta \triangleright M : A \Longrightarrow \langle N', N''\rangle$.

(3) Assume $\mathcal{M}$ is a set of paths, that $\mathcal{A}$ ok $[\Delta]$ and N satisfies

$$\forall\theta \in \Delta, p \in \mathcal{M}.\ \exists A \in \mathcal{A}.\ \ \theta \triangleright p \longrightarrow N \uparrow A.$$

—Case: $\mathcal{A} = \{b\}$. Then for all $\theta \in \Delta$ and all $p \in \mathcal{M}$, we have $\theta \triangleright p \Downarrow p$, and hence $\theta \triangleright p : b \Longrightarrow N$. Thus, by definition of the logical relation, $\mathcal{M}$ in $\{b\}$ $[\Delta]$.

—Case: $\mathcal{A} = \mathcal{S}(\mathcal{N})$. Then for all $\theta \in \Delta$ and $p \in \mathcal{M}$, we have $\theta \triangleright p \rightsquigarrow N$ for some $N \in \mathcal{N}$. But $\mathcal{N}$ in $\{b\}$ $[\Delta]$, so by Lemma 4.13 Part (2), we have $(\mathcal{M} \cup \mathcal{N})$ in $\{b\}$ $[\Delta]$. Therefore, $\mathcal{M}$ in $\mathcal{S}(\mathcal{N})$ $[\Delta]$.

—Case: $\mathcal{A} = \Pi x{:}\mathcal{A}'.\,\mathcal{A}''$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $\mathcal{A}'$ $[\Delta']$. Then inductively by Part (2), we know that there exists a type N' such that for every $\theta \in \Delta'$ and $A' \in \mathcal{A}'$ and $M' \in \mathcal{M}'$, we have $\theta \triangleright M' : A' \Longrightarrow N'$. Using Lemma 4.10, for all $\theta \in \Delta'$ and all $p \in \mathcal{M}$ and all $M' \in \mathcal{M}'$, we have $\theta \triangleright (p\,M') \longrightarrow (N\,N') \uparrow [M'/x]A''$ for some $A'' \in \mathcal{A}''$. Now $[\mathcal{M}'/x]\mathcal{A}''$ ok $[\theta']$, and so inductively by Part (3) , we have $\mathcal{M}\,\mathcal{M}'$ in $([\mathcal{M}'/x]\mathcal{A}'')$ $[\Delta']$. Therefore, $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$.

—Case: $\mathcal{A} = \Sigma x{:}\mathcal{A}'.\,\mathcal{A}''$. Then, $\mathcal{A}'$ ok $[\Delta]$ and by definition of the algorithm for every $\theta \in \Delta$ and $p \in \mathcal{M}$, we have $\theta \triangleright \pi_1 p \longrightarrow \pi_1 N \uparrow A'$ for some $A' \in \mathcal{A}'$. Thus, inductively by Part (3), we have $\pi_1\mathcal{M}$ in $\mathcal{A}'$ $[\Delta]$.

Similarly, for every $\theta \in \Delta$ and $p \in \mathcal{M}$, we have $\theta \triangleright \pi_2 p \longrightarrow \pi_2 N \uparrow [\pi_1 p/x]A''$ for some $A'' \in \mathcal{A}''$. Since $\mathcal{A}$ ok $[\Delta]$, we have $[\pi_1\mathcal{M}/x]\mathcal{A}''$ ok $[\Delta]$, so inductively by Part (3), we have $\pi_2\mathcal{M}$ in $[\pi_1\mathcal{M}/x]\mathcal{A}''$ $[\Delta]$. Therefore, $\mathcal{M}$ in $\mathcal{A}$ $[\Delta]$. □

One more lemma is used in our Fundamental Theorem:

LEMMA 4.15 (SUBSTITUTION EXTENSION). *Assume $\mathcal{G}$ in Γ $[\Delta]$, and $\Gamma, x : A \vdash$ ok, and $x \notin \mathrm{dom}(\mathcal{G})$, and $\mathcal{M}$ in $\mathcal{G}(A)$ $[\Delta]$. Then $\{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G}, M \in \mathcal{M}\}$ in $(\Gamma, x : A)$ $[\Delta]$.*

PROOF. Assume $\mathcal{G}$ in Γ $[\Delta]$, and $\Gamma, x : A \vdash$ ok, and $x \notin \mathrm{dom}(\mathcal{G})$, and $\mathcal{M}$ in $\mathcal{G}(A)$ $[\Delta]$, and put $\mathcal{G}' := \{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G}, M \in \mathcal{M}\}$. Using Propositions 3.1, and 3.2, $x \notin \mathrm{dom}(\Gamma)$ and $x \notin \mathrm{FV}(A)$. Thus, for all $y \in \mathrm{dom}(\Gamma)$, we have

$\mathcal{G}(y)$ in $\mathcal{G}(\Gamma(y))$ $[\Delta]$, and $\mathcal{G}'(y) = \mathcal{G}(y)$, and $\mathcal{G}'(\Gamma(y))) = \mathcal{G}(\Gamma(y))$, and $(\Gamma, x : A)(y) = \Gamma(y)$, and hence $\mathcal{G}'(y)$ in $\mathcal{G}'((\Gamma, x : A)(y))$ $[\Delta]$. Now $\mathcal{G}'(A) = \mathcal{G}(A)$ and by definition $\mathcal{G}'(x) = \mathcal{M}$, so $\mathcal{G}'(x)$ in $\mathcal{G}'((\Gamma, x : A)(x))$ $[\Delta]$. Thus, for all $y \in \text{dom}(\Gamma, x : A)$, we have $\mathcal{G}'(y)$ in $\mathcal{G}'((\Gamma, x : A)(y))$ $[\Delta]$. That is, $\mathcal{G}'$ in $(\Gamma, x : A)$ $[\Delta]$. □

Finally, we come to the Fundamental Theorem of Logical Relations, which relates provable equivalence of terms to the logical relations.

THEOREM 4.16 (FUNDAMENTAL THEOREM)

(1) *If* $\Gamma \vdash A$ *and* $\mathcal{G}$ in Γ $[\Delta]$*, then* $\mathcal{G}(A)$ ok $[\Delta]$.
(2) *If* $\Gamma \vdash A_1 \leq A_2$ *and* $\mathcal{G}$ in Γ $[\Delta]$*, then* $\mathcal{G}(A_1)$ ok $[\Delta]$ *and* $\mathcal{G}(A_2)$ ok $[\Delta]$ *and if* $\mathcal{M}$ in $\mathcal{G}(A_1)$ $[\Delta]$*, then* $\mathcal{M}$ in $\mathcal{G}(A_2)$ $[\Delta]$.
(3) *If* $\Gamma \vdash A_1 \equiv A_2$ *and* $\mathcal{G}$ in Γ $[\Delta]$*, then* $(\mathcal{G}(A_1) \cup \mathcal{G}(A_2))$ ok $[\Delta]$.
(4) *If* $\Gamma \vdash M : A$ *and* $\mathcal{G}$ in Γ $[\Delta]$*, then* $\mathcal{G}(M)$ in $\mathcal{G}(A)$ $[\Delta]$
(5) *If* $\Gamma \vdash M_1 \equiv M_2 : A$ *and* $\mathcal{G}$ in Γ $[\Delta]$*, then* $(\mathcal{G}(M_1) \cup \mathcal{G}(M_2))$ in $\mathcal{G}(A)$ $[\Delta]$

PROOF. By induction on the hypothesized derivations.

Type Well-formedness Rules: $\Gamma \vdash A$.

—Case: Rule (3), so $A = b$.
Then $\mathcal{G}(b) = b$ and $\{b\}$ ok $[\Delta]$.
—Case: Rule (4), with $A = \mathcal{S}(M)$ because $\Gamma \vdash M : b$.
By the inductive hypothesis, $\mathcal{G}(M)$ in $\mathcal{G}(b)$ $[\Delta]$. That is, $\mathcal{G}(M)$ in $\{b\}$ $[\Delta]$. Therefore, $\mathcal{G}(\mathcal{S}(M))$ ok $[\Delta]$.
—Case: Rule (5), with $A = \Pi x{:}A'.\, A''$ because $\Gamma, x : A' \vdash A''$.
By Proposition 3.4, we may assume $x \notin \text{dom}(\mathcal{G}) \cup \text{FV}(\text{rng}(\mathcal{G}))$. By Proposition 3.1, there is a strict subderivation $\Gamma, x : A' \vdash \text{ok}$ and by inversion a strict subderivation $\Gamma \vdash A'$. By the inductive hypothesis, $\mathcal{G}(A')$ ok $[\Delta]$. Let $\Delta' \succeq \Delta$ and assume that $\mathcal{M}$ in $\mathcal{G}(A')$ $[\Delta']$. Put $\mathcal{G}' := \{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G},\ M \in \mathcal{M}\}$. By Lemmas 4.11 and 4.15, $\mathcal{G}'$ in $(\Gamma, x : A')$ $[\Delta']$, so by the inductive hypothesis, we have $\mathcal{G}'(A'')$ ok $[\Delta']$. That is, $[\mathcal{M}/x](\mathcal{G}(A''))$ ok $[\Delta']$. Therefore, $\mathcal{G}(\Pi x{:}A'.\, A'')$ ok $[\Delta]$.
—Case: Rule (6), with $A = \Sigma x{:}A'.\, A''$ because $\Gamma, x : A' \vdash A''$. Analogous to the previous case.

Subtyping Rules. $\Gamma \vdash A_1 \leq A_2$. In all cases, the proofs that $\mathcal{G}(A_1)$ ok $[\Delta]$ and $\mathcal{G}(A_2)$ ok $[\Delta]$ follow essentially as in the proofs for the well formedness rules. Assume $\mathcal{M}$ in $\mathcal{G}(A_1)$ $[\Delta]$. We must show that $\mathcal{M}$ in $\mathcal{G}(A_2)$ $[\Delta]$.

—Case: Rule (7), with $A_1 = \mathcal{S}(M)$ and $A_2 = b$.
Then $\mathcal{M}$ in b $[\Delta]$ by the definition of the logical relations.
—Case: Rule (8). $A_1 = \mathcal{S}(M_1)$ and $A_2 = \mathcal{S}(M_2)$, with $\Gamma \vdash M_1 \equiv M_2 : b$.
By the inductive hypothesis, we have $(\mathcal{G}(M_1) \cup \mathcal{G}(M_2))$ in b $[\Delta]$. Thus, $\mathcal{G}(\mathcal{S}(M_1)) \cup \mathcal{G}(\mathcal{S}(M_2))$ ok $[\Delta]$. By Lemma 4.12, Parts (4) and (2), we have $\mathcal{M}$ in $\mathcal{G}(\mathcal{S}(M_1)) \cup \mathcal{G}(\mathcal{S}(M_2))$ $[\Delta]$ and hence $\mathcal{M}$ in $\mathcal{G}(\mathcal{S}(M_2))$ $[\Delta]$ as required.
—Case: Rule (9), with $A_1 = A_2 = b$.
Trivial, since $\mathcal{G}(b) = b$.

—Case: Rule (10), with $A_1 = \Pi x{:}A_1'.\,A_1''$ and $A_2 = \Pi x{:}A_2'.\,A_2''$ where $\Gamma \vdash A_2' \leq A_1'$ and $\Gamma, x : A_2' \vdash A_1'' \leq A_2''$.

By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $\mathcal{G}(A_2')$ $[\Delta']$. By the inductive hypothesis, $\mathcal{M}'$ in $\mathcal{G}(A_1')$ $[\Delta']$. Hence $(\mathcal{M}\,\mathcal{M}')$ in $[\mathcal{M}'/x]\mathcal{G}(A_1'')$ $[\Delta']$. That is, $(\mathcal{M}\,\mathcal{M}')$ in $\mathcal{G}'(A_1'')$ $[\Delta']$ where $\mathcal{G}' := \{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G},\ M \in \mathcal{M}\}$. By Lemma 4.11, Proposition 3.1, and Lemma 4.15, $\mathcal{G}'$ in $(\Gamma, x : A_2')$ $[\Delta']$. By the inductive hypothesis again, $(\mathcal{M}\,\mathcal{M}')$ in $\mathcal{G}'(A_2'')$ $[\Delta']$. That is, $(\mathcal{M}\,\mathcal{M}')$ in $[\mathcal{M}'/x]\mathcal{G}(A_2'')$ $[\Delta']$. Therefore, $\mathcal{M}$ in $\mathcal{G}(\Pi x{:}A_2'.\,A_2'')$ $[\Delta]$.

—Case: Rule (11). $A_1 = \Sigma x{:}A_1'.\,A_1''$ and $A_2 = \Sigma x{:}A_2'.\,A_2''$ with $\Gamma \vdash A_1' \leq A_2'$ and $\Gamma, x : A_1' \vdash A_1'' \leq A_2''$.

By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By the definitions of the logical relations, $\pi_1\mathcal{M}$ in $\mathcal{G}(A_1')$ $[\Delta]$. By the inductive hypothesis, $\pi_1\mathcal{M}$ in $\mathcal{G}(A_2')$ $[\Delta]$. Put $\mathcal{G}' := \{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G},\ M \in \pi_1\mathcal{M}\}$. By Proposition 3.1, and Lemma 4.15, we have $\mathcal{G}'$ in $\Gamma, x : A_1'$ $[\Delta]$. Now $\pi_2\mathcal{M}$ in $[\pi_1\mathcal{M}/x](\mathcal{G}(A_1''))$ $[\Delta]$, that is, $\pi_2\mathcal{M}$ in $\mathcal{G}'(A_1'')$ $[\Delta]$. So, by the inductive hypothesis $\pi_2\mathcal{M}$ in $\mathcal{G}'(A_2'')$ $[\Delta]$. That is, $\pi_2\mathcal{M}$ in $([\pi_1\mathcal{M}/x]\mathcal{G}(A_2))$ $[\Delta]$. Therefore, $\mathcal{M}$ in $\mathcal{G}(\Sigma x{:}A_2'.\,A_2'')$ $[\Delta]$.

Type Equivalence Rules: $\Gamma \vdash A_1 \equiv A_2$.

—Case: Rule (12). $A_1 = A_2 = b$.
$\{b\}$ ok $[\Delta]$ by definition.

—Case: Rule (13). $A_1 = \mathcal{S}(M_1)$ and $A_2 = \mathcal{S}(M_2)$ with $\Gamma \vdash M_1 \equiv M_2 : b$.
By the inductive hypothesis, $(\mathcal{G}(M_1) \cup \mathcal{G}(M_2))$ in b $[\Delta]$. Therefore, by definition of the logical relation, $(\mathcal{G}(\mathcal{S}(M_1)) \cup \mathcal{G}(\mathcal{S}(M_2)))$ ok $[\Delta]$.

—Case: Rule (14). $A_1 = \Pi x{:}A_1'.\,A_1''$ and $A_2 = \Pi x{:}A_2'.\,A_2''$ with $\Gamma \vdash A_2' \equiv A_1'$ and $\Gamma, x : A_1' \vdash A_1'' \equiv A_2''$.
By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By the inductive hypothesis, $(\mathcal{G}(A_1') \cup \mathcal{G}(A_2'))$ ok $[\Delta]$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}$ in $(\mathcal{G}(A_1') \cup \mathcal{G}(A_2'))$ $[\Delta']$. By Lemma 4.12 Part (2), $\mathcal{M}$ in $\mathcal{G}(A_1')$ $[\Delta']$. By Lemma 4.11, Proposition 3.1, and Lemma 4.15, then, $\mathcal{G}'$ in $(\Gamma, x : A_1')$ $[\Delta']$ where $\mathcal{G}' := \{\gamma[x \mapsto M] \mid \gamma \in \mathcal{G},\ M \in \mathcal{M}\}$. By the inductive hypothesis again, $(\mathcal{G}'(A_1'') \cup \mathcal{G}'(A_2''))$ ok $[\Delta]$. That is, $[\mathcal{M}'/x](\mathcal{G}(A_1'') \cup \mathcal{G}(A_2''))$ ok $[\Delta]$. Therefore, $(\mathcal{G}(\Pi x{:}A_1'.\,A_1'') \cup \mathcal{G}(\Pi x{:}A_2'.\,A_2''))$ ok $[\Delta]$.

—Case: Rule (15). Same proof as for previous case.

Term Validity Rules $\Gamma \vdash M : A$.

—Case: Rule (16). $M = c_i$ and $A = b$. Then $\{b\}$ ok $[\Delta]$ and $\theta \rhd c_i \longrightarrow c_i \uparrow b$ for every $\theta \in \Delta$. Thus, by Lemma 4.14 Part (3), we have $\{c_i\}$ in b $[\Delta]$.

—Case: Rule (17). $M = x$ and $A = \Gamma x$. By assumption, $\mathcal{G}(x)$ in $\mathcal{G}(\Gamma x)$ $[\Delta]$.

—Case: Rule (18). $M = \lambda x{:}A'.\,M''$, $A = \Pi x{:}A'.\,A''$, and there is a subderivation $\Gamma, x : A' \vdash M'' : A''$. By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By Proposition 3.1, there is a strict subderivation $\Gamma \vdash A'$. By the inductive hypothesis, $\mathcal{G}(A')$ ok $[\Delta]$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $\mathcal{G}(A')$ $[\Delta']$. Put $S' := \{\gamma[x \mapsto M'] \mid \gamma \in$

$\mathcal{G},\ M' \in \mathcal{M}'\}$. Then, by Lemma 4.11 and Proposition 3.1, and Lemma 4.15, $\mathcal{G}'$ in $(\Gamma, x : A')$ $[\Delta']$. By the inductive hypothesis again, $\mathcal{G}'(M'')$ in $\mathcal{G}'(A'')$ $[\Delta']$. That is, $[\mathcal{M}'/x]\mathcal{G}(M'')$ in $[\mathcal{M}'/x]\mathcal{G}(A'')$ $[\Delta']$. Now for every $\theta \in \Delta'$ and every $M' \in \mathcal{M}'$, $\theta \rhd (\mathcal{G}(\lambda x{:}A'.\,M))(M') \rightsquigarrow [M'/x]\mathcal{G}(M'')$. Thus, by Lemma 4.13, $(\mathcal{G}(\lambda x{:}A'.\,M))(\mathcal{M}')$ in $[\mathcal{M}'/x]\mathcal{G}(A'')$ $[\Delta']$. Therefore, $\mathcal{G}(\lambda x{:}A'.\,M)$ in $\mathcal{G}(\Pi x{:}A'.\,A'')$ $[\Delta]$.

—Case: Rule (19). $M = M''\,M'$ where $\Gamma \vdash M'' : \Pi x{:}A'.\,A''$ and $\Gamma \vdash M' : A'$, and $A = [M'/x]A''$. By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By the inductive hypothesis twice, we have $\mathcal{G}(M'')$ in $\Pi x{:}\mathcal{G}(A').\,\mathcal{G}(A)$ $[\Delta]$ and $\mathcal{G}(M')$ in $\mathcal{G}(A')$ $[\Delta]$. By definition of the logical relations, $\mathcal{G}(M''\,M')$ in $[\mathcal{G}(M')/x]\mathcal{G}(A'')$ $[\Delta]$. Since $[\mathcal{G}(M')/x]\mathcal{G}(A'') \supseteq \mathcal{G}([M'/x]A'')$, by Lemma 4.12, we have $\mathcal{G}(M''\,M')$ in $\mathcal{G}([M'/x]A''$ $[\Delta]$ as required.

—Case: Rule (20). $M = \langle M', M''\rangle$ and $A = \Sigma x{:}A'.\,A''$ where $\Gamma \vdash A$, $\Gamma \vdash M' : A'$ and $\Gamma \vdash M'' : [M'/x]A''$. By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By the inductive hypothesis, $\mathcal{G}(M')$ in $\mathcal{G}(A')$ $[\Delta]$ and $\mathcal{G}(M'')$ in $\mathcal{G}([M'/x]A'')$ $[\Delta]$ and $\mathcal{G}(\Sigma x{:}A'.\,A'')$ ok $[\Delta]$. Now, for every $\theta \in \Delta$, we have $\theta \rhd \mathcal{G}(\pi_1\langle M', M''\rangle) \rightsquigarrow \mathcal{G}(M')$ and $\theta \rhd \mathcal{G}(\pi_2\langle M', M''\rangle) \rightsquigarrow \mathcal{G}(M'')$. Thus, by Lemma 4.13, we have $\mathcal{G}(\pi_1\langle M', M''\rangle)$ in $\mathcal{G}(A')$ $[\Delta]$ and $\mathcal{G}(\pi_2\langle M', M''\rangle)$ in $\mathcal{G}([M'/x]A'')$ $[\Delta]$. That is, $\pi_1(\mathcal{G}(\langle M', M''\rangle))$ in $\mathcal{G}(A')$ $[\Delta]$ and $\pi_2(\mathcal{G}(\langle M', M''\rangle))$ in $\mathcal{G}([M'/x]A'')$ $[\Delta]$.

Further, by definition of the type validity logical relation we have $[\mathcal{G}(M')/x]\mathcal{G}(A'')$ ok $[\Delta]$. Then $[\mathcal{G}(M')/x]\mathcal{G}(A'') \supseteq \mathcal{G}([M'/x]A'')$, so Lemma 4.12 yields $\pi_2(\mathcal{G}(\langle M', M''\rangle))$ in $[\mathcal{G}(M')/x]\mathcal{G}(A'')$ $[\Delta]$.

By definition of the logical relation, $\mathcal{G}(\langle M', M''\rangle)$ in $\mathcal{G}(\Pi x{:}A'.\,A'')$ $[\Delta]$.

—Case: Rule (21). $M = \pi_1 M'$ and $\Gamma \vdash M' : \Sigma x{:}A.\,A''$. By the inductive hypothesis, $\mathcal{G}(M')$ in $\mathcal{G}(\Sigma x{:}A'.\,A'')$ $[\Delta]$. Therefore, by definition of the logical relation, $\pi_1(\mathcal{G}(M'))$ in $\mathcal{G}(A')$ $[\Delta]$, i.e., $\mathcal{G}(\pi_1 M')$ in $\mathcal{G}(A')$ $[\Delta]$.

—Case: Rule (22). $M = \pi_2 M'$, $A = [\pi_1 M'/x]A''$, and $\Gamma \vdash M' : \Sigma x{:}A.\,A''$. By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. By the inductive hypothesis, $\mathcal{G}(M')$ in $\mathcal{G}(\Sigma x{:}A'.\,A'')$ $[\Delta]$. Therefore, by definition of the logical relation, $\pi_2(\mathcal{G}(M'))$ in $[\pi_1(\mathcal{G}(M))/x]\mathcal{G}(A'')$ $[\Delta]$. By Lemma 4.12, $\mathcal{G}(\pi_2 M')$ in $\mathcal{G}([\pi_1 M/x]A'')$ $[\Delta]$.

—Case: Rule (23). $A = \mathcal{S}(M)$ and $\Gamma \vdash M : b$. By the inductive hypothesis, $\mathcal{G}(M)$ in b $[\Delta]$. Thus, by definition of the logical relation, $\mathcal{G}(M)$ in $\mathcal{G}(\mathcal{S}(M))$ $[\Delta]$.

—Case: Rule (24). $A = \Sigma x{:}A'.\,A''$ and $\Gamma \vdash \pi_1 M : A'$ and $\Gamma \vdash \pi_2 M : [\pi_1 M/x]A''$. By the inductive hypothesis twice, $\pi_1(\mathcal{G}(M))$ in $\mathcal{G}(A')$ $[\Delta]$ and $\pi_2(\mathcal{G}(M))$ in $\mathcal{G}([\pi_1 M'/x]A'')$ $[\Delta]$.
Put $S' := \{\gamma[x \mapsto M'] \mid \gamma \in \mathcal{G},\ M' \in \pi_1\mathcal{G}(M)\}$. Then, by Lemma 4.11 and Proposition 3.1 and Lemma 4.15, $\mathcal{G}'$ in $(\Gamma, x : A')$ $[\Delta]$. Now $\Gamma, x : A' \vdash A''$, so by the inductive hypothesis again, $\mathcal{G}'(A'')$ ok $[\Delta]$. Since $\mathcal{G}'(A'') \supseteq \mathcal{G}([\pi_1 M/x]A'')$, by Lemma 4.12, we have $\pi_2(\mathcal{G}(M))$ in $\mathcal{G}'(A'')$ $[\Delta]$. By definition of the logical relation, $\mathcal{G}(M)$ in $\mathcal{G}(\Pi x{:}A.\,A'')$ $[\Delta]$.

—Case: Rule (25). $A = \Pi x{:}A'.\,A''$ and $\Gamma, x : A' \vdash M\,x : A''$. By Proposition 3.4, we may assume $x \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{FV}(\mathrm{rng}(\mathcal{G}))$. Let $\Delta' \succeq \Delta$ and assume $\mathcal{M}'$ in $\mathcal{G}(A')$ $[\Delta']$. Put $\mathcal{G}' := \{\gamma[x \mapsto M'] \mid \gamma \in \mathcal{G},\ M' \in \mathcal{M}'\}$. By Lemma 4.11, Proposition 3.1 and Lemma 4.15, $\mathcal{G}'$ in $(\Gamma, x : A')$ $[\Delta']$. By the inductive hypothesis, $\mathcal{G}'(M\,x)$ in $\mathcal{G}'(A'')$ $[\Delta']$. That is, $(\mathcal{G}(M))\,\mathcal{M}')$ in $[\mathcal{M}'/x](\mathcal{G}(A''))$ $[\Delta']$. Therefore, $\mathcal{G}(\Pi x{:}A'.\,A'')$ ok $[\Delta]$ and $\mathcal{G}(M)$ in $\mathcal{G}(\Pi x{:}A'.\,A'')$ $[\Delta]$.

—Case: Rule (26). There exist subderivations $\Gamma \vdash M : A_1$ and $\Gamma \vdash A_1 \leq A$. By the inductive hypothesis, $\mathcal{G}(M)$ in $\mathcal{G}(A_1)$ $[\Delta]$. So, applying the inductive hypothesis to the other subderivation, we have $\mathcal{G}(M)$ in $\mathcal{G}(A)$ $[\Delta]$.

Term Equivalence Rules: $\Gamma \vdash M_1 \equiv M_2 : A$.

—Case: Rule (27). $M_1 = M_2 = M$ and $\Gamma \vdash M : A$. Then $\mathcal{G}(M_1) \cup \mathcal{G}(M_2) = \mathcal{G}(M)$, and by the inductive hypothesis, we have $\mathcal{G}(M)$ in $\mathcal{G}(A)$ $[\Delta]$.

—Case: Rule (28). There is a subderivation $\Gamma \vdash M_2 \equiv M_1 : A$. Then $\mathcal{G}(M_1) \cup \mathcal{G}(M_2) = \mathcal{G}(M_2) \cup \mathcal{G}(M_1)$ and by the inductive hypothesis, we have $(\mathcal{G}(M_2) \cup \mathcal{G}(M_1))$ in $\mathcal{G}(A)$ $[\Delta]$.

—Case: Rule (29). There are subderivations $G \vdash M_1 \equiv M' : A$ and $\Gamma \vdash M' \equiv M_2 : A$.

By the inductive hypothesis twice, we have $(\mathcal{G}(M_1) \cup \mathcal{G}(M'))$ in $\mathcal{G}(A)$ $[\Delta]$ and $(\mathcal{G}(M') \cup \mathcal{G}(M_2))$ in $\mathcal{G}(A)$ $[\Delta]$. By Lemma 4.12 Part (5), we have $(\mathcal{G}(M_1) \cup \mathcal{G}(M') \cup \mathcal{G}(M_2))$ in $\mathcal{G}(A)$ $[\Delta]$, and therefore by Lemma 4.12 Part (2), we have $(\mathcal{G}(M_1) \cup \mathcal{G}(M_2))$ in $\mathcal{G}(A)$ $[\Delta]$.

—Case: Rules (30)–(36): Analogous to the proofs for the corresponding term validity rules.

—Case: Rule (37). By the inductive hypotheses.

—Case: Rule (38). $A = \mathcal{S}(M_2)$ and there is a subderivation $\Gamma \vdash M_1 : \mathcal{S}(M_2)$. By the inductive hypothesis, $\mathcal{G}(M_1)$ in $\mathcal{G}(\mathcal{S}(M_2))$ $[\Delta]$. By definition of the logical relation, $(\mathcal{G}(M_1) \cup \mathcal{G}(M_2))$ in $\mathcal{G}(\mathcal{S}(M_2))$ $[\Delta]$. □

Finally, we need to know that the identity substitution satisfies the requirements of Theorem 4.16.

LEMMA 4.17. *If* $\Gamma \vdash$ *ok then for all* $y \in \mathrm{dom}(\Gamma)$*, we have* $\{y\}$ in $(\Gamma(y))$ $[\{\Gamma\}]$. *That is,* $\{\mathrm{id}\}$ in Γ $[\{\Gamma\}]$ *where* id *is the identity function.*

PROOF. By induction on the proof of $\Gamma \vdash$ ok.

—Case: Empty context. Vacuous.

—Case: $\Gamma, x : A$ where $\Gamma \vdash A$ and $x \notin \mathrm{dom}(\Gamma)$. By Proposition 3.1, there is a subderivation $\Gamma \vdash$ ok, so by the inductive hypothesis, id in Γ $[\{\Gamma\}]$. By Lemma 4.11, id in Γ $[\{\Gamma, x : A\}]$. Also, by Theorem 4.16, we have $\{A\}$ ok $[\{\Gamma\}]$, and and by Lemma 4.11, $\{A\}$ ok $[\{\Gamma, x : A\}]$. Now $\Gamma, x : A \rhd x \longrightarrow x \uparrow A$ so by Lemma 4.14, $\{x\}$ in $\{A\}$ $[\{\Gamma, x : A\}]$. Therefore, by Lemma 4.15, id in $(\Gamma, x : A)$ $[\{\Gamma, x : A\}]$ □

This yields the completeness result for the normalization algorithm:

COROLLARY 4.18 (COMPLETENESS)

(1) *If* $\Gamma \vdash A_1 \equiv A_2$*, then* $\{A_1, A_2\}$ ok $[\{\Gamma\}]$

(2) *If* $\Gamma \vdash M_1 \equiv M_2 : A$*, then* $\{M_1, M_2\}$ in $\{A\}$ $[\{\Gamma\}]$.

(3) *If* $\Gamma \vdash A_1 \equiv A_2$*, then there exists* B *such that* $\Gamma \rhd A_1 \Longrightarrow B$ *and* $\Gamma \rhd A_2 \Longrightarrow B$.

Type equivalence	
$\Gamma \triangleright b \Longleftrightarrow b$	always
$\Gamma \triangleright \mathcal{S}(M_1) \Longleftrightarrow \mathcal{S}(M_2)$	if $\Gamma \triangleright M_1 \Longleftrightarrow M_2 : b$
$\Gamma \triangleright \Pi x{:}A_1'.\, A_1'' \Longleftrightarrow \Pi x{:}A_2'.\, A_2''$	if $\Gamma \triangleright A_1' \Longleftrightarrow A_2'$ and $\Gamma, x : A_1' \triangleright A_1'' \Longleftrightarrow A_2''$, where $x \notin \mathrm{dom}(\Gamma)$
$\Gamma \triangleright \Sigma x{:}A_1'.\, A_1'' \Longleftrightarrow \Sigma x{:}A_2'.\, A_2''$	if $\Gamma \triangleright A_1' \Longleftrightarrow A_2'$ and $\Gamma, x : A_1' \triangleright A_1'' \Longleftrightarrow A_2''$, where $x \notin \mathrm{dom}(\Gamma)$
Term equivalence	
$\Gamma \triangleright M_1 \Longleftrightarrow M_2 : b$	if $\Gamma \triangleright M_1 \Downarrow p_1$, $\Gamma \triangleright M_2 \Downarrow p_2$, and $\Gamma \triangleright p_1 \longleftrightarrow p_2 \uparrow b$
$\Gamma \triangleright M_1 \Longleftrightarrow M_2 : \mathcal{S}(N)$	always
$\Gamma \triangleright M_1 \Longleftrightarrow M_2 : \Pi x{:}A'.\, A''$	if $\Gamma, x : A' \triangleright M_1\, x \Longleftrightarrow M_2\, x : A''$, where $x \notin \mathrm{dom}(\Gamma)$
$\Gamma \triangleright M_1 \Longleftrightarrow M_2 : \Sigma x{:}A'.\, A''$	if $\Gamma \triangleright \pi_1 M_1 \Longleftrightarrow \pi_1 M_2 : A'$ and $\Gamma \triangleright \pi_2 M_1 \Longleftrightarrow \pi_2 M_2 : [\pi_1 M_1/x]A''$
Path equivalence	
$\Gamma \triangleright c_i \longleftrightarrow c_i \uparrow b$	
$\Gamma \triangleright x \longleftrightarrow x \uparrow \Gamma(x)$	
$\Gamma \triangleright p_1\, M_1 \longleftrightarrow p_2\, M_2 \uparrow [M_1/x]A''$	if $\Gamma \triangleright p_1 \longleftrightarrow p_2 \uparrow \Pi x{:}A'.\, A''$ and $\Gamma \triangleright M_1 \Longleftrightarrow M_2 : A'$
$\Gamma \triangleright \pi_1 p_1 \longleftrightarrow \pi_1 p_2 \uparrow A'$	if $\Gamma \triangleright p_1 \longleftrightarrow p_2 \uparrow \Sigma x{:}A'.\, A''$
$\Gamma \triangleright \pi_2 p_1 \longleftrightarrow \pi_2 p_2 \uparrow [\pi_1 p_1/x]A''$	if $\Gamma \triangleright p_1 \longleftrightarrow p_2 \uparrow \Sigma x{:}A'.\, A''$

Fig. 11. Definition of binary equivalence algorithms.

(4) *If* $\Gamma \vdash M_1 \equiv M_2 : A$*, then there exists* N *such that* $\Gamma \triangleright M_1 : A \Longrightarrow N$ *and* $\Gamma \triangleright M_2 : A \Longrightarrow N$.

PROOF. By Lemma 4.17, Theorem 4.16, and Lemma 4.14. □

COROLLARY 4.19 (DECIDABILITY). *Equivalence for well-formed terms and types is decidable.*

PROOF. By Rule (27) and Corollary 4.18, normalization of any well-formed type or term terminates. Decidability therefore follows by soundness and completeness of normalization. □

We conclude with an application of completeness.

COROLLARY 4.20 (CONSISTENCY). *Assume* c_1 *and* c_2 *are distinct term constants. Then the judgment*

$$\Gamma \vdash c_1 \equiv c_2 : b$$

is not provable.

PROOF. We may assume without loss of generality that $\Gamma \vdash$ ok because otherwise by Proposition 3.10 the equivalence cannot be proved. Therefore, the constants are well formed but clearly algorithmically inequivalent, and so by completeness they are not provably equivalent. □

In proving soundness of the TILT compiler's intermediate language, these sorts of consistency properties are essential. The argument that, for example, every closed value of type int is an integer constant would fail if the type int were provably equivalent (through some long transitive chain, perhaps) to a function type, a product type, or another base type.

5. BINARY EQUIVALENCE ALGORITHMS

Alternative algorithms for equivalence-checking that avoid explicit normalization are shown in Figure 11.

The algorithmic *type equivalence* judgment

$$\Gamma \rhd A_1 \Longleftrightarrow A_2$$

models declarative equivalence; given two types satisfying $\Gamma \vdash A_1$ and $\Gamma \vdash A_2$ it determines whether there is a proof $\Gamma \vdash A_1 \equiv A_2$.

Similarly, the algorithmic *term equivalence* relation

$$\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$$

models the declarative judgment $\Gamma \vdash M_1 \equiv M_2 \,:\, A$ on well-formed terms. This implements what is effectively simultaneous normalization and comparison of the two types, but can be more efficient than the actual computation of normal forms. We never have to simultaneously store both normal forms in memory, and there are opportunities for short-circuiting. We can stop early if differences are detected, or if two types are found to be identical up to names of bound variables. As shown in Figure 11, when comparing two terms at a singleton type the algorithm can immediately report success. And, there is no need to normalize the type annotations in λ-abstractions.

This comparison judgment corresponding to path normalization is the algorithmic path equivalence relation

$$\Gamma \rhd p_1 \longleftrightarrow p_2 \uparrow A.$$

Given two well formed head-normal paths $\Gamma \vdash p_1 : A_1$ and $\Gamma \vdash p_2 : A_2$, this should succeed, yielding A, if and only if $\Gamma \vdash p_1 \equiv p_2 : A$ and A is the natural type of p_1 with respect to Γ. The only question that arises when writing down these rules is in the case for comparing two applications. If the two function parts are recursively found to be equal, the two arguments must then be compared. Since the two arguments need not be in normal form, they must be compared using the $\Leftrightarrow$ judgment; in this case we must decide at which type the two arguments should be compared.

The right answer is the domain type of the principal type of the function parts. Assume we want to compare $p_1\, M_1$ and $p_2\, M_2$ using the typing context Γ, and that the principal type of p_1 (which will be equivalent to the principal type of p_2 if they are structurally-equivalent paths, that is, with the same head variable and with equivalent arguments in any applications) is $\Pi x{:}A'.\, A''$. Then this is the *least* type at which the two paths are provably equal, and hence by contravariance the domain type is *greatest*. By comparing M_1 and M_2 at type A', then, we have the best chance of proving them equal. (Two terms equivalent at a subtype will be equivalent at a supertype, but not necessarily vice versa.) Thus, to find as many equivalences as possible, A' is intuitively the correct type for the algorithm to compare function arguments. Conveniently, the natural type agrees with the principal type in negative positions, so it suffices to look at the domain of the natural type rather than computing the (generally larger) principal type.

As an example, let Γ be $y : (\mathcal{S}(c_1)\to b)\to b$. Then:

$$
\begin{array}{l}
\Gamma \rhd y\,(\lambda x{:}b.\,x) \Longleftrightarrow y\,(\lambda x{:}b.\,c_1) : b \\
\quad \textbf{because}\quad \Gamma \rhd y\,(\lambda x{:}b.\,x) \Downarrow y\,(\lambda x{:}b.\,x) \\
\quad \textbf{and}\quad \Gamma \rhd y\,(\lambda x{:}b.\,c_1) \Downarrow y\,(\lambda x{:}b.\,c_1) \\
\quad \textbf{and}\quad \Gamma \rhd y\,(\lambda x{:}b.\,x) \longleftrightarrow y\,(\lambda x{:}b.\,c_1) \uparrow b \\
\qquad\quad \textbf{because}\quad \Gamma \rhd y \longleftrightarrow y \uparrow (\mathcal{S}(c_1)\to b)\to b \\
\qquad\quad \textbf{and}\quad \Gamma \rhd (\lambda x{:}b.\,x) \Longleftrightarrow (\lambda x{:}b.\,c_1) : \mathcal{S}(c_1)\to b \\
\qquad\qquad\quad \textbf{because}\quad \Gamma, x' : \mathcal{S}(c_1) \rhd (\lambda x{:}b.\,x)\,x' \Longleftrightarrow (\lambda x{:}b.\,c_1)\,x' : b \\
\qquad\qquad\qquad\quad \textbf{because}\quad \Gamma, x' : \mathcal{S}(c_1) \rhd (\lambda x{:}b.\,x)\,x' \Downarrow c_1 \\
\qquad\qquad\qquad\quad \textbf{and}\quad \Gamma, x' : \mathcal{S}(c_1) \rhd (\lambda x{:}b.\,c_1)\,x' \Downarrow c_1 \\
\qquad\qquad\qquad\quad \textbf{and}\quad \Gamma, x' : \mathcal{S}(c_1) \rhd c_1 \longleftrightarrow c_1 \uparrow b.
\end{array}
$$

The soundness of the equivalence algorithm follows for essentially the same reasons as the soundness of normalization.

PROPOSITION 5.1 (SOUNDNESS OF BINARY EQUIVALENCE)

(1) *If* $\Gamma \vdash A_1$, $\Gamma \vdash A_2$, *and* $\Gamma \rhd A_1 \Longleftrightarrow A_2$, *then* $\Gamma \vdash A_1 \equiv A_2$.
(2) *If* $\Gamma \vdash M_1 : A$, $\Gamma \vdash M_2 : A$, *and* $\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$, *then* $\Gamma \vdash M_1 \equiv M_2 : A$.
(3) *If* $\Gamma \vdash p_1 : A_1$, $\Gamma \vdash p_2 : A_2$, *and* $\Gamma \rhd p_1 \longleftrightarrow p_2 \uparrow A$, *then* $\Gamma \vdash p_1 \equiv p_2 : A$.

Again, completeness is more interesting. It would have been preferable to analyze the equivalence algorithm directly in the logical relations argument, but it is neither obviously symmetric nor transitive. However, we can we can reuse the soundness and completeness results for normalization to show this algorithm is correct.

LEMMA 5.2

(1) *If* $\Gamma \vdash M_1 \equiv M_2 : A$ *and* $\Gamma \rhd M_1 : A \Longrightarrow N$, *then* $\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$.
(2) *If* $\Gamma \vdash p_1 : B_1$, $\Gamma \vdash p_2 : B_2$, $\Gamma \rhd p_1 \longrightarrow p \uparrow A_1$, *and* $\Gamma \rhd p_2 \longrightarrow p \uparrow A_2$, *then* $\Gamma \rhd p_1 \longleftrightarrow p_2 \uparrow A_1$ *and* $\Gamma \vdash A_1 \equiv A_2$.
(3) *If* $\Gamma \vdash A_1 \equiv A_2$ *and* $\Gamma \rhd A_1 \Longrightarrow B$, *then* $\Gamma \rhd A_1 \Longleftrightarrow A_2$

PROOF. By simultanous induction on the normalization assumptions (the first such assumption in the case of Part (2)).

(1) Assume $\Gamma \vdash M_1 \equiv M_2 : A$ and $\Gamma \rhd M_1 : A \Longrightarrow N$.
 —Case: $A = T$, $\Gamma \rhd M_1 \Downarrow p_1$, and $\Gamma \rhd p_1 \longrightarrow N \uparrow b$. By Proposition 3.10 and Rule (27) and Corollary 4.18 we know that $\Gamma \rhd M_2 \Downarrow p_2$ and $\Gamma \rhd p_2 \longrightarrow N \uparrow b$. By Propositions 4.8 and 3.10, $\Gamma \vdash p_1 : b$ and $\Gamma \vdash p_2 : b$. Inductively by Part (2), we have $\Gamma \rhd p_1 \longleftrightarrow p_2 \uparrow A_1$. Therefore, by definition of the equivalence algorithm, $\Gamma \rhd M_1 \Longleftrightarrow M_2 : b$.
 —Case: $A = \mathcal{S}(M_3)$. Then $\Gamma \rhd M_1 \Longleftrightarrow M_2 : \mathcal{S}(M_3)$ by definition of the equivalence algorithm.
 —Case: $A = \Pi x{:}A'.\,A''$ and $\Gamma, x : A' \rhd M_1\,x : A'' \Longrightarrow N''$. By Proposition 3.10 and inversion of Rule (5) we have $\Gamma \vdash A'$. Thus, $\Gamma, x : A' \vdash x \equiv x : A'$ and so by admissible Rule (31), we have $\Gamma, x : A' \vdash M_1\,x \equiv M_2\,x$:

A''. Inductively by Part (1) we have $\Gamma, x : A' \rhd M_1\, x \Longleftrightarrow M_2\, x : A''$. Therefore, by definition of the equivalence algorithm we have $\Gamma \rhd M_1 \Longleftrightarrow M_2 : \Pi x{:}A'.\, A''$.

—Case: $A = \Sigma x{:}A'.\, A''$ and $\Gamma \rhd \pi_1 M_1 : A' \Longrightarrow N'$, and $\Gamma \rhd \pi_2 M_1 : [\pi_1 M_1/x]A'' \Longrightarrow N''$, and $N = \langle N', N''\rangle$. By Rules (32) and (33), we have $\Gamma \vdash \pi_1 M_1 \equiv \pi_1 M_2 : A'$ and $\Gamma \vdash \pi_2 M_1 \equiv \pi_2 M_2 : [\pi_1 M_1/x]A''$. Thus, inductively by Part (2) (twice), we have $\Gamma \rhd \pi_1 M_1 \Longleftrightarrow \pi_1 M_2 : A'$ and $\Gamma \rhd \pi_2 M_1 \Longleftrightarrow \pi_2 M_2 : [\pi_1 M_1/x]A''$. Therefore, by definition of the equivalence algorithm, we have $\Gamma \rhd M_1 \Longleftrightarrow M_2 : \Sigma x{:}A'.\, A''$.

(2) Assume $\Gamma \vdash p_1 : B_1$ and $\Gamma \vdash p_2 : B_2$, and $\Gamma \rhd p_1 \longrightarrow p \uparrow A_1$ and $\Gamma \rhd p_2 \longrightarrow p \uparrow A_2$. Path normalization is shape-preserving, so p_1 and p_2 and p must have the same structural form.

—The base cases where p_1 and p_2 and p are the same variable or same constant all follow by definition of the equivalence algorithm.

—Case: $p_1 = \pi_1 p_1'$, and $p_2 = \pi_1 p_1'$, $p = \pi_1 p'$ $\Gamma \rhd p_1' \longrightarrow p' \uparrow \Sigma x{:}A_1'.\, A_1''$, and $\Gamma \rhd p_2' \longrightarrow p' \uparrow \Sigma x{:}A_2'.\, A_2''$, where $A_1 = A_1'$ and $A_2 = A_2'$. By Proposition 3.1 there exist B_1' and B_2' such that $\Gamma \vdash p_1 : B_1'$ and $\Gamma \vdash p_2 : B_2'$. Inductively by Part (2) we have $\Gamma \rhd p_1' \longleftrightarrow p_2' \uparrow \Sigma x{:}A_1'.\, A_1''$ and $\Gamma \vdash \Sigma x{:}A_1'.\, A_1'' \equiv \Sigma x{:}A_2'.\, A_2''$. Therefore, $\Gamma \vdash A_1' \equiv A_2'$, and by definition of the equivalence algorithm, $\Gamma \rhd \pi_1 p_1' \longleftrightarrow \pi_1 p_2' \uparrow A_1'$

—Case: $p_1 = \pi_2 p_1'$, and $p_2 = \pi_2 p_1'$, $p = \pi_2 p'$, $\Gamma \rhd p_1' \longrightarrow p' \uparrow \Sigma x{:}A_1'.\, A_1''$, $\Gamma \rhd p_2' \longrightarrow p' \uparrow \Sigma x{:}A_2'.\, A_2''$, $A_1 = [\pi_1 p_1'/x]A_1''$, and $A_2 = [\pi_1 p_2'/x]A_2''$. By Proposition 3.1, there exist B_1' and B_2' such that $\Gamma \vdash p_1 : B_1'$ and $\Gamma \vdash p_2 : B_2'$. Inductively, by Part (2), we have $\Gamma \rhd p_1' \longleftrightarrow p_2' \uparrow \Sigma x{:}A_1'.\, A_1''$ and $\Gamma \vdash \Sigma x{:}A_1'.\, A_1'' \equiv \Sigma x{:}A_2'.\, A_2''$. Then, $\Gamma \rhd \pi_2 p_1' \longleftrightarrow \pi_2 p_2' \uparrow [\pi_1 p_1'/x]A_1''$. Finally, using Propositions 5.1 and 3.14 $\Gamma \vdash [\pi_1 p_1'/x]A_1'' \equiv [\pi_1 p_2'/x]A_2''$.

—Case: $p_1 = p_1'\, M_1'$, $p_2 = p_2'\, M_2'$, $p = p'\, M'$, $\Gamma \rhd p_1' \longrightarrow p' \uparrow \Pi x{:}A_1'.\, A_1''$, $\Gamma \rhd p_2' \longrightarrow p' \uparrow \Pi x{:}A_2'.\, A_2''$, $\Gamma \rhd M_1' : A_1' \Longrightarrow M'$, $\Gamma \rhd M_2' : A_2' \Longrightarrow M'$, $A_1 = [M_1'/x]A_1''$, and $A_2 = [M_2'/x]A_2''$. By Proposition 3.1, there exist B_1' and B_2' such that $\Gamma \vdash p_1 : B_1'$ and $\Gamma \vdash p_2 : B_2'$. Inductively by Part (2), $\Gamma \rhd p_1' \longleftrightarrow p_2' \uparrow \Pi x{:}A_1'.\, A_1''$ and $\Gamma \vdash \Pi x{:}A_1'.\, A_1'' \equiv \Pi x{:}A_2'.\, A_2''$. and so by inversion of Rule (14), $\Gamma \vdash A_1' \equiv A_2'$. Using Propositions 4.8 and 3.11, $\Gamma \vdash M_1' \equiv M_2' : A_1'$, so by Proposition 3.14, $\Gamma \vdash [M_1'/x]A_1'' \equiv [M_1'/x]A_2''$. Inductively by Part (1) $\Gamma \rhd M_1' \Longleftrightarrow M_2' : A_1'$. Therefore, by definition of the equivalence algorithm we have $\Gamma \rhd p_1'\, M_1' \longleftrightarrow p_2'\, M_2' \uparrow [M_1'/x]A_1''$.

(3) Assume $\Gamma \vdash A_1 \equiv A_2$ and $\Gamma \rhd A_1 \Longrightarrow B$.

—Case: $A_1 = B = b$. Then $A_2 = b$, and $\Gamma \rhd b \Longleftrightarrow b$.

—Case: $A_1 = \mathcal{S}(M_1)$ and $B = \mathcal{S}(N)$ and $\Gamma \rhd M_1 : b \Longrightarrow N$. Then $A_2 = \mathcal{S}(M_2)$ with $\Gamma \vdash M_1 \equiv M_2 : b$. Inductively, by Part (1), $\Gamma \rhd M_1 \Longleftrightarrow M_2 : b$. Therefore, $\Gamma \rhd \mathcal{S}(M_1) \Longleftrightarrow \mathcal{S}(M_2)$.

—Case: $A_1 = \Pi x{:}A_1'.\, A_1''$ and $B = \Pi x{:}B'.\, B''$ with $\Gamma \rhd A_1' \Longrightarrow B'$ and $\Gamma, x : A_1' \rhd A_1'' \Longrightarrow B''$. Then, $A_2 = \Pi x{:}A_2'.\, A_2''$ with $\Gamma \vdash A_1' \equiv A_2'$ and $\Gamma, x : A_1' \vdash A_1'' \equiv A_2''$. Inductively by Part (3), $\Gamma \rhd A_1' \Longleftrightarrow A_2'$ and $\Gamma, x : A_1' \rhd A_1'' \Longleftrightarrow A_2''$. Therefore, $\Gamma \rhd \Pi x{:}A_1'.\, A_1'' \Longleftrightarrow \Pi x{:}A_2'.\, A_2''$.

—Case: $A_1 = \Sigma x{:}A_1'.\, A_1''$. Same as the previous case. □

Type validity

$\Gamma \rhd b$	
$\Gamma \rhd \mathcal{S}(M)$	if $\Gamma \rhd M \Leftarrow b$
$\Gamma \rhd \Pi x{:}A'.\, A''$	if $\Gamma \rhd A'$ and $\Gamma, x : A' \rhd A''$.
$\Gamma \rhd \Sigma x{:}A'.\, A''$	if $\Gamma \rhd A'$ and $\Gamma, x : A' \rhd A''$, where $x \notin \mathrm{dom}(\Gamma)$

Subtyping

$\Gamma \rhd b \le b$	always
$\Gamma \rhd \mathcal{S}(M) \le b$	always
$\Gamma \rhd \mathcal{S}(M_1) \le \mathcal{S}(M_2)$	if $\Gamma \rhd M_1 \Longleftrightarrow M_2 : b$.
$\Gamma \rhd \Pi x{:}A_1'.\, A_1'' \le \Pi x{:}A_2'.\, A_2''$	if $\Gamma \rhd A_2' \le A_1'$ and $\Gamma, x : A_2' \rhd A_1'' \le A_2''$, where $x \notin \mathrm{dom}(\Gamma)$.
$\Gamma \rhd \Sigma x{:}A_1'.\, A_1'' \le \Sigma x{:}A_2'.\, A_2''$	if $\Gamma \rhd A_1' \le A_2'$ and $\Gamma, x : A_1' \rhd A_1'' \le A_2''$, where $x \notin \mathrm{dom}(\Gamma)$.

Fig. 12. Algorithms for types.

COROLLARY 5.3 (COMPLETENESS FOR EQUIVALENCE)

(1) *If* $\Gamma \vdash M_1 \equiv M_2 : A$*, then* $\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$.
(2) *If* $\Gamma \vdash A_1 \equiv A_2$*, then* $\Gamma \rhd A_1 \Longleftrightarrow A_2$.

Finally, the binary equivalence algorithms are terminating.

COROLLARY 5.4 (DECIDABILITY OF BINARY EQUIVALENCE)

(1) *If* $\Gamma \vdash M_1 : A$ *and* $\Gamma \vdash M_2 : A$*, then* $\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$ *is decidable.*
(2) *If* $\Gamma \vdash p_1 : A$ *and* $\Gamma \vdash p_2 : A$*, then* $\Gamma \rhd M_1 \longleftrightarrow M_2 \uparrow A$ *is decidable.*
(3) *If* $\Gamma \vdash A_1$ *and* $\Gamma \vdash A_2$*, then* $\Gamma \rhd A_1 \Longleftrightarrow A_2$ *is decidable.*

PROOF. The maximum number of $\Longleftrightarrow$ and $\longleftrightarrow$ steps that can occur is bounded by the number of $\Longrightarrow$ and and $\longrightarrow$ steps required to normalize M_1, p_1, or A_1 as appropriate. Finally, we can rule out the only other possibility of non-termination by observing that the algorithms preserve well-formedness, and so head normalizations will always terminate. □

6. DECIDING OTHER JUDGMENTS

Finally, we consider the other type and term-level judgments. Figure 12 gives algorithms for determining type validity and subtyping. Each is specified as a deterministic set of inference rules.

The algorithmic *type validity* judgment

$$\Gamma \rhd A$$

models the declarative type validity judgment $\Gamma \vdash A$. Viewed as an algorithm, this takes a well-formed context Γ and a type A and determines whether there is a proof of $\Gamma \vdash A$. For any conclusion, at most one rule could apply; there is one rule for each syntactic form that A might have. Since the premises involve syntactically smaller kinds and terms, proof search for this judgment must terminate and so the judgment is decidable.

The algorithmic *subtyping* judgment

$$\Gamma \rhd A_1 \le A_2$$

models the declarative subtyping judgment $\Gamma \vdash A_1 \le A_2$. As an algorithm, given types satisfying $\Gamma \vdash A_1$ and $\Gamma \vdash A_2$ it determines whether there is a

Type synthesis

$\Gamma \rhd c_1 \rightrightarrows \mathcal{S}(c_1)$	
$\Gamma \rhd x \rightrightarrows \mathcal{S}_{\Gamma(x)}(x)$	if $x \in \mathrm{dom}(\Gamma)$.
$\Gamma \rhd \lambda x{:}A'.\, M \rightrightarrows \Pi x{:}A'.\, A''$	if $\Gamma \rhd A'$ and $\Gamma, x : A' \rhd M \rightrightarrows A''$.
$\Gamma \rhd M\, M' \rightrightarrows [M'/x]A''$	if $\Gamma \rhd M \rightrightarrows \Pi x{:}A'.\, A''$ and $\Gamma \rhd M' \leftleftarrows A'$.
$\Gamma \rhd \langle M', M'' \rangle \rightrightarrows A' \times A''$	if $\Gamma \rhd M' \rightrightarrows A'$ and $\Gamma \rhd M'' \rightrightarrows A''$.
$\Gamma \rhd \pi_1 M \rightrightarrows A'$	if $\Gamma \rhd M \rightrightarrows \Sigma x{:}A'.\, A''$
$\Gamma \rhd \pi_2 M \rightrightarrows [\pi_1 M/x]A''$	if $\Gamma \rhd M \rightrightarrows \Sigma x{:}A'.\, A''$

Type checking

$\Gamma \rhd M \leftleftarrows A$	if $\Gamma \rhd M \rightrightarrows B$ and $\Gamma \rhd B \leq A$.

Fig. 13. Algorithms for term validity.

proof $\Gamma \vdash A_1 \leq A_2$. The rules are syntax-directed and the premises involve syntactically smaller kinds, and we already know we can determine constructor equivalence for well-formed types (the correctness of the particular form of equivalence used here is shown below), so this judgment too is decidable.

Figure 13 shows the algorithms for term validity. The algorithmic *type synthesis* judgment

$$\Gamma \rhd M \rightrightarrows A$$

combines term validity checking with principal type synthesis. As an algorithm, given a well-formed context Γ and a term M it returns a principal type A of M if M is well-formed (i.e., if it can be given any type at all) and fails otherwise.

Because all well formed terms have principal types, it is easy to define a algorithmic *type checking* judgment

$$\Gamma \rhd M \leftleftarrows A.$$

which directly models the term validity checking. Given a context and type satisfying $\Gamma \vdash A$ and term M, this algorithm determines whether $\Gamma \vdash M : A$ holds. The rules are again syntax-directed, and the premises involve only strict subterms.

THEOREM 6.1 (SOUNDNESS OF REMAINING ALGORITHMS)

(1) *If* $\Gamma \vdash A_1$, $\Gamma \vdash A_2$, *and* $\Gamma \rhd A_1 \leq A_2$, *then* $\Gamma \vdash A_1 \leq A_2$.
(2) *If* $\Gamma \vdash ok$ *and* $\Gamma \rhd A$, *then* $\Gamma \vdash A$.
(3) *If* $\Gamma \vdash ok$ *and* $\Gamma \rhd M \rightrightarrows A$, *then* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M \Uparrow A$.
(4) *If* $\Gamma \vdash A$ *and* $\Gamma \rhd M \leftleftarrows A$, *then* $\Gamma \vdash M : A$.

PROOF. By (simultaneous) induction on proofs of the algorithmic judgments (i.e., by induction on the execution of the algorithms). □

THEOREM 6.2 (COMPLETENESS)

(1) *If* $\Gamma \vdash M_1 \equiv M_2 : A$, *then* $\Gamma \rhd M_1 \Longleftrightarrow M_2 : A$.
(2) *If* $\Gamma \vdash A$, *then* $\Gamma \rhd A$.
(3) *If* $\Gamma \vdash A_1 \leq A_2$, *then* $\Gamma \rhd A_1 \leq A_2$.
(4) *If* $\Gamma \vdash A_1 \equiv A_2$, *then* $\Gamma \rhd A_1 \Longleftrightarrow A_2$.
(5) *If* $\Gamma \vdash M : A$, *then* $\Gamma \rhd M \rightrightarrows B$ *and* $\Gamma \vdash M \Uparrow B$.
(6) *If* $\Gamma \vdash M : A$, *then* $\Gamma \rhd M \leftleftarrows A$.

PROOF. By the completeness of algorithmic equivalence and induction on derivations. □

7. CONCLUSION

7.1 Related Work

7.1.1 *Singletons and Definitions in Type Systems.* The main previous study of singleton types in the literature is due to Aspinall [1995, 1997]. He studied term equivalence in a calculus $\lambda_{\leq\{\}}$ containing singleton types, dependent function types, and β-equivalence. Labeled singletons are primitive notions in his system; in the absence of η-equivalence the encoding of Section 2.3 is not possible. He conjectured that equivalence in $\lambda_{\leq\{\}}$ was decidable, but gave no proof or algorithm.

Aspinall's system included a limited form of extensionality when comparing two λ-abstractions, enough to make equivalence depend on classifier as well as the typing context. More recently, Courant [2002] defined a language with labeled singletons but no extensionality properties at all, so that equivalence depended on definitions (singletons) in the typing context but not on the classifying kind. This permitted a more straightforward rewriting-based approach to deciding equivalence.

Crary has made use of singleton types and singleton kinds in several contexts. His thesis [Crary 1998] includes a system whose kind system extends the one presented here with subtyping and power kinds, and he conjectured that type equivalence and type checking were decidable. Later, he used an extremely simple form of singleton type (with no elimination rule or subtyping) in order to prove parametricity results [Crary 1999]. As one example, he shows that any function f of type $\forall\alpha.\alpha\to\alpha$ must act as a the identity because

$$f(\mathbf{S}(v : \tau))(v) : \mathbf{S}(v : \tau)$$

where $\mathbf{S}(v : \tau)$ is the type classifying only the value v of type τ; by soundness of the type system any value returned by this application must be equal to v. Furthermore, evaluation in his system does not depend upon type arguments to polymorphic functions, so f must act as an identity function for every argument of every type.

There are other ways to support equational information in a type system besides singleton types [Stone 2005]. Severi and Poll [1994] study confluence and normalization of $\beta\delta$-reduction for a pure type system with definitions (let bindings), where δ is the replacement of an occurrence of a variable with its definition. In this system, the typing context contains both the type for each variable, and an optional definition. This calculus contains no notion of partial definition, no subtyping, and cannot express constraints on function arguments. This approach may be sufficient to represent information needed for cross-module inlining (particularly when based upon the lambda-splitting work of Blume and Appel [1997] and Blume [1997]), but it cannot model sharing constraints or definitions in a modular framework (where only some parts of a module have known definition).

Type theoretic studies of the SML module system have been studied by Harper and Lillibridge under the name of *translucent sums* [Harper and Lillibridge 1994; Lillibridge 1997] in which modules are first-class values, and by Leroy [1994] under the name of *manifest types* in which modules are second-class. These two systems are essentially similar: the calculus includes module constructs, and corresponding signatures; as in Standard ML the type components of signatures may optionally specify definitions. The key difference from $\lambda_{\leq}^{\Pi\Sigma S}$ is that type definitions are specified at the *type* level, rather than at the *kind* level. Because of this, type equivalence does depend on the typing context but not on the (unique) classifying kind. Type checking for translucent sums is undecidable (although type equivalence is decidable). No analogous result is known for manifest types; modules lack most-specific signatures, prohibiting standard methods for type checking.

A very powerful construct is the I-type of Martin-Löf's extensional type theory [Martin-Löf 1984; Hofmann 1995]. A term of type $I(e_1, e_2)$ represents a *proof* that e_1 and e_2 are equivalent. This can lead to undecidable type checking very quickly, as one can use this to add arbitrary equations as assumptions in the typing context.

The language Dylan [Shalit 1996] contains "singleton types", but these are checked only at run-time (essentially pointer-equality) to resolve dynamic overloading.

7.1.2 *Decidability of Equivalence and Type Checking.* This algorithm was (indirectly) inspired by Coquand's approach to $\beta\eta$-equivalence for a type theory with Π types and one universe [Coquand 1991]. Coquand's algorithm directly decides equivalence, rather than being defined in terms of reduction or normalization. However, unlike the type system studied by Coquand, $\lambda_{\leq}^{\Pi\Sigma S}$ terms cannot be compared in isolation; equivalence depends both on the typing context and on the type at which the terms are being compared. Thus, where Coquand maintains a set of bound variables, our algorithm maintains a full typing context. Similarly, Coquand's algorithm uses the shapes of terms to guide the algorithm where our algorithm maintains the type. (For example, Coquand's would check for one of the terms being lambda-abstraction, whereas our algorithm would check for comparison at a Π type.) In our system, it is technically more convenient to work with normalization, rather than direct equivalence, because asymmetries in the natural binary equivalence algorithm make it difficult to show directly that it is symmetric and transitive. (These properties were immediately evident in Coquand's case.)

There are also similarities between our work and that of Compagnoni and Goguen [2003], who also use a normalization algorithm and Kripke logical relations argument for proving decidability of a subtyping algorithm for $\mathcal{F}_{\leq}^{\omega}$, a variant of $F_{<:}^{\omega}$ with higher-order subtyping and the kernel Fun Rule [Cardelli and Wegner 1985] for quantifier subtyping. They largely ignore the term level of $F_{<:}^{\omega}$. Roughly speaking, our terms and types (dropping singletons and Σ) would correspond to their types and kinds if we added an uninterpreted *term* constant $\forall$ (e.g., to represent polymorphic types using higher-order abstract syntax), defined a relation $\preceq$ on our *terms*, and allowed $\preceq$ bounds on the domains of λ, and Π. In both $\lambda_{\leq}^{\Pi\Sigma S}$ and $F_{\leq}^{\omega}$ there are no variables at the topmost level (i.e., no variables whose value can be a Π) and so there arise no issues of impredicativity.

There are a number of technical differences between the proofs. Their normalization algorithm is not driven at all by the classifier (types have unique kinds and the rules are entirely determined by the shapes of the types being normalized and/or the shapes of their normal form; the kind seems naturally considered an output of the algorithm rather than an input), and in fact Compagnoni and Goguen show that their normalization algorithm implements and is invariant under untyped β-reduction. This allows a direct proof that their logical relations are closed under normalization. We do not have a corresponding rewrite rule and so must take a slightly different approach. It would be interesting, however, to try to combine features of our normalization algorithm and set-based logical relations with the core of their subtyping algorithm to obtain the decidability of subtyping in $F_{\leq}^{\omega}$ extended with singletons.

In previous work [Stone and Harper 2000] we used a different variant of logical relations involving pairs rather than sets. The resulting definitions were quite verbose (due to the lack of a natural transitivity property) and led to a very large number of conditions to be checked in the proofs. Our generalization here from ordered pairs to sets allows the logical relations and therefore the proof of completeness to be substantially simpler and more elegant.

Crary [2000] has used the correctness of our algorithms to show that a language with singleton kinds can be translated into a language without, in a fashion which preserves well typedness. Although one can certainly "substitute away" all of the definitions induced by singletons, because of partial definitions the resulting term might still refer to variables classified by singletons. The fact that all singleton kinds can thereafter be erased is a nontrivial strengthening property. Crary obtains this property by working with the (more tractable) algorithmic form of term equivalence for $\lambda_{\leq}^{\Pi\Sigma S}$.

Recently a few researchers have been studying more direct approaches to singletons using very similar η and definition expansions. Coquand et al. [2003] starts from a PER model to obtain a similar language of labeled singleton types in which term equivalence can be decided by checking β-equivalence of η-expanded and definition-expanded terms. This is a less practical method than the binary equivalence algorithm of Section 5, but correctness of our algorithm might be derivable. More recently, Goguen [2005] has given an equivalence algorithm for a system more similar to ours, having labeled singletons but no subtyping.

Several systems in which equivalence depends upon the typing context were already mentioned. However, there appear to be relatively few decidability results for other lambda calculi with typing-context-sensitive or classifier-sensitive equivalences, perhaps because standard techniques of rewriting to normal form are difficult to apply. Many calculi include subtyping but not subkinding; in such cases either only type equivalence is considered (which is independent of subtyping) or else term equivalence is not affected by subtyping and hence can be computed in a context-free manner.

One exception is the work of Curien and Ghelli [1994], who proved the decidability of term equivalence in $F_{\leq}$ with $\beta\eta$-reduction and a Top type. Because their Top type is both terminal and maximal, equivalence depends on both the typing context and the type at which terms are compared. They eliminate context-sensitivity by inserting explicit coercions to mark uses of subsumption and then give a rewriting strategy for the calculus with coercions; in total, their proof involves translations among three different typed λ-calculi.

Our language is dependently typed, but has no polymorphism and no type variables. It would be interesting to see if the approach used for $\lambda_{\leq}^{\Pi\Sigma S}$ could be applied to their source language, avoiding the use of translations. Although adapting the equivalence algorithm seems easy, an impredicative calculus would require an extension of the logical relations, for example, as done by Girard [1972].

7.2 Summary

In this article, we have presented the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus, which models the constructors and kinds of the internal language used by the TILT compiler for standard ML. We studied the equational and proof-theoretic properties of the $\lambda_{\leq}^{\Pi\Sigma S}$ calculus, and have shown that type checking is decidable.

We have presented algorithms for implementing type checking; they form the basis of the implementation of the type checker in the TILT compiler [Petersen et al. 2000]. The equivalence algorithms employ a useful type-directed framework. This is extremely well-suited for any case in which equivalence is dependent upon the classifier. Examples of other such languages include those with terminal types (where all terms of this type are equal), or calculi with records and width subtyping (where equivalence of two records depends only on the equivalence of the subset of fields mentioned in the classifying record type). This approach can even be used for efficiency in the absence of subkinding and singletons [Harper and Pfenning 1999].

The correctness proofs employ an new variant of Kripke logical relation, working directly with (subsets of) equivalence classes instead of the more usual binary relations. This permits a very straightforward proof of correctness for the equivalence algorithms. We have found the logical relations approach to proving completeness to be remarkably robust under changes to the definition of the equational theory; even the addition of type analysis constructs [Harper and Morrisett 1995] requires few changes [Stone 2000].

ACKNOWLEDGMENTS

We are grateful to Lars Birkedal for originally suggesting we not limit ourselves to traditional formulations of Kripke logical relations. We also thank Karl Crary, Derek Dreyer, Peter Lee, John Reynolds, and Jon Riecke for particularly detailed comments on earlier versions of this work, Perry Cheng, Mark Lillibridge, Leaf Petersen, Frank Pfenning, and Rick Statman for many helpful discussions, and Andreas Rossberg the anonymous referees for their comments.

REFERENCES

ASPINALL, D. 1995. Subtyping with singleton types. In *Proceedings of the Computer Science Logic (CSL '94)*. Lecture Notes in Computer Science, vol. 933, Springer-Verlag, New York.

ASPINALL, D. 1997. Type systems for modular programs and specifications. Ph.D. dissertation, Department of Computer Science, University of Edinburgh.

ASPINALL, D. 2000. Subtyping with power types. In *Proceedings of the Computer Science Logic (CSL 2000)*. Lecture Notes in Computer Science, vol. 1862, Springer-Verlag, New York. 156–171.

BLUME, M. 1997. Hierarchical modularity and intermodule optimization. Ph.D. dissertation. Princeton University, Princeton, NJ.

BLUME, M. AND APPEL, A. W. 1997. Lambda-splitting: A higher-order approach to cross-module optimizations. In *Proceedings 1997 ACM International Conference on Functional Programming (ICFP '97)*. 112–124.

CARDELLI, L. AND WEGNER, P. 1985. On understanding types, data abstraction and polymorphism. *ACM Comput. Surv. 17*, 4, 471–522.

COMPAGNONI, A. AND GOGUEN, H. 2003. Typed operational semantics for higher-ordersubtyping. *Info. Comput. 184*, 2 (Aug.), 242–297.

COQUAND, T. 1991. An algorithm for testing conversion in type theory. In *Logical frameworks*, G. Huet and G. Plotkin, Eds. Cambridge University Press, Cambridge, MA, 255–277.

COQUAND, T., POLLACK, R., AND TAKEYAMA, M. 2003. A logical framework with dependently typed records. In *Proceedings of the Typed Lambda Calculi and Applications (TLCA 2003)*. 105–119. (Available as LNCS 2701). Lecture Notes in Computer Science, vol. 2701, Springer-Verlag, New York.

COURANT, J. 2002. Strong normalization with singleton types. In *Proceeedings of the Second Workshop on Intersection Types and Related Systems (ITRS '02)*. ENTCS, vol. 70.

CRARY, K. 1999. A simple proof technique for certain parametricity results. In *Proceedings of the 1999 ACM International Conference on Functional Programming (ICFP '99)*. ACM, New York, 82–89.

CRARY, K. 2000. Sound and complete elimination of singleton kinds. Tech. Rep. CMU-CS-00-104, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

CRARY, K. 2005. Logical relations and a case study in equivalence checking. In *Advanced Topics in Types and Programming Languages*, B. C. Pierce, Ed. MIT Press, Cambridge, MA.

CRARY, K. F. 1998. Type-theoretic methodology for practical programming languages. Ph.D. dissertation, Department of Computer Science, Cornell University.

CURIEN, P.-L. AND GHELLI, G. 1994. Decidability and confluence of $\beta\eta\mathrm{top}_{\leq}$ reduction in $F_{\leq}$. *Info. Comput.* 1/2, 57–114.

GIRARD, J. 1972. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Ph.D. dissertation, Université Paris 7.

GOGUEN, H. 1994. A typed operational semantics for type theory. Ph.D. dissertation, Department of Computer Science, University of Edinburgh. (Available as Technical Report ECS-LFCS-94-304).

GOGUEN, H. 2005. A syntactic approach to eta equality in type theory. In *Proceedings of the 32nd ACM Symposium on Principles of Programming Languages (POPL '05)*. ACM, New York.

HARPER, R. AND LILLIBRIDGE, M. 1994. A type-theoretic approach to higher-order modules with sharing. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL '94)*. ACM, New York, 123–137.

HARPER, R., MITCHELL, J. C., AND MOGGI, E. 1990. Higher-order modules and the phase distinction. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL '90)*. ACM, New York, 341–354.

HARPER, R. AND MORRISETT, G. 1995. Compiling polymorphism using intensional type analysis. In *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages (POPL '95)*. ACM, New York, 130–141.

HARPER, R. AND PFENNING, F. 1999. On equivalence and canonical forms in the LF type theory. In *Proceedings of the Workshop on Logical Frameworks and Meta-Languages*. Extended version available as Tech. Rep. CMU-CS-99-159.

HOFMANN, M. 1995. Extensional concepts in intensional type theory. Ph.D. dissertation, Edinburgh LFCS. (Available as Edinburgh LFCS Technical Report ECS-LFCS-95-327).

LEROY, X. 1994. Manifest types, modules, and separate compilation. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL '94)*. ACM, New York, 109–122.

Leroy, X. 1995. Applicative Functors and Fully Transparent Higher-Order Modules. In *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages (POPL '95)*. ACM, New York, 142–153.

Lillibridge, M. 1997. Translucent Sums: A Foundation for Higher-Order Module Systems. Ph.D. dissertation, School of Computer Science, Carnegie Mellon University. (Available as Technical Report) CMU-CS-97-122.

Martin-Löf, P. 1984. *Intuitionistic Type Theory*. Bibliopolis-Napoli.

Minamide, Y., Morrisett, G., and Harper, R. 1996. Typed closure conversion. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages (POPL '96)*. ACM, New York, 271–283.

Morrisett, G., Walker, D., Crary, K., and Glew, N. 1997. From system F to typed assembly language. Tech. Rep. TR97-1651, Department of Computer Science, Cornell University.

Petersen, L., Cheng, P., Harper, R., and Stone, C. 2000. Implementing the TILT internal language. Tech. Rep. CMU-CS-00-180, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Severi, P. and Poll, E. 1994. Pure Type Systems with definitions. In *Proceedings of the Logical Foundations of Computer Science '94*. Lecture Notes in Computer Science, vol. 813, Springer-Verlag, New York.

Shalit, A. 1996. *The Dylan Reference Manual: The Definitive Guide to the New Object-Oriented Dynamic Language*. Addison-Wesley, Reading, MA.

Shao, Z. 1998. Typed Cross-Module Compilation. In *Proceedings of the 1998 ACM International Conference on Functional Programming (ICFP '98)*. ACM, New York, 141–152.

Stone, C. A. 2000. Singleton kinds and singleton types. Ph.D. dissertation. School of Computer Science, Carnegie Mellon University. Pittsburgh, PA. (Available as Technical Report CMU-CS-00-153.)

Stone, C. A. 2005. Type definitions. In *Proceedings of the Advanced Topics in Types and Programming Languages*, B. C. Pierce, Ed. MIT Press, Cambridge, MA.

Stone, C. A. and Harper, R. 2000. Decidable type equivalence with singleton kinds. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL '00)*. ACM, New York, 214–227.

Stone, C. A. and Harper, R. 2004. Extensional equivalence and singleton types. Tech. Rep. HMC-CS-04-100, Computer Science Department, Harvey Mudd College.

Tarditi, D., Morrisett, G., Cheng, P., Stone, C., Harper, R., and Lee, P. 1996. TIL: A Type-Directed Optimizing Compiler for ML. In *Proceedings of the ACM 1996 Conference on Programming Language Design and Implementation (PLDI '96)*. ACM, New York, 181–192.

Received August 2002; revised January 2004; accepted July 2004