Proof Irrelevance and Strict Definitions in a Logical Framework

Jason Reed

June 2002

CMU-CS-02-153

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements of the senior thesis program.

Advisor
Frank Pfenning

Abstract

The addition of proof irrelevant types to LF permits more expressive encodings and a richer (but still decidable) definitional equality. These types can also be used to maintain invariants concerning the interchangeability of subproofs of large proofs, which can be useful for proof compression for proof-carrying code. We investigate the metatheoretic properties of this language extension and reconcile it with the notion of strictness, a property of notational definitions which can improve implementation efficiency of proofchecking.

This research was supported in part by the National Science Foundation under grant CCR-9988281

Keywords: proof irrelevance, type theory, LF, logical frameworks, dependent types, strictness, proof compression

1 Introduction

The notion of *proof irrelevance* is considering any two proofs of a given proposition equal. Although it can (for example, in [Bar98]) be of some use to deal with a system which is 'globally proof irrelevant' where all types have at most one inhabitant, we consider, following Pfenning [Pfe01], the addition of a *proof irrelevant modality* to the logical framework of LF. That is, individual functions can be marked as treating certain arguments as proof irrelevant, and it can be mechanically checked that they fulfill their promise not to care about the structure of that argument.

This paper contributes two things: First, we present a variation of the inference rules in [Pfe01] which satisfies a functionality lemma (Lemma 4.10). Second, we a generalize the notion of *strict definitions* and the strictness-checking algorithm to the new type theory. We have also created a prototype implementation of a version of Twelf which is able to type-check and test for equality and strictness terms involving irrelevance.

The remainder of the paper is organized as follows: Section 2 sketches some background ideas and techniques. Section 3 describes proof irrelevance at a high level and some applications of it. In Sections 4 and 5 we give in full the basic theory, and show that all judgments are decidable. Finally, Section 6 contains the generalization of strictness to proof irrelevance and proves its correctness.

1.1 Related Work

The NuPRL System [C+86] has a notion of *squash types* which erases computational content in a manner similar to irrelevant types. However, the definitional equality of NuPRL is not decidable; it similarly possible to define in Twelf an equality function and add constructors which simulate irrelevance, forcing the user to carry around proofs of equality. The advantage of putting irrelevance into a language in such a way that retains a decidable equality is that the implementation can easily check without a full theorem-prover whether terms involving irrelevance are equal or well-typed.

Awodey and Bauer [AB01] propose a different type-theoretic foundation of 'erased' types, which instead of having a distinct judgment $\Gamma \vdash M \div A$ has a type constructor $[\text{—}]$, and would instead type $\Gamma \vdash M : [A]$. While it allows certain function types the present system does not (for instance, while their $[A] \to B$ corresponds to $\Pi x \div A.B$ here, their $A \to [B]$ appears to have no counterpart) it lacks certain properties desirable in a logical framework. In particular, the elimination rule appears to require a commuting conversion rule which makes a theory of canonical forms difficult.

1.2 Acknowledgements

I would like to thank Kaustuv Chaudhuri and Tom Murphy VII for helpful discussion and comments, Carsten Schürmann and Kevin Watkins for advice and information about implementation issues, and Frank Pfenning for introducing me to the subject and being an excellent advisor throughout the year.

2 Preliminaries

2.1 Logic and Representation

There is a close relationship between type systems and logics known as the Curry-Howard correspondence. In its most basic form, it comes very naturally from an (intuitionistic) understanding of what it means to be a proof of a complex proposition. If we know what counts as a proof of a proposition A, and what counts as a proof of B, then a proof of the proposition $A \wedge B$ ("A and B") ought to be a sort of package containing evidence for A on one side, and evidence for B on the other. Similarly, we might expect a proof of the proposition $A \Rightarrow B$ ("A implies B") to be a machine that somehow transforms a proof of A into a proof of B. With this explanation of the meaning of these two connectives, we have already touched some basic concepts of a programming language: organizing data into larger chunks, and applying functions to arguments to obtain some result.

Given the above discussion, it may not seem unlikely that the rules of logic for $\wedge$ and $\Rightarrow$ might have something to do with programming with simple data and functions, but it is somewhat surprising how exact a correspondence can be made. For this logic, and many others, there exists a statically typed programming language corresponding to it such that every type in the language corresponds to a proposition in the logic, and every well-typed program corresponds to a derivation of the proposition that corresponds to its type, and vice-versa. To summarize the correspondence so far:

$$\begin{array}{rcl} \text{Propositions} & \Leftrightarrow & \text{Types} \\ \text{Proofs} & \Leftrightarrow & \text{Programs} \\ A \wedge B & \Leftrightarrow & A \times B \\ A \Rightarrow B & \Leftrightarrow & A \to B \end{array}$$

In fact the language's notion of 'well-typed' is essentially a definition of a valid proof. For instance, a program which has type $A \to B$ must faithfully produce some piece of data of type B whenever it is given data of type A, just as a valid proof of $A \Rightarrow B$ is obliged to show that B holds given any proof of A.

2.2 Dependent Types

Armed with an understanding of how a propositional logic can be seen through programming-language glasses, it is natural to try to add quantification on the logic side and see what new types appear. For a concrete example, suppose that K is the set of books, $O(k)$ is the proposition that k is an old book for any $k \in K$, and $L(k)$ is the proposition that k is in the library, again for any $k \in K$. The statement of "every book in the library is old" in symbols is

$$\forall k \in K. L(k) \Rightarrow O(k)$$

What counts as a proof of this proposition? It should be a program f which takes a book $k \in K$, a proof that $L(k)$ holds, and produce a proof of $O(k)$. In other words, a function of two arguments, whose type we might try to write as $K \to (L\ k \to O\ k)$. The obvious problem with this definition, however, is that it doesn't explain where k comes from. The trouble is that $L\ k$, the type of proofs that k is in the library, *depends on* the book chosen from the type K. The correct typing of f therefore involves a new *dependent type constructor* Π to take the place of the $\forall$.

$$f : \Pi k\colon K.(L\ k \to O\ k)$$

The type $\Pi x\colon A.B$ is a generalization of the type $A \to B$ where the argument is given a name x that can be used somewhere in the result type B. If x does not appear in B at all, then $\Pi x\colon A.B$ has the same meaning as $A \to B$, and because of this most theories of dependent types include $A \to B$ only as a syntactic abbreviation of $\Pi x\colon A.B$.

There remains the question of exactly what status the predicates O and L have in the programming language. Clearly for any $k\colon K$ we must have that $O\ k$ and $L\ k$ are types, since they appear on the left and right side of an arrow. As is suggested by the notation chosen, we might reasonably treat them as functions which take some element of K, and return a type. One then speaks of O and L as *type families* indexed by the type K, which have the *kind*[1] $K \to \mathsf{type}$. We can extend the table above as follows:

Propositions	$\Leftrightarrow$	Types
$\cdots$	$\Leftrightarrow$	$\cdots$
$\forall x\colon A.B$	$\Leftrightarrow$	$\Pi x\colon A.B$
$P(x)$	$\Leftrightarrow$	$P\ x$

We can also extend the use of dependency to type families; The symbol Π is typically overloaded to also be the *dependent kind constructor*. We could express the proposition that a certain book from the library is checked out by a type family C with the kind $\Pi k\colon K.(L(k) \to \mathsf{type})$. In more plain language, this kind says that given a book k, and given a proof that k is in the library's collection, there is a type of proofs of the proposition that k is checked out.

Dependent types are in fact very common 'behind the scenes' in the definitions and reasoning of ordinary mathematics. As a more realistic example, we treat the type of the determinant operator in linear algebra. First we must establish what type a matrix has; we know that for every positive integers m, n we can consider the $m \times n$ matrices, so we write

$$\mathsf{mat} : (\Pi m\colon \mathsf{int}\,.\Pi n\colon \mathsf{int}\,.\mathsf{type})$$

to indicate the type family of matrices indexed by two integers m and n. We obtain a particular type by applying mat to particular integers. For instance, $\mathsf{mat}\ 3\ 4$ is the type of 3×4 matrices.

Now the determinant only exists for a square matrix, which is to say, for every n there is a determinant operator on $n \times n$ matrices. The type of det, then, is

$$\det : (\Pi n\colon \mathsf{int}\,.\Pi A\colon (\mathsf{mat}\ n\ n).\,\mathsf{real})$$

[1]Kinds are like types, except that they 'classify type families.' In other words, a valid object has a type, whereas a valid type family has a kind.

This type says that for each n which has type type, and each A which has type $\mathsf{mat}\ n\ n$ (which depends on the choice made for n) the expression det $n\ A$ has type real. One can also check that the $n \times n$ identity matrix is

$$I : (\Pi n{:}\, \mathsf{int}\, .\, \mathsf{mat}\ n\ n)$$

("For every n an integer, $I\ n$ is an $n \times n$ matrix") and matrix transposition can be typed as

$$\mathsf{transp} : (\Pi m{:}\, \mathsf{int}\, . \Pi n{:}\, \mathsf{int}\, . \Pi A{:}\, (\mathsf{mat}\ m\ n).\ \mathsf{mat}\ n\ m)$$

("For every m, n integers and A an $m \times n$ matrix, $\mathsf{transp}\ m\ n\ A$ is an $n \times m$ matrix") and so on.

2.3 The Twelf System

Twelf [PS99] is a *logical framework*, a system suitable for the formal description of programming languages and logics, and the automatic verification of properties of such descriptions. It is based on the type theory of LF [HHP93], which is a λ-calculus with dependent types as described above. We use the word *language* by itself in what follows to mean either a programming language or logic.

A language can be represented in LF by declaring a *signature* of type families and constants which describe how the syntactic elements of the language are created and organized. A signature is essentially a formalization of the sort of preliminary assumptions made in each example above, e.g., "suppose K is a type, and O is a type family indexed by K." There are powerful and elegant techniques for representing common structures in languages in the presence such signatures. In particular, bound variables in programs and hypothetical reasoning in logic can both be rendered by using higher-order function types — that is, by declaring constants in the signature which take a function as an argument. Assertions about a programming language as a whole — for example, that every well-typed expression evaluates to a value — can be encoded as type families, and their validity can be automatically checked by the Twelf system.

A ubiquitous example is the encoding of the untyped lambda calculus. If we declare a signature of three constants, $\Sigma =$

$$\begin{aligned}
&\mathsf{tm} : \mathsf{type}\\
&\mathsf{app} : \mathsf{tm} \to (\mathsf{tm} \to \mathsf{tm})\\
&\mathsf{lam} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm}
\end{aligned}$$

then every LF object of type tm corresponds uniquely to a term of the untyped lambda calculus, and in fact the variable binding of the object language (in this case the untyped lambda calculus) is implemented by variable binding in the representation language (LF). For instance, the lambda-calculus term $\lambda x.\lambda y.y\ x$ is represented in this signature as

$$\mathsf{lam}(\lambda x{:}\, \mathsf{tm}\, .\, \mathsf{lam}(\lambda y{:}\, \mathsf{tm}\, .\, \mathsf{app}\ \ y\ x))$$

This technique is known as higher-order abstract syntax [PE98].

It is worth noting that declaring a constant $c : A$ (resp. constant type family $a : K$) in a signature is very different from observing, as we did with the examples from linear algebra, that a certain already-defined function c (resp. type family a) has type A (resp. kind K). A constant declaration creates a fresh object with the given type, and, much like freely adjoining a variable to a ring to create a ring of polynomials, this new object 'satisfies as few equations as possible,' so that declared constants at a function type are automatically injective in the usual mathematical sense. This issue comes up later in Section 6, when we give a condition on *defined* functions which guarantees that they are injective.

3 Proof Irrelevance

3.1 Equality

The two kinds of *judgments* in the theory of LF define typing of terms, and equality between terms. As mentioned, one of the consequences of the equality rules is that for a declared constant $c : \tau_1 \to \cdots \to \tau_n \to \tau$ of n arguments, c is injective, i.e. the terms $c\ M_1 \cdots M_n$ and $c\ M_1' \cdots M_n'$ are equal if only if $M_i = M_i'$ for all $1 \leq i \leq n$. There are times when this is not the desired behavior; sometimes we would like to be able make certain arguments to a declared function *irrelevant* when it comes to deciding equality. In particular, this is a common situation when some arguments to a function are meant to be thought of as *witnesses of provability* rather than objects whose internal structure is important. Generally, suppose we use the following signature:

$$\begin{aligned}
&t, u : \mathsf{type}\\
&p : \Pi x{:}\, t.\mathsf{type}\\
&c : \Pi x{:}\, t.\Pi y{:}\, (p\ x).u\\
&a : t\\
&b, b' : p\ a.
\end{aligned}$$

Intuitively, we have an object a of type t, and some predicate p on objects of type t. The function c takes two arguments: an argument x of type t, and an argument y of type $p\ x$ (which might be thought of as a proof that p holds of x) and returns an object of type u. As the preceding discussion describes, $c\ a\ b \neq c\ a\ b'$ because not all of c's arguments are the same on both sides of the equation — b and b' differ.

We would like to declare instead a $d : \Pi x{:}\, t.\Pi y \div (p\ x).u$, where the $\div$ symbol is supposed to be a promise that the argument y 'doesn't matter' in the output of d. The next section outlines the type theory in which this declaration makes sense, and has the desired behavior.

3.2 Proof Irrelevant Typing

The main typing judgment in LF has the form $\Gamma \vdash M : A$, where Γ is a list of typed variables $x_1 : A_1, \ldots, x_n : A_n$, M is an object, and A is a type. It is read as "in the context Γ, M has the type A." The modifications to typing to accommodate reasoning about irrelevance are on the one hand extending contexts to allow also *irrelevant variables* $x \div A$ in contexts and on the other defining an additional *irrelevant typing judgment* $\Gamma \vdash M \div A$. A variable $x \div A$ can be thought of as the assumption that an object of type A exists, but whose structure cannot be inspected. We define an operation $—^{\oplus}$ on contexts by

$$\begin{array}{rcl} \cdot^{\oplus} & := & \cdot \\ (\Gamma, x : A)^{\oplus} & := & \Gamma^{\oplus}, x : A \\ (\Gamma, x \div A)^{\oplus} & := & \Gamma^{\oplus}, x : A \end{array}$$

which converts all such variables to normal variable declarations. The intuition behind this definition is that $\Gamma^{\oplus}$ is the world in which we can inspect all proofs, derived from the world Γ where some proofs are hidden; with this intuition, we can reason that if we want to establish in Γ that A is provable without making guarantees about the structure of our proof, then we should be allowed to 'live in $\Gamma^{\oplus}$' and inspect irrelevant assumptions freely. We therefore have[2] the inference rule

$$\frac{\Gamma^{\oplus} \vdash M : A}{\Gamma \vdash M \div A}$$

The judgment $\Gamma \vdash M \div A$ says that, in the context Γ, M is a proof that there exists a term of type A.

3.3 Irrelevant Equality

The corresponding equality judgment in LF is also typed; in other words, we do not ask if $\Gamma \vdash M = N$, but if $\Gamma \vdash M = N : A$ — if M and N are equal objects *at type* A. We add a new *irrelevant equality judgment* $\Gamma \vdash M = N \div A$, which is defined[3] by the rule

$$\frac{\Gamma \vdash M \div A \qquad \Gamma \vdash N \div A}{\Gamma \vdash M = N \div A}$$

We see immediately that, as desired, this means that equality at an irrelevant type is trivial: any two objects at the same irrelevant type are equated. The consequences of this definition become more clear when we consider the natural definition of the application equality rule.

[2]In fact with one additional premise $\Gamma \vdash A : \mathsf{type}$ for technical reasons as explained in Section 4.5.

[3]This rule captures the correct intuition, but the actual premises are of the form $\Gamma^{\oplus} \vdash M = M : A$ rather than $\Gamma \vdash M \div A$ to make certain lemmas easier to prove, and again also include $\Gamma \vdash A : \mathsf{type}$ for technical reasons.

There is a rule in LF which can be used to show that two terms which are applications of a function to an argument are equal:

$$\frac{\Gamma \vdash M_1 = M_2 : \Pi x : A.B \qquad \Gamma \vdash N_1 = N_2 : A}{\Gamma \vdash M_1\ N_1 = M_2\ N_2 : B[N_1/x]}$$

For functions which take an *irrelevant argument* we add the rule

$$\frac{\Gamma \vdash M_1 = M_2 : \Pi x \div A.B \qquad \Gamma \vdash N_1 = N_2 \div A}{\Gamma \vdash M_1 \circ N_1 = M_2 \circ N_2 : B[N_1/x]}$$

where $\circ$ is the *irrelevant application* operator. We see in particular that if $M : \Pi x \div A.B$, then $M \circ N_1$ and $M \circ N_2$ for any $N_1, N_2 \div A$. This exactly captures the fact that M is allowed to require an argument of type A, but cannot observe its structure, for the result of applying M to an object is the same regardless of what object it is given.

3.4 Applications

3.4.1 Subset Types

As hinted at earlier, irrelevance can be used to adequately encode *subset types*. For example, suppose we wish to encode the type of composite numbers as a subset of the natural numbers. A standard representation of the naturals with addition and multiplication is

$$\begin{aligned}
&\mathsf{nat} : \mathsf{type} \\
&\mathsf{z} : \mathsf{nat} \\
&\mathsf{s} : \mathsf{nat} \to \mathsf{nat} \\
&\mathsf{sum} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \to \mathsf{type} \\
&\mathsf{prod} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \to \mathsf{type}
\end{aligned}$$

followed by declarations which define *logic programs* for addition and multiplication such that $\mathsf{sum}\,A\ B\ C$ is the type of proofs that $A + B = C$ and $\mathsf{prod}\,A\ B\ C$ is the type of proofs that $A \times B = C$. Since a number x is composite if and only if there exist natural numbers $a, b \geq 2$ such that $ab = x$ (or equivalently, if there exist arbitrary natural numbers a, b such that $(a+2)(b+2) = x$) we might then try to define a type

$$\mathsf{comp} : \mathsf{type}$$

of composites and provide a constructor

$$\begin{aligned}
\mathsf{comp_rule} \quad : \quad & \Pi x : \mathsf{nat}\,.\Pi a \div \mathsf{nat}\,.\Pi b \div \mathsf{nat}\,. \\
& \Pi p \div \mathsf{prod}\,(\mathsf{s}\,\mathsf{s}\,a)\ (\mathsf{s}\,\mathsf{s}\,b)\ x.\,\mathsf{comp}
\end{aligned}$$

which takes a natural x, natural numbers a, b, and a proof p that $(a + 2)(b + 2) = x$, and produces an object of type comp. By marking a, b, and p irrelevant we intend that there is exactly one object of type comp for each composite number; we would hope that $\mathsf{rcomp\_rule}\ 12\ 1\ 2\ p_1$ and $\mathsf{rcomp\_rule}\ 12\ 0\ 4\ p_2$ (where p_1, p_2 are proofs that $3 \times 4 = 12$ and $2 \times 6 = 12$, respectively) are simply different ways of exhibiting the compositeness of the number 12, and are therefore equated. Unfortunately, it is not the case that $p_1 = p_2 \div A$ for any type A, since p_1 and p_2 have different types, since they are not merely different proofs, but proofs of different facts. In fact the signature given is not well-typed: The expression $\mathsf{prod}(\mathsf{s\,s}\,a)\ (\mathsf{s\,s}\,b)\ x$ is a type under the assumptions $a, b : \mathsf{nat}$, but what we really have available is merely $a, b \div \mathsf{nat}$. Both of these issues are related to the 'technical premise' $\Gamma \vdash A : \mathsf{type}$ of the irrelevant typing rule.

Thankfully the problem can be worked around, but it serves to illuminate the subtleties inherent in mixing the apparently simple idea of ignoring the structure of proofs with rich type systems. The solution is to declare a separate predicate of compositeness

$$\begin{aligned} \mathsf{is\_comp} &: \mathsf{nat} \to \mathsf{type} \\ \mathsf{is\_comp\_rule} &: \Pi x : \mathsf{nat}\,.\Pi a : \mathsf{nat}\,.\Pi b : \mathsf{nat}\,. \\ &\quad\ \Pi p : \mathsf{prod}\,(\mathsf{s\,s}\,a)\ (\mathsf{s\,s}\,b)\ x.(\mathsf{is\_comp}\ x) \end{aligned}$$

and mark as irrelevant a proof of this predicate

$$\mathsf{comp\_rule} : \Pi x : \mathsf{nat}\,.\Pi p \div \mathsf{is\_comp}\ x.\,\mathsf{comp}$$

Subset types can also appear more realistically in encodings of programming languages. Suppose we want to represent a lambda calculus which has a binder $\lambda^{\geq 1}$ which requires the variable it binds to occur at least once. We could try doing this by beginning with the usual untyped lambda calculus

$$\begin{aligned} &\mathsf{tm} : \mathsf{type} \\ &\mathsf{app} : \mathsf{tm} \to (\mathsf{tm} \to \mathsf{tm}) \\ &\mathsf{lam} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm} \end{aligned}$$

defining a predicate

$$\mathsf{occurs} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{type}$$

via the *logic program*

$$\begin{aligned} &\mathsf{occurs\_var} : \mathsf{occurs}(\lambda x.x) \\ &\mathsf{occurs\_appl} : \mathsf{occurs}(\lambda x.\,\mathsf{app}\ \ M_1\ M_2) \leftarrow \mathsf{occurs}(\lambda x.M_1) \\ &\mathsf{occurs\_appr} : \mathsf{occurs}(\lambda x.\,\mathsf{app}\ \ M_1\ M_2) \leftarrow \mathsf{occurs}(\lambda x.M_2) \\ &\mathsf{occurs\_lam} : \mathsf{occurs}(\lambda x.\,\mathsf{lam}(M\ x)) \leftarrow (\Pi y{:}\,\mathsf{tm}\,.\,\mathsf{occurs}(\lambda x.M\ x\ y)) \end{aligned}$$

which captures the proposition that an open term uses its free variable, and declaring a constant

$$\mathsf{olam} : \Pi t : (\mathsf{tm} \to \mathsf{tm}).\,\mathsf{occurs}\,t \to \mathsf{tm}$$

This encoding, however, is not *adequate* in the sense that there are generally multiple LF terms which represent an object-language term. This is because there are potentially many proofs that a variable occurs bound — in fact there is one proof per occurrence of the variable. Therefore we want to equate all terms using olam that differ only in which occurrence proof they use. That is, we should declare instead

$$\mathsf{olam} : \Pi t \div (\mathsf{tm} \to \mathsf{tm}).\, \mathsf{occurs}\, t \to \mathsf{tm}$$

and our encoding is adequate.

3.4.2 Subterm Omission in Proof-Carrying Code

In a language like Java, some measure of safety of running code is insured by running the code 'in a sandbox,' inside a trusted virtual machine. Proof-carrying code is another technique which aims to achieve the same (if not greater) safety properties without sacrificing runtime efficiency to emulating a virtual machine. The burden of a making a program safe falls instead on the author of the program, the *code producer*. The recipient of a program, the *code consumer* requires a *proof* that the program received satisfies some safety policy, and so the code producer must send with the program a formal certificate of safety which can be mechanically checked by the code consumer to actually prove that the program won't violate the policy.

Unfortunately, these certificates can sometimes be large, even on the order of the size of the size of the program being proved safe. Techniques for reducing proof size are desirable. Although the problem of finding a proof for a given proposition is typically undecidable, a proof may have many subproofs which could be easily and efficiently reconstructed by the code consumer. For instance, as part of a large proof that shows that a program always computes a certain mathematical function correctly, it might be necessary to show some trivial fact, say $3 + 4 = 7$. Now the formal correctness of this program depends on every last detail of the proof being correct, but there is no need to send a proof of $3 + 4 = 7$ across the network — there can simply be a blank spot in the proof with an instruction saying, "please check that in fact 3+4=7." The trade-off here is saving network bandwidth by perhaps spending more time reconstructing proofs on the code consumer end.

In practice, proofs are frequently represented as terms in a type theory like LF, and checked with a tool like Twelf. In this case, the idea of omitting subproofs really means omitting sub*terms*. The question to be addressed is, when can a subterm be safely omitted? Twelf already has facilities for providing guarantees of *termination* of some predicates considered as logic programs. When we can show using these tools that searching for a proof of P (which is what is meant by "running the logic program P") always terminates, and we know a proof of P, then we can be sure that the code consumer can also find a proof of P. What we do not know is that the code consumer will find the *same* proof. It may seem like a desirable property of a type system that if we replace a subterm S of a term M with a different subterm S' of the same type, then M is still well-typed, but dependent types systems do not necessarily have this property exactly because of the dependence of types on terms. For example, in the signature

$$
\begin{aligned}
&a, z : \mathsf{type} \\
&b : a \rightarrow \mathsf{type} \\
&c : \Pi x\colon a.(b\ x) \\
&d : \Pi x\colon a.(b\ x) \rightarrow z \\
&k_1, k_2 : a
\end{aligned}
$$

we have the typing $\cdot \vdash (d\ k_1)(c\ k_1) : z$ but not $\cdot \vdash (d\ k_1)(c\ k_2) : z$, even though we have only changed one subterm of type a to another.

If we introduce proof irrelevance, however, then it can be shown that replacing one subterm under an irrelevant application with another preserves the whole term being well-typed. Therefore, it is safe to omit a subproof of a large proof if we can show that the subproof can be decidably recovered (i.e. if the predicate can be shown to be terminating) and occurs under an irrelevant application.

4 Type Theory

We begin by defining a type theory similar to a fragment of the theory in [Pfe01], which is in turn based on LF. There are several important differences between this system and that of [Pfe01]. For simplicity, we omit intensional typing entirely. The rules for deriving $\Gamma \vdash M \div A$ and $\Gamma \vdash M = M' \div A$ carry an extra type validity premise. Finally, we avoid defining the judgment $\Gamma \vdash A \div \mathsf{type}$, and consequently the first premise of each of the the Π-introduction rules has $: \mathsf{type}$ in place of $\star\, \mathsf{type}$.

4.1 Syntax

The syntax, as usual, is divided into objects, families, and kinds.

Kinds	K	$::= \mathsf{type} \mid \Pi x{:}A.K \mid \Pi x{\div}A.K$
Families	A	$::= A\ M \mid A \circ M \mid \Pi x{:}A.B \mid \Pi x{\div}A.B$
Objects	M	$::= x \mid c \mid \lambda x{:}A.M \mid \lambda x{\div}A.M \mid M_1\ M_2 \mid M_1 \circ M_2$
Signatures	Σ	$::= \cdot \mid \Sigma, a : K \mid \Sigma, c : A$
Contexts	Γ	$::= \cdot \mid \Gamma, x : A \mid \Gamma, x \div A$

The symbols appearing here foreign to LF are $\circ$ and $\div$, whose intended meanings are irrelevant application and irrelevant typing, respectively.

4.2 Judgments

The main judgments of the theory are as follows:

Valid Signatures	$\vdash \Sigma\ \mathsf{sig}$	Σ is a valid signature.
Valid Contexts	$\vdash_\Sigma \Gamma\ \mathsf{ctx}$	Γ is a valid context.
Object Typing	$\Gamma \vdash_\Sigma M : A$	In Γ, M has type A.
Proof Object Typing	$\Gamma \vdash_\Sigma M \div A$	In Γ, M is a proof of A.
Type Validity	$\Gamma \vdash_\Sigma A : K$	In Γ, A has kind K.
Kind Validity	$\Gamma \vdash_\Sigma K : \mathsf{kind}$	In Γ, K is a valid kind.
Object Equality	$\Gamma \vdash_\Sigma M_1 = M_2 : A$	In Γ, $M_1 = M_2$ at type A.
Proof Equality	$\Gamma \vdash_\Sigma M_1 = M_2 \div A$	In Γ, $M_1 = M_2$ as proofs of A.
Family Equality	$\Gamma \vdash_\Sigma A_1 = A_2 : K$	In Γ, $A_1 = A_2$ at K.
Kind Equality	$\Gamma \vdash_\Sigma K_1 = K_2 : \mathsf{kind}$	In Γ, K_1 and K_2 are equal kinds.

4.3 Typing Rules

4.3.1 Signatures

Valid signatures are defined by

$$\frac{}{\vdash \cdot\ \mathsf{sig}}\ \mathsf{vs\_empty}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad \cdot \vdash_\Sigma A : \mathsf{type}}{\vdash \Sigma, c : A\ \mathsf{sig}}\ \mathsf{vs\_cons}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad \cdot \vdash_\Sigma K : \mathsf{kind}}{\vdash \Sigma, a : K\ \mathsf{sig}}\ \mathsf{vs\_fam}$$

4.3.2 Contexts

Valid contexts are defined by

$$\frac{}{\vdash_\Sigma \cdot\ \mathsf{ctx}}\ \mathsf{vc\_empty}$$

$$\frac{\vdash_\Sigma \Gamma : \mathsf{ctx} \qquad \Gamma \vdash_\Sigma A : \mathsf{type}}{\vdash_\Sigma \Gamma, x \star A\ \mathsf{ctx}}\ \mathsf{vc\_var}$$

In what follows, unless specified otherwise, $\Gamma \vdash_\Sigma J$ for some judgment J assumes that $\vdash \Sigma\ \mathsf{sig}$ and $\vdash_\Sigma \Gamma\ \mathsf{ctx}$, and we typically omit the subscript Σ for brevity.

4.3.3 Proofs

We define recursively an operation $—^\oplus$ ('context promotion') on contexts:

$$\begin{array}{rcl} \cdot^\oplus & := & \cdot \\ (\Gamma, x : A)^\oplus & := & \Gamma^\oplus, x : A \\ (\Gamma, x \div A)^\oplus & := & \Gamma^\oplus, x : A \end{array}$$

The proof typing and equality rules are then

$$\frac{\Gamma^{\oplus} \vdash M : A \qquad \Gamma \vdash A : \mathsf{type}}{\Gamma \vdash M \div A}\,\mathsf{pot}$$

$$\frac{\Gamma^{\oplus} \vdash M = M : A \qquad \Gamma^{\oplus} \vdash N = N : A \qquad \Gamma \vdash A : \mathsf{type}}{\Gamma \vdash M = N \div A}\,\mathsf{pe}$$

An assumption $x \div A$ is weaker than $x : A$ — it asserts that there exists a proof of A, but does not reveal the proof's structure. The context promotion operator allows such weakened assumptions to be recovered and used to derive a judgment of the form $M \div A$, "M witnesses the existence of a proof of A". When comparing two proofs for equality, the result is trivially true as long as the proofs have the same type, and that type is again valid in Γ. Both rules differ from their analogues in [Pfe01] by additionally requiring that A is a valid type in the *unpromoted* context Γ.

4.3.4 Object Typing

The typing rules for objects, families, and kinds are straightforward generalizations of those in LF as presented in [HP01]. We write $\star$ to mean either $:$ or $\div$, and $*$ to mean either juxtaposition (i.e. normal application) or $\circ$. When more than one $\star$ and/or $*$ appear in the same rule, they are to be either all relevant, or all irrelevant.

$$\frac{c : A \in \Sigma}{\Gamma \vdash c : A}\,\mathsf{ot\_const}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{ot\_var}$$

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x \star A \vdash M : B}{\Gamma \vdash \lambda x{\star}A.M : \Pi x{\star}A.B}\,\mathsf{ot\_lam}$$

$$\frac{\Gamma \vdash M_1 : \Pi x{\star}A.B \qquad \Gamma \vdash M_2 \star A}{\Gamma \vdash M_1 * M_2 : [M_2/x]B}\,\mathsf{ot\_app}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B : \mathsf{type}}{\Gamma \vdash M : B}\,\mathsf{ot\_conv}$$

4.3.5 Family Validity

$$\frac{a : K \in \Sigma}{\Gamma \vdash a : K}\,\mathsf{fv\_const}$$

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x \star A \vdash B : \mathsf{type}}{\Gamma \vdash \Pi x{\star}A.B : \mathsf{type}}\,\mathsf{fv\_pi}$$

$$\frac{\Gamma \vdash A_1 : \Pi x{\star}A_2.K \qquad \Gamma \vdash M \star A_2}{\Gamma \vdash A_1 * M : [M/x]K}\,\mathsf{fv\_app}$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K = L : \mathsf{kind}}{\Gamma \vdash A : L}\ \mathsf{fv\_conv}$$

4.3.6 Kind Validity

$$\frac{}{\Gamma \vdash \mathsf{type} : \mathsf{kind}}\ \mathsf{kv\_type}$$

$$\frac{\Gamma \vdash A : \mathsf{type} \qquad \Gamma, x \star A \vdash K : \mathsf{kind}}{\Gamma \vdash \Pi x{\star}A.K : \mathsf{kind}}\ \mathsf{kv\_pi}$$

4.4 Definitional Equality

We define equality of terms based on parallel conversion and extensionality. Premises which will later be shown to be redundant are enclosed in {braces}.

4.4.1 Object Equality

Simultaneous Congruence

$$\frac{c : A \in \Sigma}{\Gamma \vdash c = c : A}\ \mathsf{oe\_const}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x = x : A}\ \mathsf{oe\_var}$$

$$\frac{\Gamma \vdash A = A' : \mathsf{type} \qquad \Gamma \vdash A = A'' : \mathsf{type} \qquad \Gamma, x \star A \vdash M = M' : B}{\Gamma \vdash \lambda x{\star}A'.M = \lambda x{\star}A''.M' : \Pi x{\star}A.B}\ \mathsf{oe\_lam}$$

$$\frac{\Gamma \vdash M_1 = M_1' : \Pi x{\star}A.B \qquad \Gamma \vdash M_2 = M_2' \star A}{\Gamma \vdash M_1 * M_2 = M_1' * M_2' : [M_2/x]B}\ \mathsf{oe\_app}$$

Extensionality

$$\frac{\Gamma \vdash A : \mathsf{type} \quad \Gamma \vdash M, N : \Pi x{\star}A.B \quad \Gamma, x \star A \vdash M * x = N * x : B}{\Gamma \vdash M = N : \Pi x{\star}A.B}\ \mathsf{oe\_ext}$$

Parallel Reduction

$$\frac{\{\Gamma \vdash A : \mathsf{type}\} \qquad \Gamma, x \star A \vdash M_2 = M_2' : B \qquad \Gamma \vdash M_1 = M_1' \star A}{\Gamma \vdash (\lambda x{\star}A.M_2) * M_1 = [M_1'/x]M_2' : [M_1/x]B}\ \mathsf{oe\_red}$$

Type Conversion

$$\frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash A = B : \mathsf{type}}{\Gamma \vdash M = N : B}\ \mathsf{oe\_conv}$$

Equivalence

$$\frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash N = O : A}{\Gamma \vdash M = O : A}\,\textsf{oe\_trans}$$

$$\frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A}\,\textsf{oe\_sym}$$

4.4.2 Family Equality

Simultaneous Congruence

$$\frac{a : K \in \Sigma}{\Gamma \vdash a = a : K}\,\textsf{fe\_const}$$

$$\frac{\Gamma \vdash A = A' : \textsf{type} \qquad \{\Gamma \vdash A : \textsf{type}\} \qquad \Gamma, x \star A \vdash B = B' : \textsf{type}}{\Gamma \vdash \Pi x{\star}A.B = \Pi x{\star}A'.B' : \textsf{type}}\,\textsf{fe\_pi}$$

$$\frac{\Gamma \vdash A = A' : \Pi x{\star}A.K \qquad \Gamma \vdash M = M' \star A}{\Gamma \vdash A * M = A' * M' : [M/x]K}\,\textsf{fe\_app}$$

Kind Conversion

$$\frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash K = L : \textsf{kind}}{\Gamma \vdash A = B : L}\,\textsf{fe\_app}$$

Equivalence

$$\frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash B = C : K}{\Gamma \vdash A = C : K}\,\textsf{fe\_trans}$$

$$\frac{\Gamma \vdash A = B : K}{\Gamma \vdash B = A : K}\,\textsf{fe\_sym}$$

4.4.3 Kind Equality

Simultaneous Congruence

$$\frac{}{\Gamma \vdash \textsf{type} = \textsf{type} : \textsf{kind}}\,\textsf{ke\_type}$$

$$\frac{\Gamma \vdash A = A' : \textsf{type} \qquad \{\Gamma \vdash A : \textsf{type}\} \qquad \Gamma, x \star A \vdash K = K' : \textsf{kind}}{\Gamma \vdash \Pi x{\star}A.K = \Pi x{\star}A'.K' : \textsf{kind}}\,\textsf{ke\_pi}$$

Equivalence

$$\frac{\Gamma \vdash K = L : \textsf{kind} \qquad \Gamma \vdash L = L' : \textsf{kind}}{\Gamma \vdash K = L' : \textsf{kind}}\,\textsf{ke\_trans}$$

$$\frac{\Gamma \vdash K = L : \textsf{kind}}{\Gamma \vdash L = K : \textsf{kind}}\,\textsf{ke\_sym}$$

4.5 Elementary Properties

We can immediately establish several very basic lemmas concerning the judgments defined.

Lemma 4.1 (Weakening) *If* $\Gamma, \Gamma' \vdash J$*, then* $\Gamma, x \star A, \Gamma' \vdash J$.

Proof Straightforward induction. ■

Lemma 4.2 (Reflexivity)

1. *If* $\Gamma \vdash M \star A$*, then* $\Gamma \vdash M = M \star A$.
2. *If* $\Gamma \vdash A : K$*, then* $\Gamma \vdash A = A : K$.
3. *If* $\Gamma \vdash K : \mathsf{kind}$*, then* $\Gamma \vdash K = K : \mathsf{kind}$.

Proof Straightforward induction. ■

Lemma 4.3 (Substitution) *Suppose* $\vdash \Gamma, x \star A, \Gamma' : \mathsf{ctx}$*. If* $\Gamma \vdash M \star A$ *and* $\Gamma, x \star A, \Gamma' \vdash J$*, then* $\Gamma, [M/x]\Gamma' \vdash [M/x]J$.

Proof Straightforward induction. ■

Our eventual goal is to prove validity, Lemma 4.12. In its proof, the inherent asymmetry of rule $\mathsf{oe\_app}$ (we must arbitrarily choose one of M_2 or M_2' to substitute for x in B in the conclusion) requires functionality (Lemma 4.10) to be shown first. In fact the functionality lemma does not hold for the system in [Pfe01]. A counterexample is as follows: let Σ be the signature

$$\begin{aligned}
&o : \mathsf{type}\\
&a : o \to \mathsf{type}\\
&b : \Pi x{:}o.a\ x\\
&c : \Pi x{\div}o.\Pi y{\div}(a\ x).o\\
&k_1, k_2 : o
\end{aligned}$$

Observe that the present system would not admit this declaration of c; for the expression $\Pi x{\div}o.\Pi y{\div}(a\ x).o$ to be a valid type we would need to have not merely $x \div o \vdash_\Sigma a\ x \div \mathsf{type}$ (which requires only $(x \div o)^\oplus \vdash_\Sigma a\ x : \mathsf{type}$, which does hold) but $x \div o \vdash_\Sigma a\ x : \mathsf{type}$, which does not hold. In [Pfe01] we could form a derivation

$$\dfrac{\dfrac{\dfrac{c \in \Sigma}{z \div o \vdash c : \Pi x{\div}o.\Pi y{\div}(a\ x).o} \quad \dfrac{\dfrac{}{z : o \vdash z : o}}{z \div o \vdash z \div o}}{z \div o \vdash c \circ z : \Pi y{\div}(a\ z).o} \qquad \dfrac{\dfrac{\dfrac{b \in \Sigma}{z : o \vdash b : \Pi x{:}o.a\ x} \quad \dfrac{}{z : o \vdash z : o}}{z : o \vdash b\ z : a\ z}}{z \div o \vdash b\ z \div a\ z}}{z \div o \vdash c \circ z \circ (b\ z) : o}$$

The step from $z : o \vdash b\ z : a\ z$ to $z \div o \vdash b\ z \div a\ z$ is also prohibited by the present system. The functionality lemma would allow us to conclude from $\vdash k_1 = k_2 \div o$ that

$$\vdash c \circ k_1 \circ (b\ k_1) = c \circ k_2 \circ (b\ k_2) : o$$

By inversion, the only way this derivation could exist is if it were the case that $\vdash b\ k_1 = b\ k_2 \div A$ for some type A. However, the terms $b\ k_1$ and $b\ k_2$ have distinct types $a\ k_1$ and $a\ k_2$, so this cannot occur — they are proof terms for different types, so they cannot even be equal as (irrelevant) proofs.

To show our system satisfies functionality, we use a somewhat delicate induction argument which depends on two auxilliary judgments:

Context Equality	$\vdash \Gamma = \Gamma' : \mathsf{ctx}$	Γ and Γ' are equal contexts.
Context Ordering	$\vdash \Gamma \preceq \Gamma' : \mathsf{ctx}$	Γ is weaker than Γ'.

They are defined as follows:

$$\frac{}{\vdash \cdot = \cdot : \mathsf{ctx}}\ \mathsf{ce\_nil}$$

$$\frac{\vdash \Gamma = \Gamma' : \mathsf{ctx} \qquad \Gamma' \vdash A = A' : \mathsf{type}}{\vdash \Gamma, x \star A = \Gamma', x \star A' : \mathsf{ctx}}\ \mathsf{ce\_var}$$

$$\frac{}{\cdot \preceq \cdot}\ \mathsf{co\_nil}$$

$$\frac{\Gamma \preceq \Gamma'}{\Gamma, x \star A \preceq \Gamma', x \star A}\ \mathsf{co\_eq}$$

$$\frac{\Gamma \preceq \Gamma'}{\Gamma, x \div A \preceq \Gamma', x : A}\ \mathsf{co\_lt}$$

The meaning of context equality is simply a corresponding equality of the typings of all variables. The ordering $\Gamma \preceq \Gamma'$ says that Γ contains the same assumptions as Γ', except perhaps some of them have been weakened from normal to irrelevant assumptions. We proceed to establish some basic facts about these judgments. Some of the proofs are not quite as trivial as might be expected, because of the asymmetry of $\mathsf{ce\_var}$. As with function application, there is an arbitrary choice in the statement of the rule, in this case the choice of under which context (Γ or Γ') we need to show $A = A' : \mathsf{type}$.

Lemma 4.4 (Context Equality Reflexivity) *If* $\vdash \Gamma : \mathsf{ctx}$, *then* $\vdash \Gamma = \Gamma : \mathsf{ctx}$.

Proof Straightforward induction using Lemma 4.2. ■

Lemma 4.5 (Judgment Lifting) *If* $\Gamma \vdash J$ *and* $\Gamma \preceq \Gamma'$, *then* $\Gamma' \vdash J$.

Proof Straightforward induction on $\mathcal{D} :: \Gamma \vdash J$. Some representative cases:

Case:

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \mathsf{type}} \quad \overset{\mathcal{D}_2}{\Gamma, x \star A \vdash M : B}}{\Gamma \vdash \lambda x{\star}A.M : \Pi x{\star}A.B}\ \mathsf{ot\_lam}$$

We apply the induction hypothesis to $\mathcal{D}_1$ to get that $\Gamma' \vdash A : \mathsf{type}$, and to $\mathcal{D}_2$ and the fact that $\Gamma, x \star A \preceq \Gamma', x \star A$ to obtain $\Gamma', x \star A \vdash M : B$, from which we conclude using $\mathsf{ot\_lam}$ that $\Gamma' \vdash \lambda x{\star}A.M : \Pi x{\star}A.B$.

Case:

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_1}{\Gamma^{\oplus} \vdash M : A} \quad \overset{\mathcal{D}_2}{\Gamma \vdash A : \mathsf{type}}}{\Gamma \vdash M \div A}\ \mathsf{pot}$$

It is easily seen that $\Gamma \preceq \Gamma'$ implies that $\Gamma'^{\oplus}$ is identical to $\Gamma^{\oplus}$, so $\mathcal{D}_1$ is also a derivation of $\Gamma'^{\oplus} \vdash M : A$. We know by the induction hypothesis applied to $\mathcal{D}_2$ that $\Gamma' \vdash A : \mathsf{type}$, so by pot we have $\Gamma' \vdash M \div A$.

■

Corollary 4.6 *If* $\Gamma \vdash J$*, then* $\Gamma^{\oplus} \vdash J$.

Proof By inspection, $\Gamma \preceq \Gamma^{\oplus}$.

Lemma 4.7 (Context Equality Promotion) *If* $\vdash \Gamma_1 = \Gamma_2 : \mathsf{ctx}$*, then we have* $\vdash \Gamma_1^{\oplus} = \Gamma_2^{\oplus} : \mathsf{ctx}$.

Proof By induction on $\mathcal{D} :: (\vdash \Gamma_1 = \Gamma_2 : \mathsf{ctx})$.

Case:

$$\cfrac{}{\vdash \cdot = \cdot : \mathsf{ctx}}\ \mathsf{ce\_nil}$$

Immediately we have $\vdash \cdot^{\oplus} = \cdot^{\oplus} : \mathsf{ctx}$.

Case:

$$\cfrac{\overset{\mathcal{D}_1}{\vdash \Gamma_1 = \Gamma_2 : \mathsf{ctx}} \quad \overset{\mathcal{D}_2}{\Gamma_2 \vdash A = A' : \mathsf{type}}}{\vdash \Gamma_1, x \star A = \Gamma_2, x \star A' : \mathsf{ctx}}\ \mathsf{ce\_var}$$

By the induction hypothesis on $\mathcal{D}_1$ we know $\vdash \Gamma_1^{\oplus} = \Gamma_2^{\oplus} : \mathsf{ctx}$. Applying Corollary 4.6 to $\mathcal{D}_2$ provides $\Gamma_2^{\oplus} \vdash A = A' : \mathsf{type}$. So we can apply $\mathsf{ce\_var}$ to obtain $\vdash \Gamma_1^{\oplus}, x : A = \Gamma_2^{\oplus}, x : A$.

■

Lemma 4.8 (Context Conversion) *Suppose* $\vdash \Gamma = \Gamma' : \mathsf{ctx}$*. If* $\Gamma \vdash J$*, then* $\Gamma' \vdash J$.

Proof By induction on $\mathcal{D} :: \Gamma \vdash J$. Some important cases:

Case:

$$\dfrac{\overset{\mathcal{D}_1}{\Gamma^{\oplus} \vdash M : A} \quad \overset{\mathcal{D}_2}{\Gamma \vdash A : \mathsf{type}}}{\Gamma \vdash M \div A}\,\mathsf{pot}$$

By Lemma 4.7, $\vdash \Gamma^{\oplus} = \Gamma'^{\oplus} : \mathsf{ctx}$ so the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2$ gives $\Gamma'^{\oplus} \vdash M : A$ and $\Gamma' \vdash A : \mathsf{type}$, hence $\Gamma' \vdash M \div A$.

Case:

$$\mathcal{D} = \dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{ot\_var}$$

Since $\vdash \Gamma = \Gamma' : \mathsf{ctx}$, we know by inversion on the context validity rules that Γ' is of the form $\Gamma'_1, x : A', \Gamma'_2$ such that $\Gamma'_1 \vdash A = A' : \mathsf{type}$. By weakening, $\Gamma' \vdash A = A' : \mathsf{type}$. Therefore

$$\dfrac{\dfrac{x : A' \in \Gamma'}{\Gamma' \vdash x : A'}\,\mathsf{ot\_var} \quad \dfrac{\Gamma' \vdash A = A' : \mathsf{type}}{\Gamma' \vdash A' = A : \mathsf{type}}\,\mathsf{fe\_sym}}{\Gamma' \vdash x : A}\,\mathsf{ot\_conv}$$

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \mathsf{type}} \quad \overset{\mathcal{D}_2}{\Gamma, x \star A \vdash M : B}}{\Gamma \vdash \lambda x{\star}A.M : \Pi x{\star}A.B}\,\mathsf{ot\_lam}$$

Applying the induction hypothesis to $\mathcal{D}_1$ gives $\Gamma' \vdash A : \mathsf{type}$. Reflexivity implies $\Gamma' \vdash A = A : \mathsf{type}$, hence $\vdash \Gamma, x \star A = \Gamma', x \star A$ using $\mathsf{ce\_var}$. Therefore we can apply the induction hypothesis to $\Gamma, x \star A \vdash M : B$ to obtain $\Gamma', x \star A \vdash M : B$, and conclude using $\mathsf{ot\_lam}$ that $\Gamma' \vdash \lambda x{\star}A.M : \Pi x{\star}A.B$.

Case: $\mathcal{D} =$

$$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A = A' : \mathsf{type}} \quad \overset{\mathcal{D}_2}{\Gamma \vdash A = A'' : \mathsf{type}} \quad \overset{\mathcal{D}_3}{\Gamma, x \star A \vdash M = M' : B}}{\Gamma \vdash \lambda x{\star}A'.M = \lambda x{\star}A''.M' : \Pi x{\star}A.B}\,\mathsf{oe\_lam}$$

Applying the induction hypothesis to $\mathcal{D}_1$ provides a derivation $\mathcal{D}'_1 :: \Gamma' \vdash A = A' : \mathsf{type}$, which we can use to infer

$$\dfrac{\vdash \Gamma = \Gamma' : \mathsf{ctx} \quad \dfrac{\overset{\mathcal{D}'_1}{\Gamma' \vdash A = A' : \mathsf{type}} \quad \dfrac{\overset{\mathcal{D}'_1}{\Gamma' \vdash A = A' : \mathsf{type}}}{\Gamma' \vdash A' = A : \mathsf{type}}\,\mathsf{fe\_sym}}{\Gamma' \vdash A = A : \mathsf{type}}\,\mathsf{fe\_trans}}{\vdash \Gamma, x \star A = \Gamma', x \star A : \mathsf{ctx}}\,\mathsf{ce\_var}$$

so that we can apply the induction hypothesis to $\mathcal{D}_3$ to obtain $\Gamma', x \star A \vdash M = M' : B$. Finally, we apply the induction hypothesis to $\mathcal{D}_2$ to obtain $\Gamma' \vdash A = A'' : \mathsf{type}$ and conclude that

$$\Gamma' \vdash \lambda x \star A'.M = \lambda x \star A''.M : \Pi x{\star}A.B$$

using $\mathsf{oe\_lam}$.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma^{\oplus} \vdash M = M : A} \quad \overset{\mathcal{D}_2}{\Gamma^{\oplus} \vdash N = N : A} \quad \overset{\mathcal{D}_3}{\Gamma \vdash A : \mathsf{type}}}{\Gamma \vdash M = N \div A}\,\mathsf{pe}$$

By Lemma 4.7, we know $\vdash \Gamma^{\oplus} = \Gamma'^{\oplus} : \mathsf{ctx}$, so the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2$ provides $\Gamma'^{\oplus} \vdash M = M : A$ and $\Gamma'^{\oplus} \vdash N = N : A$. The induction hypothesis applied to $\mathcal{D}_3$ gives $\Gamma' \vdash A : \mathsf{type}$, so pe implies $\Gamma' \vdash M = N \div A$. ■

Lemma 4.9 (Context Equality Symmetry) *If* $\vdash \Gamma = \Gamma' : \mathsf{ctx}$, *then* $\vdash \Gamma' = \Gamma : \mathsf{ctx}$.

Proof By induction on $\mathcal{D} :: (\vdash \Gamma = \Gamma' : \mathsf{ctx})$.

Case:

$$\mathcal{D} = \dfrac{}{\vdash \cdot = \cdot : \mathsf{ctx}}\,\mathsf{ce\_nil}$$

Immediate.

Case:

$$\dfrac{\overset{\mathcal{D}_1}{\vdash \Gamma = \Gamma' : \mathsf{ctx}} \quad \overset{\mathcal{D}_2}{\Gamma' \vdash A = A' : \mathsf{type}}}{\vdash \Gamma, x \star A = \Gamma', x \star A' : \mathsf{ctx}}\,\mathsf{ce\_var}$$

By the induction hypothesis applied to $\mathcal{D}_1$, we have $\vdash \Gamma' = \Gamma : \mathsf{ctx}$. By Lemma 4.8 this fact together with $\mathcal{D}_2$ implies that $\Gamma \vdash A = A' : \mathsf{type}$, and by $\mathsf{fv\_sym}$ we deduce $\Gamma \vdash A' = A : \mathsf{type}$, so by $\mathsf{ce\_var}$ we have $\vdash \Gamma', x \star A' = \Gamma, x \star A : \mathsf{ctx}$.

■

We are now prepared to show functionality. The statment of the lemma claims that substitution of equal objects for a variable in a valid context (resp. object, family, kind) results in equal contexts (resp. objects, families, kinds). It is important that we have already shown that we can replace a context with an equal one in arbitrary judgments; proving functionality without its part 1 does not appear to work. Another feature of the proof worth mentioning is the lexicographic order involving the *sum* of the number of inference rules used to show the context valid and those used to actually derive the relevant judgment. This allows the induction hypothesis to be used during the case of, for instance, $\mathsf{ot\_lam}$, even though the context gets larger.

Lemma 4.10 (Functionality) *Suppose*

$$\mathcal{D} :: (\vdash \Gamma, y \star D, \Gamma' \; \mathsf{ctx})$$

$$\Gamma \vdash N \star D \qquad \Gamma \vdash N' \star D$$

$$\Gamma \vdash N = N' \star D$$

then

1. $\vdash \Gamma, [N/y]\Gamma' = \Gamma, [N'/y]\Gamma' : \mathsf{ctx}$.
2. *If* $\mathcal{E} :: \Gamma, y \star D, \Gamma' \vdash M : A$ *then* $\Gamma, [N/y]\Gamma' \vdash [N/y]M = [N'/y]M : [N/y]A$.
3. *If* $\mathcal{E} :: \Gamma, y \star D, \Gamma' \vdash A : K$ *then* $\Gamma, [N/y]\Gamma' \vdash [N/y]A = [N'/y]A : [N/y]K$.
4. *If* $\mathcal{E} :: \Gamma, y \star D, \Gamma' \vdash K : \mathsf{kind}$ *then* $\Gamma, [N/y]\Gamma' \vdash [N/y]K = [N'/y]K : \mathsf{kind}$.

Proof By lexicographic induction on

- The number of inference rules in $\mathcal{E}$ (if any) plus the number of inference rules in $\mathcal{D}$.
- The size of the subject expression in $\mathcal{E}$, if any. (i.e. M in part 2, A in part 3, K in part 4)

1. We proceed by cases on the structure of $\mathcal{D}$.

   Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\vdash \Gamma : \mathsf{ctx}} \quad \overset{\mathcal{D}_2}{\Gamma \vdash D : \mathsf{type}}}{\vdash \Gamma, y \star D : \mathsf{ctx}}\;\mathsf{vc\_var}$$

   By reflexivity and $\mathcal{D}_1$ we have $\vdash \Gamma = \Gamma : \mathsf{ctx}$, as required.

   Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\vdash \Gamma, y \star D, \Gamma'' : \mathsf{ctx}} \quad \overset{\mathcal{D}_2}{\Gamma, y \star D, \Gamma'' \vdash A : \mathsf{type}}}{\vdash \Gamma, y \star D, \Gamma'', x \star' A \; \mathsf{ctx}}\;\mathsf{vc\_var}$$

   The induction hypothesis (part 1) applied to $\mathcal{D}_1$ gives

$$\vdash \Gamma, [N/y]\Gamma'' = \Gamma, [N'/y]\Gamma'' : \mathsf{ctx}$$

   The induction hypothesis (part 3) applied to $\mathcal{D}_1, \mathcal{D}_2$ (which together have one less inference rule than $\mathcal{D}$) gives

$$\Gamma, [N/y]\Gamma'' \vdash [N/y]A = [N'/y]A : \mathsf{type}$$

   Lemma 4.8 then implies that

$$\Gamma, [N'/y]\Gamma'' \vdash [N/y]A = [N'/y]A : \mathsf{type}$$

hence

$$\dfrac{\begin{array}{c}\vdash \Gamma, [N/y]\Gamma'' = \Gamma, [N'/y]\Gamma'' : \mathsf{ctx} \\ \Gamma, [N'/y]\Gamma'' \vdash [N/y]A = [N'/y]A : \mathsf{type}\end{array}}{\vdash \Gamma, [N/y]\Gamma'', x \star' [N/y]A = \Gamma, [N'/y]\Gamma'', x \star' [N'/y]A : \mathsf{ctx}}\ \mathsf{ce\_var}$$

2, 3, 4. We proceed by cases on the structure of $\mathcal{E}$. Some representative cases:

Case:

$$\mathcal{E} = \dfrac{x : A \in \Gamma, y \star D, \Gamma'}{\Gamma, y \star D, \Gamma' \vdash x : A}\ \mathsf{ot\_var}$$

If $x \neq y$, then we must show that $\Gamma, [N/y]\Gamma' \vdash x = x : [N/y]A$, which follows by substitution and reflexivity. If $x = y$, then we know that in fact A is identical to D and '$\star$' is in this case ':'. By weakening on our assumption that $\Gamma \vdash N = N' : D$, we obtain $\Gamma, y : D, \Gamma' \vdash N = N' : D$. By substitution, and the fact that y does not appear in N, N', we have $\Gamma, [N/y]\Gamma' \vdash N = N' : [N/y]D$.

Case:

$$\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ \Gamma, y \star D, \Gamma' \vdash M_1 : \Pi x{\div}A.B & \Gamma, y \star D, \Gamma' \vdash M_2 \div A\end{array}}{\Gamma, y \star D, \Gamma' \vdash M_1 \circ M_2 : [M_2/x]B}\ \mathsf{ot\_app}$$

By inversion,

$$\mathcal{E}_2 = \dfrac{\begin{array}{cc}\mathcal{E}_2' & \mathcal{E}_2'' \\ \Gamma^{\oplus}, y : D, \Gamma'^{\oplus} \vdash M_2 : A & \Gamma, y \star D, \Gamma' \vdash A : \mathsf{type}\end{array}}{\Gamma, y \star D, \Gamma' \vdash M_2 \div A}\ \mathsf{pot}$$

for some $\mathcal{E}_2', \mathcal{E}_2''$. By substitution of N, N' into $\mathcal{E}_2$ and reflexivity, we have

$$\Gamma, [N/y]\Gamma' \vdash [N/y]M_2 = [N/y]M_2 \div [N/y]A \qquad (\flat)$$

and

$$\Gamma, [N'/y]\Gamma' \vdash [N'/y]M_2 = [N'/y]M_2 \div [N'/y]A \qquad (*)$$

By the induction hypothesis (part 3) on $\mathcal{E}_2''$, we know

$$\Gamma, [N/y]\Gamma' \vdash [N/y]A = [N'/y]A : \mathsf{type} \qquad (**)$$

Also by the induction hypothesis (part 1) we know that

$$\vdash \Gamma, [N/y]\Gamma' = \Gamma, [N'/y]\Gamma' : \mathsf{ctx}$$

so by Lemma 4.9 we have

$$\vdash \Gamma, [N'/y]\Gamma' = \Gamma, [N/y]\Gamma' : \mathsf{ctx}$$

hence we can use Lemma 4.8 to infer from $(*)$ that

$$\Gamma, [N/y]\Gamma' \vdash [N'/y]M_2 = [N'/y]M_2 \div [N'/y]A$$

and by inversion obtain a derivation of

$$(\Gamma, [N/y]\Gamma')^{\oplus} \vdash [N'/y]M_2 = [N'/y]M_2 : [N'/y]A$$

Corollary 4.6 and $(**)$ together imply that

$$(\Gamma, [N/y]\Gamma')^{\oplus} \vdash [N/y]A = [N'/y]A : \mathsf{type}$$

so we can use ot_sym, ot_conv to conclude

$$(\Gamma, [N/y]\Gamma')^{\oplus} \vdash [N'/y]M_2 = [N'/y]M_2 : [N/y]A \qquad (\sharp_2)$$

Now by inversion on $(\flat)$ we have a derivation

$$(\Gamma, [N/y]\Gamma')^{\oplus} \vdash [N/y]M_2 = [N/y]M_2 : [N/y]A \qquad (\sharp_1)$$

and substitution on $\mathcal{E}_2''$ provides

$$\Gamma, [N/y]\Gamma' \vdash [N/y]A : \mathsf{type} \qquad (\sharp_3)$$

so we use pe on $(\sharp_1)$, $(\sharp_2)$, and $(\sharp_3)$ to obtain

$$\Gamma, [N/y]\Gamma' \vdash [N/y]M_2 = [N'/y]M_2 \div [N/y]A$$

Finally, we can apply the induction hypothesis (part 2) directly to $\mathcal{E}_1$ to see

$$\Gamma, [N/y]\Gamma' \vdash [N/y]M_1 = [N'/y]M_1 : [N/y]\Pi x{\div}A.B$$

and conclude using oe_app that

$$\Gamma, [N/y]\Gamma' \vdash [N/y](M_1 \circ M_2) = [N'/y](M_1 \circ M_2) : [N/y][M_2/x]B$$

Case:

$$\mathcal{E} = \cfrac{\overset{\mathcal{E}_1}{\Gamma, y \star D, \Gamma' \vdash A : \mathsf{type}} \quad \overset{\mathcal{E}_2}{\Gamma, y \star D, \Gamma', x \div A \vdash M : B}}{\Gamma, y \star D, \Gamma' \vdash \lambda x{\div}A.M : \Pi x{\div}A.B}\ \mathsf{ot\_lam}$$

The induction hypothesis (part 3) applied to $\mathcal{E}_1$ gives

$$\Gamma, [N/y]\Gamma' \vdash [N/y]A = [N'/y]A : \mathsf{type}$$

We can also apply the induction hypothesis (part 2) to $\mathcal{E}_2$, but it requires showing that $\mathcal{E}_2$ is strictly smaller than $\mathcal{E}$ in the ordering we have chosen. We can prove $\Gamma, y \star D, \Gamma', x \div A : \mathsf{ctx}$ by

$$\mathcal{D}' = \cfrac{\overset{\mathcal{D}}{\Gamma, y \star D, \Gamma' : \mathsf{ctx}} \quad \overset{\mathcal{E}_1}{\Gamma, y \star D, \Gamma' \vdash A : \mathsf{type}}}{\vdash \Gamma, y \star D, \Gamma', x \div A : \mathsf{ctx}}\ \mathsf{vc\_var}$$

so that the number of inference rules in the two derivations $|\mathcal{D}'| + |\mathcal{E}_2| = (|\mathcal{D}| + |\mathcal{E}_1| + 1) + |\mathcal{E}_2| = |\mathcal{D}| + (|\mathcal{E}_1| + |\mathcal{E}_2| + 1) = |\mathcal{D}| + |\mathcal{E}|$ stays the same, but the subject expression is smaller — it changes from $\lambda x \div A.M$ to M. So we obtain

$$\Gamma, [N/y]\Gamma', x \div [N/y]A \vdash [N/y]M = [N'/y]M : [N/y]B$$

We then can use substitution of N for y in $\mathcal{E}_1$ and reflexivity to derive

$$\Gamma, [N/y]\Gamma' \vdash [N/y]A = [N/y]A : \mathsf{type}$$

and use oe_lam to conclude

$$\Gamma, [N/y]\Gamma' \vdash [N/y](\lambda x : A.M) = [N'/y](\lambda x : A.M) : [N/y]B$$

■

We can now proceed to show validity and the main lemmas which follow from it. Our development closely follows [HP01].

Lemma 4.11 (Inversion of Products)

1. *If* $\Gamma \vdash \Pi x{\star}A_1.A_2 : K$ *then* $\Gamma \vdash A_1 : \mathsf{type}$ *and* $\Gamma, x \star A_1 \vdash A_2 : \mathsf{type}$.
2. *If* $\Gamma \vdash \Pi x{\star}A.K : \mathsf{kind}$ *then* $\Gamma \vdash A : \mathsf{type}$ *and* $\Gamma, x \star A \vdash K : \mathsf{kind}$.

Proof By induction and inversion, respectively. ■

Lemma 4.12 (Validity)

1. *If* $\Gamma \vdash M \star A$, *then* $\Gamma \vdash A : \mathsf{type}$.
2. *If* $\Gamma \vdash M = N \star A$, *then* $\Gamma \vdash M \star A$, $\Gamma \vdash N \star A$, *and* $\Gamma \vdash A : \mathsf{type}$.
3. *If* $\Gamma \vdash A : K$, *then* $\Gamma \vdash K : \mathsf{kind}$.
4. *If* $\Gamma \vdash A = B : K$, *then* $\Gamma \vdash A : K$, $\Gamma \vdash B : K$, *and* $\Gamma \vdash K : \mathsf{kind}$.
5. *If* $\Gamma \vdash K = L : \mathsf{kind}$, *then* $\Gamma \vdash K : \mathsf{kind}$, *and* $\Gamma \vdash L : \mathsf{kind}$.

Proof Straightforward induction. Some representative cases:

Case:

$$\mathcal{D} = \cfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \vdash M_1 = M_1' : \Pi x{\star}A.B & \Gamma \vdash M_2 = M_2' \star A\end{array}}{\Gamma \vdash M_1 * M_2 = M_1' * M_2' : [M_2/x]B}\ \mathsf{oe\_app}$$

The induction hypotheses allow us to conclude

$$\Gamma \vdash M_1, M_1' : \Pi x{\star}A.B$$

$$\Gamma \vdash \Pi x{\star}A.B : \mathsf{type}$$
$$\Gamma \vdash M_2, M_2' \star A$$
$$\Gamma \vdash A : \mathsf{type}$$

So $\Gamma \vdash M_1 * M_2 : [M_2/x]B$ follows immediately from $\mathsf{ot\_app}$. By inversion of products, $\Gamma, x \star A \vdash B : \mathsf{type}$. By substitution, $\Gamma \vdash [M_2/x]B : \mathsf{type}$, and by functionality, $\Gamma \vdash [M_2/x]B = [M_2'/x]B : \mathsf{type}$. Therefore we can form a derivation

$$\dfrac{\dfrac{\begin{array}{c}\Gamma \vdash M_1' : \Pi x{\star}A.B \\ \Gamma \vdash M_2' \star A\end{array}}{\Gamma \vdash M_1' * M_2' : [M_2'/x]B}\,\mathsf{ot\_app} \quad \dfrac{\Gamma \vdash [M_2/x]B = [M_2'/x]B : \mathsf{type}}{\Gamma \vdash [M_2'/x]B = [M_2/x]B : \mathsf{type}}\,\mathsf{fv\_sym}}{\Gamma \vdash M_1' * M_2' : [M_2/x]B}\,\mathsf{ot\_conv}$$

Case:

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ \{\Gamma \vdash A : \mathsf{type}\} & \Gamma, x \star A \vdash M_2 = M_2' : B & \Gamma \vdash M_1 = M_1' \star A\end{array}}{\Gamma \vdash (\lambda x{\star}A.M_2) * M_1 = [M_1'/x]M_2' : [M_1/x]B}\,\mathsf{oe\_red}$$

By the induction hypothesis on $\mathcal{D}_2$ and $\mathcal{D}_3$ we have $\Gamma, x \star A \vdash M_2, M_2' : B$, $\Gamma, x \star A \vdash B : \mathsf{type}$, and $\Gamma \vdash M_1, M_1' \star A$. Immediately we can show

$$\dfrac{\dfrac{\begin{array}{cc}D_1 & \\ \Gamma \vdash A : \mathsf{type} & \Gamma, x \star A \vdash M_2 : B\end{array}}{\Gamma \vdash \lambda x{\star}A.M_2 : \Pi x{\star}A.B}\,\mathsf{ot\_lam} \quad \Gamma \vdash M_1 \star A}{\Gamma \vdash (\lambda x{\star}A.M_2) * M_1 : [M_1/x]B}\,\mathsf{ot\_app}$$

and apply substitution to see that $\Gamma \vdash [M_1/x]B : \mathsf{type}$. Substitution also provides $\Gamma \vdash [M_1'/x]M_2' : [M_1'/x]B$, so we again use functionality, symmetry of type equality, and type conversion to obtain $\Gamma \vdash [M_1'/x]M_2' : [M_1/x]B$.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{ccc}\mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ \Gamma^{\oplus} \vdash M = M : A & \Gamma^{\oplus} \vdash N = N : A & \Gamma \vdash A : \mathsf{type}\end{array}}{\Gamma \vdash M = N \div A}\,\mathsf{pe}$$

The induction hypothesis gives $\mathcal{D}_1' :: \Gamma^{\oplus} \vdash M : A$, $\mathcal{D}_2' :: \Gamma^{\oplus} \vdash N : A$. Therefore

$$\dfrac{\begin{array}{cc}\mathcal{D}_1' & \mathcal{D}_3 \\ \Gamma^{\oplus} \vdash M : A & \Gamma \vdash A : \mathsf{type}\end{array}}{\Gamma \vdash M \div A}\,\mathsf{pot}$$

and

$$\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\oplus} \vdash N : A} \quad \overset{\mathcal{D}_3}{\Gamma \vdash A : \mathsf{type}}}{\Gamma \vdash N \div A}\,\mathsf{pot}$$

and immediately $\Gamma \vdash A : \mathsf{type}$.

■

Lemma 4.13 (Functionality for Equality) *Suppose* $\vdash \Gamma, x \star A : \mathsf{ctx}$ *and* $\Gamma \vdash M = N \star A$. *Then*

1. *If* $\Gamma, x \star A \vdash O = P : B$, *then* $\Gamma \vdash [M/x]O = [N/x]P : [M/x]B$.
2. *If* $\Gamma, x \star A \vdash B = C : K$, *then* $\Gamma \vdash [M/x]B = [N/x]C : [M/x]K$.
3. *If* $\Gamma, x \star A \vdash K = L : \mathsf{kind}$, *then* $\Gamma \vdash [M/x]K = [N/x]L : \mathsf{kind}$.

Proof Direct. We show the first part; the other two are similar.

$\Gamma, x \star A \vdash O = P : B$	Assumption.
$\Gamma \vdash M = N \star A$	Assumption.
$\Gamma \vdash M \star A$	Validity.
$\Gamma \vdash N \star A$	Validity.
$\Gamma \vdash [M/x]O = [M/x]P : [M/x]B$	Substitution.
$\Gamma, x \star A \vdash P : B$	Validity.
$\Gamma \vdash [M/x]P = [N/x]P : [M/x]B$	Functionality.
$\Gamma \vdash [M/x]O = [N/x]P : [M/x]B$	oe_trans.

■

Lemma 4.14 (Typing Inversion) *Suppose* $\vdash \Gamma : \mathsf{ctx}$. *Then*

1. *If* $\Gamma \vdash x : A$ *then* $x : B \in \Gamma$ *and* $\Gamma \vdash A = B : \mathsf{type}$ *for some* B.
2. *If* $\Gamma \vdash c : A$ *then* $c : B \in \Sigma$ *and* $\Gamma \vdash A = B : \mathsf{type}$ *for some* B.
3. *If* $\Gamma \vdash M_1 * M_2 : A$ *then* $\Gamma \vdash M_1 : \Pi x{\star}A_2.A_1$, $\Gamma \vdash M_2 \star A_2$, *and* $\Gamma \vdash [M_2/x]A_1 = A : \mathsf{type}$ *for some* A_1 *and* A_2.
4. *If* $\Gamma \vdash \lambda x{\star}A.M : B$, *then* $\Gamma \vdash B = \Pi x{\star}A.A' : \mathsf{type}$, $\Gamma \vdash A : \mathsf{type}$, *and* $\Gamma, x \star A \vdash M : A'$.
5. *If* $\Gamma \vdash \Pi x{\star}A_1.A_2 : K$, *then* $\Gamma \vdash K = \mathsf{type} : \mathsf{kind}$, $\Gamma \vdash A_1 : \mathsf{type}$, *and* $\Gamma, x \star A_1 \vdash A_2 : \mathsf{type}$.
6. *If* $\Gamma \vdash a : K$, *then* $a : L \in \Sigma$ *and* $\Gamma \vdash K = L : \mathsf{kind}$ *for some* L.
7. *If* $\Gamma \vdash A * M : K$ *then* $\Gamma \vdash A : \Pi x{\star}A_1.K_2$, $\Gamma \vdash M \star A_1$, *and* $\Gamma \vdash [M/x]K_2 = K : \mathsf{kind}$ *for some* K_1 *and* K_2.

8. *If* $\Gamma \vdash \Pi x{\star}A_1.K : \mathsf{kind}$, *then* $\Gamma \vdash A_1 : \mathsf{type}$, *and* $\Gamma, x \star A_1 \vdash K : \mathsf{kind}$.

Proof Straightforward induction, using validity to be able to apply reflexivity. ∎

Lemma 4.15 (Redundancy) *The typing premises in rules* oe_red, fe_pi, *and* ke_pi *enclosed in {braces} are redundant. That is, the rules without the braced premises are admissible.*

Proof Straightforward from validity. ∎

Lemma 4.16 (Equality Inversion)

1. *If* $\Gamma \vdash A = \Pi x{\star}B_1.B_2 : \mathsf{type}$ *or* $\Gamma \vdash \Pi x{\star}B_1.B_2 = A : \mathsf{type}$, *then* A *is of the form* $\Pi x{\star}A_1.A_2$ *such that* $\Gamma \vdash A_1 = B_1 : \mathsf{type}$ *and* $\Gamma, x : A_1 \vdash A_2 = B_2 : \mathsf{type}$.

2. *If* $\Gamma \vdash K = \mathsf{type} : \mathsf{kind}$ *or* $\Gamma \vdash \mathsf{type} = K : \mathsf{kind}$, *then* K *is* type.

3. *If* $\Gamma \vdash K = \Pi x{\star}B_1.L_2 : \mathsf{kind}$ *or* $\Gamma \vdash \Pi x{\star}B_1.L_2 = K : \mathsf{kind}$, *then* K *is of the form* $\Pi x{\star}A_1.K_2$ *such that* $\Gamma \vdash A_1 = B_1 : \mathsf{type}$ *and* $\Gamma, x : A_1 \vdash K_2 = L_2 : \mathsf{type}$.

Proof By induction. ∎

Lemma 4.17 (Injectivity of Products)

1. *If* $\Gamma \vdash \Pi x{\star}A_1.A_2 = \Pi x{\star}B_1.B_2 : \mathsf{type}$ *then* $\Gamma \vdash A_1 = B_1 : \mathsf{type}$ *and* $\Gamma, x \star A_1 \vdash A_2 = B_2 : \mathsf{type}$.

2. *If* $\Gamma \vdash \Pi x{\star}A.K = \Pi x{\star}B.L : \mathsf{kind}$ *then* $\Gamma \vdash A = B : \mathsf{type}$ *and* $\Gamma, x \star A \vdash K = L : \mathsf{kind}$.

Proof Immediate from Lemma 4.16. ∎

5 Algorithmic Equality

We outline below an algorithm adapted directly from [Pfe01] and [HP01] for deciding equality of well-typed terms. It follows that the typing judgment and all other judgments are decidable. The additional typing assumptions on rules pot and pe turn out to incur no extra computational cost for deciding the derivability $\Gamma \vdash M : A$. Intuitively this is because the only way a judgment of the form $\Gamma \vdash N \div A$ or $\Gamma \vdash N_1 = N_2 \div A$ can have an observable effect is under an irrelevant application. The typing rule for functions with an irrelevant argument, however, already requires its argument's type to be valid in an unpromoted context.

To make proving transitivity of algorithmic equality feasible, we define *approximate* types which omit dependency information. Carrying only approximate types is sufficient for the type-directedness of the algorithm. For each constant family a, we posit a simple base type a^-, and use α to denote the simple base types.

Simple Kinds	κ	$::= \mathsf{type}^- \mid \tau \to \kappa \mid \tau \xrightarrow{\div} \kappa$
Simple Types	τ	$::= \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \xrightarrow{\div} \tau_2$
Simple Contexts	Δ	$::= \cdot \mid \Delta, x \star \tau$

The symbol $\stackrel{\star}{\to}$ in what follows denotes either $\to$ or $\stackrel{\div}{\to}$, in parallel with the earlier conventions about $\star$ and $*$. The *erasure* $(-)^-$ of a family (resp. kind, context) to a simple type (resp. simple kind, simple context) is defined by

$$
\begin{array}{rcl}
(a)^- & = & a^- \\
(A * M)^- & = & A^- \\
(\Pi x{\star}A_1.A_2) & = & A_1^- \stackrel{\star}{\to} A_2^- \\
(\Pi x{\star}A.K) & = & A^- \stackrel{\star}{\to} K^- \\
\cdot^- & = & \cdot \\
(\Gamma, x \star A)^- & = & \Gamma^-, x \star A^-
\end{array}
$$

The erasure of a type or kind is invariant under equality and substitution.

Lemma 5.1 (Erasure Preservation)

1. *If* $\Gamma \vdash A = B : K$*, then* A^- *is identical to* B^-*.*
2. *If* $\Gamma \vdash K = L : \mathsf{kind}$*, then* K^- *is identical to* L^-*.*
3. *If* $\Gamma, x \star A \vdash B : K$*, then* B^- *is identical to* $(B[M/x])^-$*.*
4. *If* $\Gamma, x \star A \vdash K : \mathsf{kind}$*, then* K^- *is identical to* $(K[M/x])^-$*.*

Proof Straightforward induction. ∎

5.1 Judgments

The equality algorithm is given by three judgments (and analogues at types and kinds):

Weak Head Reduction	$M_1 \xrightarrow{whr} M_2$	Σ is a valid signature.
Algorithmic Equality	$M_1 \Longleftrightarrow M_2 : \tau$	M_1 is algorithmically equal to M_2 at simple type τ
Structural Equality	$M_1 \longleftrightarrow M_2 : \tau$	M_1 is structurally equal to M_2 at simple type τ

Algorithmically, weak head reduction takes M_1 as input and returns M_2, algorithmic equality takes Δ, M_1, M_2, τ as input and succeeds or fails, and structural equality takes Δ, M_1, M_2 as input and either succeeds returning an approximate type τ, or fails.

5.1.1 Weak Head Reduction

$$
\frac{}{(\lambda x{\star}A_1.M_2) * M_1 \xrightarrow{whr} [M_1/x]M_2}\ \mathsf{whr\_beta}
$$

$$
\frac{M_1 \xrightarrow{whr} M_1'}{M_1 * M_2 \xrightarrow{whr} M_1' * M_2}\ \mathsf{whr\_head}
$$

5.1.2 Algorithmic Object Equality

$$\frac{M \xrightarrow{whr} M' \qquad \Delta \vdash M' \Longleftrightarrow N : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha}\ \textsf{ae\_whrl}$$

$$\frac{N \xrightarrow{whr} N' \qquad \Delta \vdash M \Longleftrightarrow N' : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha}\ \textsf{ae\_whrr}$$

$$\frac{\Delta, x \star \tau_1 \vdash M * x \Longleftrightarrow N * x : \tau_2}{\Delta \vdash M \Longleftrightarrow N : \tau_1 \xrightarrow{\star} \tau_2}\ \textsf{ae\_ext}$$

$$\frac{\Delta \vdash M \longleftrightarrow N : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha}\ \textsf{ae\_str}$$

5.1.3 Structural Object Equality

$$\frac{x : \tau \in \Delta}{\Delta \vdash x = x : \tau}\ \textsf{se\_var}$$

$$\frac{c : A \in \Sigma}{\Delta \vdash c = c : A^-}\ \textsf{se\_const}$$

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \to \tau_1 \qquad \Delta \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Delta \vdash M_1\ M_2 \longleftrightarrow N_1\ N_2 : \tau_1}\ \textsf{se\_app}$$

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \xrightarrow{\div} \tau_1}{\Delta \vdash M_1 \circ M_2 \longleftrightarrow N_1 \circ N_2 : \tau_1}\ \textsf{se\_iapp}$$

5.1.4 Algorithmic Family Equality

$$\frac{\Delta, x \star \tau \vdash A * x \Longleftrightarrow B * x : \kappa}{\Delta \vdash A \Longleftrightarrow B : \tau \xrightarrow{\star} \kappa}\ \textsf{afe\_ext}$$

$$\frac{\Delta \vdash A \longleftrightarrow B : \textsf{type}^-}{\Delta \vdash A \Longleftrightarrow B : \textsf{type}^-}\ \textsf{afe\_str}$$

$$\frac{\Delta \vdash A_1 \Longleftrightarrow B_1 : \textsf{type}^- \qquad \Delta, x \star A_1 \vdash A_2 \Longleftrightarrow B_2 : \textsf{type}^-}{\Delta \vdash \Pi x{\star}A_1.A_2 \Longleftrightarrow \Pi x{\star}B_1.B_2 : \textsf{type}^-}\ \textsf{afe\_pi}$$

5.1.5 Structural Family Equality

$$\frac{a : K \in \Sigma}{\Delta \vdash a = a : K^-}\ \textsf{sfe\_const}$$

$$\frac{\Delta \vdash A \longleftrightarrow B : \tau \to \kappa \qquad \Delta \vdash M \Longleftrightarrow N : \tau}{\Delta \vdash A\ M \longleftrightarrow B\ N : \kappa}\ \textsf{sfe\_app}$$

$$\frac{\Delta \vdash A \longleftrightarrow B : \tau \xrightarrow{\div} \kappa}{\Delta \vdash A \circ M \longleftrightarrow B \circ N : \kappa}\ \textsf{sfe\_iapp}$$

5.1.6 Algorithmic Kind Equality

$$\frac{}{\Delta \vdash \mathsf{type} \Longleftrightarrow \mathsf{type} : \mathsf{kind}^-}\ \mathsf{ake\_type}$$

$$\frac{\Delta \vdash A \Longleftrightarrow B : \mathsf{type}^- \qquad \Delta, x \star A \vdash K \Longleftrightarrow L : \mathsf{type}^-}{\Delta \vdash \Pi x{\star}A.K \Longleftrightarrow \Pi x{\star}B.L : \mathsf{type}^-}\ \mathsf{ake\_pi}$$

5.2 Algorithmic Equality Lemmas

We state without proof some basic properties of the algorithm that are required to show correctness. Most proofs are easy generalizations of those in [HP01].

Lemma 5.2 (Weakening) *If $\Delta, \Delta' \vdash J$ then $\Delta, x : \tau, \Delta' \vdash J$, for any algorithmic equality judgment J.*

Lemma 5.3 (Determinacy)

1. *If $M \overset{whr}{\longrightarrow} M'$ and $M \overset{whr}{\longrightarrow} M''$, then M' is identical to M''.*
2. *If $\Delta \vdash M \longleftrightarrow N : \tau$, then there is no M' such that $M \overset{whr}{\longrightarrow} M'$.*
3. *If $\Delta \vdash M \longleftrightarrow N : \tau$, then there is no N' such that $N \overset{whr}{\longrightarrow} N'$.*
4. *If $\Delta \vdash M \longleftrightarrow N : \tau$ and $\Delta \vdash M \longleftrightarrow N : \tau'$, then τ is identical to τ'.*
5. *If $\Delta \vdash A \longleftrightarrow B : \kappa$ and $\Delta \vdash A \longleftrightarrow B : \kappa'$, then κ is identical to κ'.*

Lemma 5.4 (Symmetry) *If $\Delta \vdash M \Longleftrightarrow N : \tau$, then $\Delta \vdash N \Longleftrightarrow M : \tau$, and similarly for $\longleftrightarrow$ and families and kinds.*

Lemma 5.5 (Transitivity) *If $\Delta \vdash M \Longleftrightarrow N : \tau$ and $\Delta \vdash N \Longleftrightarrow O : \tau$ then $\Delta \vdash M \Longleftrightarrow O : \tau$, and similarly for $\longleftrightarrow$ and families and kinds.*

5.3 Completeness of Algorithmic Equality

Completeness is the statement that definitional equality implies algorithmic equality, in other words

$$\text{If } \Gamma \vdash M = N : A, \text{ then } \Gamma^- \vdash M \Longleftrightarrow N : A^-$$

A direct induction proof fails at the application case, so we use a standard logical relations argument. We define Kripke logical relations $\Gamma \vdash M = N \in [\![\tau]\!]$, $\Gamma \vdash A = B \in [\![\kappa]\!]$, $\Gamma \vdash \sigma = \theta \in [\![\Theta]\!]$ by recursion on τ, κ, Θ as follows:

1. $\Delta \vdash M = N \in [\![\alpha]\!]$ iff $\Delta \vdash M \Longleftrightarrow N : \alpha$.
2. $\Delta \vdash M = N \in [\![\tau_1 \to \tau_2]\!]$ iff for all Δ' extending Δ and for all M', N' such that $\Delta' \vdash M' = N' \in [\![\tau_1]\!]$ we have $\Delta' \vdash M\ M' = N\ N' \in [\![\tau_2]\!]$.

3. $\Delta \vdash M = N \in [\![\tau_1 \stackrel{\div}{\rightarrow} \tau_2]\!]$ iff for all Δ' extending Δ and for all M', N' we have $\Delta' \vdash M \circ M' = N \circ N' \in [\![\tau_2]\!]$.

4. $\Delta \vdash A = B \in [\![\mathsf{type}^-]\!]$ iff $\Delta \vdash A \Longleftrightarrow B : \mathsf{type}^-$.

5. $\Delta \vdash A = B \in [\![\tau \rightarrow \kappa]\!]$ iff for all Δ' extending Δ and for all M', N' such that $\Delta' \vdash M' = N' \in [\![\tau]\!]$ we have $\Delta' \vdash A\ M' = B\ N' \in [\![\kappa]\!]$.

6. $\Delta \vdash A = B \in [\![\tau \stackrel{\div}{\rightarrow} \kappa]\!]$ iff for all Δ' extending Δ and for all M', N' we have $\Delta' \vdash A \circ M' = B \circ N' \in [\![\kappa]\!]$.

7. $\Delta \vdash \sigma = \theta \in [\![\cdot]\!]$ iff $\sigma = \cdot$ and $\theta = \cdot$.

8. $\Delta \vdash \sigma = \theta \in [\![\Theta, x : \tau]\!]$ iff $\sigma = (\sigma', M/x)$ and $\theta = (\theta', N/x)$ such that $\Delta \vdash \sigma' = \theta' \in [\![\Theta]\!]$ and $\Delta \vdash M = N \in [\![\tau]\!]$.

9. $\Delta \vdash \sigma = \theta \in [\![\Theta, x \div \tau]\!]$ iff $\sigma = (\sigma', M/x)$ and $\theta = (\theta', N/x)$ such that $\Delta \vdash \sigma' = \theta' \in [\![\Theta]\!]$.

and thereby reduce completeness to showing

1. If $\Gamma \vdash M = N : A$, then $\Gamma^- \vdash M = N \in [\![A^-]\!]$.

2. If $\Delta \vdash M = N \in [\![\tau]\!]$, then $\Delta \vdash M \Longleftrightarrow N : \tau$.

5.3.1 Related Terms are Algorithmically Equal

The only structural lemma we need concerning the logical relation is Weakening:

Lemma 5.6 (Weakening) *For all logical relations R, if $\Delta, \Delta' \vdash R$, then $\Delta, x \star \tau, \Delta' \vdash R$.*

Proof Straightforward induction. ■

Lemma 5.7 (Related Terms are Algorithmically Equal)

1. If $\Delta \vdash M = N \in [\![\tau]\!]$, then $\Delta \vdash M \Longleftrightarrow N : \tau$

2. If $\Delta \vdash A = B \in [\![\kappa]\!]$, then $\Delta \vdash A \Longleftrightarrow B : \kappa$

3. If $\Delta \vdash M \longleftrightarrow N : \tau$, then $\Delta \vdash M = N \in [\![\tau]\!]$.

4. If $\Delta \vdash A \longleftrightarrow B : \kappa$, then $\Delta \vdash A = B \in [\![\kappa]\!]$.

Proof By induction on τ, κ. We show only the cases that differ from [HP01].

Case: $\tau = \tau_1 \stackrel{\div}{\rightarrow} \tau_2$, part 1. By assumption, $\Delta \vdash M = N \in [\![\tau_1 \stackrel{\div}{\rightarrow} \tau_2]\!]$. By definition of $[\![\tau_1 \stackrel{\div}{\rightarrow} \tau_2]\!]$, we have $\Delta, x \div \tau_1 \vdash M \circ x = N \circ x \in [\![\tau_2]\!]$. By the induction hypothesis (part 1) on τ_2 we know $\Delta, x \div \tau_1 \vdash M \circ x \Longleftrightarrow N \circ x : \tau_2$. By $\mathsf{ae\_ext}$ we conclude $\Delta \vdash M \Longleftrightarrow N : \tau_1 \stackrel{\div}{\rightarrow} \tau_2$.

Case: $\kappa = \tau \xrightarrow{\div} \kappa'$, part 2. By assumption, $\Delta \vdash A = B \in [\![\tau \xrightarrow{\div} \kappa']\!]$. By definition of $[\![\tau \xrightarrow{\div} \kappa']\!]$, we have $\Delta, x \div \tau \vdash A \circ x = B \circ x \in [\![\kappa']\!]$. By the induction hypothesis (part 2) on κ' we know $\Delta, x \div \tau \vdash A \circ x \Longleftrightarrow B \circ x : \kappa'$. By afe_ext we conclude $\Delta \vdash A \Longleftrightarrow B : \tau \xrightarrow{\div} \kappa'$.

Case: $\tau = \tau_1 \xrightarrow{\div} \tau_2$, part 3. By assumption, $\Delta \vdash M \longleftrightarrow N : \tau_1 \xrightarrow{\div} \tau_2$. Suppose Δ' extends Δ and let M', N' be given. By se_iapp we have $\Delta \vdash M \circ M' \longleftrightarrow N \circ N' : \tau_2$, and weakening of the logical relation gives $\Delta' \vdash M \circ M' \longleftrightarrow N \circ N' : \tau_2$. By the induction hypothesis (part 3) on τ_2, we conclude $\Delta' \vdash M \circ M' = N \circ N' \in [\![\tau_2]\!]$, and so $\Delta \vdash M = N \in [\![\tau_1 \xrightarrow{\div} \tau_2]\!]$.

Case: $\kappa = \tau \xrightarrow{\div} \kappa'$, part 4. By assumption, $\Delta \vdash A \longleftrightarrow B : \tau \xrightarrow{\div} \kappa'$. Suppose Δ' extends Δ and let M', N' be given. By sfe_iapp we have $\Delta \vdash A \circ M' \longleftrightarrow B \circ N' : \kappa'$, and weakening of the logical relation gives $\Delta' \vdash A \circ M' \longleftrightarrow B \circ N' : \kappa'$. By the induction hypothesis (part 4) on κ', we conclude $\Delta' \vdash A \circ M' = B \circ N' \in [\![\kappa']\!]$, and so $\Delta \vdash A = B \in [\![\tau \xrightarrow{\div} \kappa']\!]$.

5.3.2 Definitionally Equal Terms are Related

Several more lemmas concerning the logical relation are required. We then show that definitionally equal terms under equal substitutions are in the logical relation, and the desired result follows from the identity substitution being equal to itself.

Lemma 5.8 (Closure under Head Expansion)

1. *If* $M \xrightarrow{whr} M'$ *and* $\Delta \vdash M' = N \in [\![\tau]\!]$, *then* $\Delta \vdash M = N \in [\![\tau]\!]$.
2. *If* $N \xrightarrow{whr} N'$ *and* $\Delta \vdash M = N' \in [\![\tau]\!]$, *then* $\Delta \vdash M = N \in [\![\tau]\!]$.

Proof By induction on τ. We show the only new case for part 1. Part 2 is symmetric.

Case: $\tau = \tau_1 \xrightarrow{\div} \tau_2$. By assumption, $M \xrightarrow{whr} M'$ and $\Delta \vdash M' = N \in [\![\tau_1 \xrightarrow{\div} \tau_2]\!]$. Suppose Δ' extends Δ and let M_1, N_1 be given. By definition of $\tau_1 \xrightarrow{\div} \tau_2$, we know $\Delta' \vdash M' \circ M_1 = N \circ N_1 \in [\![\tau_2]\!]$. By whr_head, we have $M \circ M_1 \xrightarrow{whr} M' \circ M_1$, so the induction hypothesis on τ_2 gives $\Delta' \vdash M \circ M_1 = N \circ N_1 \in [\![\tau_2]\!]$. By definition of $\tau_1 \xrightarrow{\div} \tau_2$ we conclude $\Delta \vdash M = N \in [\![\tau_1 \xrightarrow{\div} \tau_2]\!]$.

■

Lemma 5.9 (Symmetry)

1. *If* $\Delta \vdash M = N \in [\![\tau]\!]$, *then* $\Delta \vdash N = M \in [\![\tau]\!]$.
2. *If* $\Delta \vdash A = B \in [\![\kappa]\!]$, *then* $\Delta \vdash B = A \in [\![\kappa]\!]$.
3. *If* $\Delta \vdash \sigma = \theta \in [\![\Theta]\!]$, *then* $\Delta \vdash \theta = \sigma \in [\![\Theta]\!]$.

Proof Straightforward induction on τ, κ, Θ, using symmetry of algorithmic equality at simple base types. ■

Lemma 5.10 (Transitivity)

1. *If* $\Delta \vdash M = N \in [\![\tau]\!]$ *and* $\Delta \vdash N = O \in [\![\tau]\!]$*, then* $\Delta \vdash M = O \in [\![\tau]\!]$*.*
2. *If* $\Delta \vdash A = B \in [\![\kappa]\!]$ *and* $\Delta \vdash B = C \in [\![\kappa]\!]$*, then* $\Delta \vdash A = C \in [\![\kappa]\!]$*.*
3. *If* $\Delta \vdash \sigma = \theta \in [\![\Theta]\!]$ *and* $\Delta \vdash \theta = \delta \in [\![\Theta]\!]$*, then* $\Delta \vdash \sigma = \delta \in [\![\Theta]\!]$*.*

Proof Straightforward induction on τ, κ, Θ, using transitivity of algorithmic equality at simple base types. We show the new case for part 1.

Case: $\tau = \tau_1 \stackrel{\div}{\to} \tau_2$. By assumption, $\Delta \vdash M = N \in [\![\tau_1 \stackrel{\div}{\to} \tau_2]\!]$ and $\Delta \vdash N = O \in [\![\tau_1 \stackrel{\div}{\to} \tau_2]\!]$. Suppose Δ' extends Δ and let M_1, O_1 be given. By definition of $[\![\tau_1 \stackrel{\div}{\to} \tau_2]\!]$, we know $\Delta' \vdash M \circ M_1 = N \circ M_1 \in [\![\tau_2]\!]$ and $\Delta' \vdash N \circ M_1 = O \circ O_1 \in [\![\tau_2]\!]$. The induction hypothesis (part 1) applied to τ_2 gives $\Delta' \vdash M \circ M_1 = O \circ O_1 \in [\![\tau_2]\!]$. By definition of $[\![\tau_1 \stackrel{\div}{\to} \tau_2]\!]$, we have shown $\Delta \vdash M = O \in [\![\tau_1 \stackrel{\div}{\to} \tau_2]\!]$.

■

Lemma 5.11 (Definitionally Equal Terms are Related under Related Substitutions)

1. *If* $\Gamma \vdash M = N : A$ *and* $\Delta \vdash \sigma = \theta \in [\![\Gamma^-]\!]$*, then* $\Delta \vdash M[\sigma] = N[\theta] \in [\![A^-]\!]$*.*
2. *If* $\Gamma \vdash A = B : K$ *and* $\Delta \vdash \sigma = \theta \in [\![\Gamma^-]\!]$*, then* $\Delta \vdash A[\sigma] = B[\theta] \in [\![K^-]\!]$*.*

Proof By induction on the equality derivation $\mathcal{D}$. We show only the object-level cases that are different from [HP01].

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash M_1 = M_1' : \Pi x{\div}A.B} \quad \overset{\mathcal{D}_2}{\Gamma \vdash M_2 = M_2' \div A}}{\Gamma \vdash M_1 \circ M_2 = M_1' \circ M_2' : [M_2/x]B}\ \mathsf{oe\_app}$$

By the induction hypothesis on $\mathcal{D}_1$ we know $\Delta \vdash M_1[\sigma] = M_1'[\theta] \in [\![A^- \stackrel{\div}{\to} B^-]\!]$. By definition of $[\![A^- \stackrel{\div}{\to} B^-]\!]$ we have $\Delta \vdash M_1[\sigma] \circ M_2[\sigma] = M_1'[\theta] \circ M_2'[\theta] \in [\![B^-]\!]$. Erasure preservation (using validity inversion of products to get the required $\Gamma, x{\div}A \vdash B : \mathsf{type}$) and properties of substitution imply $\Delta \vdash (M_1 \circ M_2)[\sigma] = (M_1' \circ M_2')[\theta] \in [\![([M_2/x]B)^-]\!]$,

Case: $\mathcal{D} =$

$$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A = A' : \mathsf{type}} \quad \overset{\mathcal{D}_2}{\Gamma \vdash A = A'' : \mathsf{type}} \quad \overset{\mathcal{D}_3}{\Gamma, x \div A \vdash M = M' : B}}{\Gamma \vdash \lambda x{\div}A'.M = \lambda x{\div}A''.M' : \Pi x{\div}A.B}\ \mathsf{oe\_lam}$$

Towards showing that $\Delta \vdash (\lambda x{\div}A'.M)[\sigma] = (\lambda x{\div}A''.M)[\theta] \in [\![A^- \stackrel{\div}{\to} B^-]\!]$, suppose that Δ' extends Δ and let N, N' be given. By definition of $[\![\Gamma^-]\!]$ and weakening, we know $\Delta' \vdash (\sigma, N/x) = (\theta, N'/x) \in [\![(\Gamma, x \div A')^-]\!]$. By the induction hypothesis on $\mathcal{D}_3$ we obtain $\Delta' \vdash M[\sigma, N/x] = N[\theta, N'/x] \in [\![B^-]\!]$. Using closure of the logical relation under head expansion twice we see $\Delta' \vdash ((\lambda x{\div}A'.M)[\sigma]) \circ N = ((\lambda x{\div}A''.M)[\theta]) \circ N' \in [\![B^-]\!]$. The definition of $[\![A^- \xrightarrow{\div} B^-]\!]$ implies $\Delta \vdash (\lambda x{\div}A'.M)[\sigma] = (\lambda x{\div}A''.M)[\theta] \in [\![A^- \xrightarrow{\div} B^-]\!]$.

Case: $\mathcal{D} =$

$$\frac{\begin{array}{ccc}\mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ \Gamma \vdash A : \mathsf{type} & \Gamma \vdash M, N : \Pi x{\div}A.B & \Gamma, x \div A \vdash M \circ x = N \circ x : B\end{array}}{\Gamma \vdash M = N : \Pi x{\div}A.B}\ \mathsf{oe\_ext}$$

Towards showing that $\Delta \vdash M[\sigma] = N[\theta] \in [\![A^- \xrightarrow{\div} B^-]\!]$, suppose that Δ' extends Δ and let M', N' be given. By definition of $[\![\Gamma^-]\!]$ and weakening, we know $\Delta' \vdash (\sigma, M'/x) = (\theta, N'/x) \in [\![(\Gamma, x \div A')^-]\!]$. By the induction hypothesis on $\mathcal{D}_3$ we obtain $\Delta' \vdash (M \circ x)[\sigma, M'/x] = (N \circ x)[\theta, N'/x] \in [\![B^-]\!]$. In other words, $\Delta' \vdash M[\sigma] \circ M' = N[\theta] \circ N' \in [\![B^-]\!]$. The definition of $[\![A^- \xrightarrow{\div} B^-]\!]$ implies $\Delta \vdash M[\sigma] = N[\theta] \in [\![A^- \xrightarrow{\div} B^-]\!]$.

Case:

$$\mathcal{D} = \frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma, x \div A \vdash M_2 = M_2' : B & \Gamma \vdash M_1 = M_1' \div A\end{array}}{\Gamma \vdash (\lambda x{\div}A.M_2) \circ M_1 = [M_1'/x]M_2' : [M_1/x]B}\ \mathsf{oe\_red}$$

By definition of $[\![\Gamma^-]\!]$ we have $\Delta \vdash (\sigma, M_1[\sigma]/x) = (\sigma, M_1'[\theta]/x) \in [\![(\Gamma, x \div A)^-]\!]$. By the induction hypothesis on $\mathcal{D}_1$, we infer $\Delta \vdash M_2[\sigma, M_1[\sigma]/x] = M_2'[\theta, M_1'[\theta]/x] \in [\![B^-]\!]$. By properties of substitution, this means that $\Delta \vdash (M_1[\sigma]/x)(M_2[\sigma]) = M_2'[\theta, M_1'[\theta]/x] \in [\![B^-]\!]$. Closure of the logical relation under head expansion implies that we have $\Delta \vdash (\lambda x{\div}A[\sigma].M_2[\sigma]) \circ (M_1[\sigma]) = M_2'[\theta, M_1'[\theta]/x] \in [\![B^-]\!]$. And applying properties of substitution again, we see $\Delta \vdash ((\lambda x{\div}A.M_2) \circ M_1)[\sigma] = ([M_1'/x]M_2')[\theta] \in [\![B^-]\!]$. Validity applied to $\Gamma, x \div A \vdash M_2 = M_2' : B$ gives $\Gamma, x \div A \vdash B : \mathsf{type}$, and so erasure preservation implies $\Delta \vdash ((\lambda x{\div}A.M_2) \circ M_1)[\sigma] = ([M_1'/x]M_2')[\theta] \in [\![([M_1/x]B)^-]\!]$, as required.

■

Lemma 5.12 (Identity Substitutions are Related) *For any context Γ we have $\Gamma^- \vdash \mathrm{id}_\Gamma = \mathrm{id}_\Gamma \in [\![\Gamma^-]\!]$.*

Proof By induction using part 3 of Lemma 5.7 to infer $\Gamma^-, x : A^- \vdash x = x \in [\![A^-]\!]$ from $\Gamma, x : A \vdash x \longleftrightarrow x : A$. ■

Lemma 5.13 (Definitionally Equal Terms are Related)

1. *If $\Gamma \vdash M = N : A$, then $\Delta \vdash M = N \in [\![A^-]\!]$.*

2. *If* $\Gamma \vdash A = B : K$*, then* $\Delta \vdash A = B \in [\![K^-]\!]$.

Proof By Lemmas 5.11 and 5.12. ■

Theorem 5.14 (Completeness of Algorithmic Equality)

1. *If* $\Gamma \vdash M = N : A$*, then* $\Delta \vdash M \Longleftrightarrow N : [\![A^-]\!]$.

2. *If* $\Gamma \vdash A = B : K$*, then* $\Delta \vdash A \Longleftrightarrow B : [\![K^-]\!]$.

Proof By Lemmas 5.7 and 5.13. ■

5.4 Soundness of Algorithmic Equality

Given the lemmas of section 4.5 and subject reduction, soundness admits a direct induction proof.

Lemma 5.15 (Subject Reduction) *If* $M \xrightarrow{whr} M'$ *and* $\Gamma \vdash M : A$*, then* $\Gamma \vdash M' : A$ *and* $\Gamma \vdash M = M' : A$.

Proof By induction on $\mathcal{D} :: M \xrightarrow{whr} M'$. There are two new cases (the irrelevant version of each rule defining weak head normalization) but their proof is a straightforward generalization of the cases already in [HP01]. The only possible difficulty comes when we need to show something of the form $\Gamma \vdash M_1 = M_1 \div A$ instead of $\Gamma \vdash M_1 = M_1 : A$. Fortunately in both cases the fact that $\Gamma \vdash A : \mathsf{type}$ is easily obtained from the application of existing inversion and validity lemmas to the typing assumptions. We can therefore use Corollary 4.6 to see $\Gamma^\oplus \vdash M_1 = M_1 : A$ and derive

$$\frac{\Gamma^\oplus \vdash M_1 = M_1 : A \qquad \Gamma^\oplus \vdash M_1 = M_1 : A \qquad \Gamma \vdash A : \mathsf{type}}{\Gamma \vdash M_1 = M_1 \div A}\,\mathsf{poe}$$

■

Theorem 5.16 (Soundness of Algorithmic Equality)

1. *If* $\Gamma \vdash M, N : A$ *and* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$*, then* $\Gamma \vdash M = N : A$.

2. *If* $\Gamma \vdash M : A$*,* $\Gamma \vdash N : B$ *and* $\Gamma^- \vdash M \longleftrightarrow N : \tau$*, then* $\Gamma \vdash M = N : A$*,* $\Gamma \vdash A = B : \mathsf{type}$*, and* τ, A^-, B^- *are identical.*

3. *If* $\Gamma \vdash A, B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$*, then* $\Gamma \vdash A = B : K$.

4. *If* $\Gamma \vdash A : K$*,* $\Gamma \vdash B : L$ *and* $\Gamma^- \vdash A \longleftrightarrow B : \kappa$*, then* $\Gamma \vdash A = B : K$*,* $\Gamma \vdash K = L : \mathsf{type}$*, and* κ, K^-, L^- *are identical.*

5. *If* $\Gamma \vdash K, L : \mathsf{kind}$ *and* $\Gamma^- \vdash K \Longleftrightarrow L : \mathsf{kind}^-$*, then* $\Gamma \vdash K = L : \mathsf{kind}$.

Proof By induction on the equality derivation. We show only the new object-level cases.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \stackrel{\div}{\to} \tau_1\end{array}}{\Delta \vdash M_1 \circ M_2 \longleftrightarrow N_1 \circ N_2 : \tau_1}\,\textsf{se\_iapp}$$

By assumption $\Gamma \vdash M_1 \circ M_2 : A$ and $\Gamma \vdash N_1 \circ N_2 : B$ Inversion implies the existence of A_1, A_2, B_1, B_2 such that

$$\Gamma \vdash M_1 : \Pi x{\star}A_2.A_1 \qquad (*_M)$$

$$\Gamma \vdash M_2 \div A_2$$

$$\Gamma \vdash [M_2/x]A_1 = A : \textsf{type}$$

and

$$\Gamma \vdash N_1 : \Pi x{\star}B_2.B_1 \qquad (*_N)$$

$$\Gamma \vdash N_2 \div B_2$$

$$\Gamma \vdash [N_2/x]B_1 = B : \textsf{type}$$

By applying validity and inversion to $(*_M)$ and $(*_N)$ we get

$$\Gamma \vdash \Pi x{\star}A_2.A_1 : \textsf{type}$$

$$\Gamma, x \star A_2 \vdash A_1 : \textsf{type}$$

$$\Gamma \vdash \Pi x{\star}B_2.B_1 : \textsf{type}$$

$$\Gamma, x \star B_2 \vdash B_1 : \textsf{type}$$

By the induction hypothesis on $\mathcal{D}_1$, there is τ such that

$$\Gamma \vdash M_1 = N_1 : \Pi x{\star}A_2.A_1$$

$$\Gamma \vdash \Pi x{\star}A_2.A_1 = \Pi x{\star}B_2.B_1 : \textsf{type}$$

$$(\Pi x{\star}A_2.A_1)^-, (\Pi x{\star}B_2.B_1)^-, \tau \text{ identical.}$$

so clearly τ must be of the form $\tau_2 \stackrel{\div}{\to} \tau_1$. Injectivity of products and Corollary 4.6 imply that $\Gamma^\oplus \vdash A_2 = B_2 : \textsf{type}$, so we can apply inversion to $\Gamma \vdash N \div B_2$ to see that $\Gamma^\oplus \vdash N : B_2$ and derive

$$\dfrac{\Gamma^\oplus \vdash N : B_2 \qquad \dfrac{\Gamma^\oplus \vdash A_2 = B_2 : \textsf{type}}{\Gamma^\oplus \vdash B_2 = A_2 : \textsf{type}}\,\textsf{fe\_sym}}{\Gamma^\oplus \vdash N : A_2}\,\textsf{ot\_conv}$$

Now by inversion also $\Gamma^{\oplus} \vdash M_2 : A_2$ and $\Gamma \vdash A_2 : \mathsf{type}$, so by poe we know $\Gamma \vdash M_2 = N_2 \div A_2$. We can now derive

$$\dfrac{\Gamma \vdash [M_2/x]A_1 = A : \mathsf{type} \quad \dfrac{\Gamma \vdash M_1 = N_1 : \Pi x \div A_2.A_1 \qquad \Gamma \vdash M_2 = N_2 \div A_2}{\Gamma \vdash M_1 \circ M_2 = N_1 \circ N_2 : [M_2/x]A_1}\,\mathsf{oe\_app}}{\Gamma \vdash M_1 \circ M_2 = N_1 \circ N_2 : A}\,\mathsf{oe\_conv}$$

By injectivity of products and functionality we have

$$\Gamma \vdash [M_2/x]A_1 = [N_2/x]B_1 : \mathsf{type}$$

and all of $A^-, A_1^-, B_1^-, B^-, \tau_1$ are identical by erasure preservation.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Delta, x \div \tau_1 \vdash M \circ x \Longleftrightarrow N \circ x : \tau_2}}{\Delta \vdash M \Longleftrightarrow N : \tau_1 \overset{\div}{\to} \tau_2}\,\mathsf{ae\_ext}$$

Since A^- is $\tau_1 \overset{\div}{\to} \tau_2$, we know A is of the form $\Pi x \div A_1.A_2$. Therefore $\Gamma \vdash M, N : \Pi x \div A_1.A_2$. By inversion, $\Gamma \vdash A_1 : \mathsf{type}$, so using weakening we can derive

$$\dfrac{\Gamma, x \div A_1 \vdash M : \Pi x \div A_1.A_2 \quad \dfrac{\dfrac{}{\Gamma^{\oplus}, x : A_1 \vdash x : A_1}\,\mathsf{ot\_var} \qquad \Gamma \vdash A_1 : \mathsf{type}}{\Gamma \vdash x \div A_1}\,\mathsf{pot}}{\Gamma, x \div A_1 \vdash M \circ x : A_2}\,\mathsf{ot\_app}$$

and similarly $\Gamma, x \div A_1 \vdash N \circ x : A_2$. Therefore we can apply the induction hypothesis to $\mathcal{D}_1$ and obtain $\Gamma, x \div A_1 \vdash M \circ x = N \circ x : A_2$. By oe_ext, we conclude $\Gamma \vdash M = N : A$.

■

5.5 Decidability

Algorithmic equality is a decision procedure on *normalizing* terms, that is, those terms which are algorithmically equal to something.

Lemma 5.17 (Decidiability of Equality for Normalizing Terms)

1. *If $\Delta \vdash M_0 \Longleftrightarrow M_1 : \tau$ and $\Delta \vdash N_0 \Longleftrightarrow N_1 : \tau$, then it is decidable whether $\Delta \vdash M_0 \Longleftrightarrow N_0 : \tau$.*

2. *If $\Delta \vdash M_0 \longleftrightarrow M_1 : \tau_1$ and $\Delta \vdash N_0 \longleftrightarrow N_1 : \tau_2$, then it is decidable whether there exists τ_3 such that $\Delta \vdash M_0 \longleftrightarrow N_0 : \tau_3$.*

3. *If* $\Delta \vdash A_0 \Longleftrightarrow A_1 : \kappa$ *and* $\Delta \vdash B_0 \Longleftrightarrow B_1 : \kappa$*, then it is decidable whether* $\Delta \vdash A_0 \Longleftrightarrow B_0 : \kappa$*.*

4. *If* $\Delta \vdash A_0 \longleftrightarrow A_1 : \kappa_1$ *and* $\Delta \vdash B_0 \longleftrightarrow B_1 : \kappa_2$*, then it is decidable whether there exists* κ_3 *such that* $\Delta \vdash A_0 \longleftrightarrow B_0 : \kappa_3$*.*

5. *If* $\Delta \vdash K_0 \Longleftrightarrow K_1 : \mathsf{kind}^-$ *and* $\Delta \vdash L_0 \Longleftrightarrow L_1 : \mathsf{kind}^-$*, then it is decidable whether* $\Delta \vdash K_0 \Longleftrightarrow L_0 : \mathsf{kind}^-$*.*

Proof By induction on the structure of the given derivations. The determinacy lemma is used in several places, as well as the fact that symmetry and transitivity of algorithmic equality imply that showing $\Delta \vdash M_0 \Longleftrightarrow N_0 : \tau$ is equivalent to showing $\Delta \vdash M_i \Longleftrightarrow N_j : \tau$ for any particular i, j. ■

Theorem 5.18 (Decidability of Algorithmic Equality)

1. *If* $\Gamma \vdash M, N : A$ *then it is decidable whether* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$*.*

2. *If* $\Gamma \vdash A, B : K$ *then it is decidable whether* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$*.*

3. *If* $\Gamma \vdash K, L : \mathsf{kind}$ *then it is decidable whether* $\Gamma^- \vdash K \Longleftrightarrow L : \mathsf{kind}^-$*.*

Proof By reflexivity of definitional equality and completeness of algorithmic equality, M, N (resp. A, B, K, L) are normalizing. Therefore it is decidable if they are algorithmically equal. ■

Corollary 5.19 (Decidability of Definitional Equality)

1. *If* $\Gamma \vdash M, N : A$ *then it is decidable whether* $\Gamma \vdash M = N : A$*.*

2. *If* $\Gamma \vdash A, B : K$ *then it is decidable whether* $\Gamma \vdash A = B : K$*.*

3. *If* $\Gamma \vdash K, L : \mathsf{kind}$ *then it is decidable whether* $\Gamma \vdash K = L : \mathsf{kind}$*.*

Proof By soundness and completeness of algorithmic equality. ■

This result allows us to define and prove correct an algorithm for type-checking as follows:

Objects

$$\frac{c : A \in \Sigma}{\Gamma \vdash c \Rightarrow A}\ \mathsf{aot\_const}$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x \Rightarrow A}\ \mathsf{aot\_var}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{type} \qquad \Gamma, x \star A \vdash M \Rightarrow B}{\Gamma \vdash \lambda x{\star}A.M \Rightarrow \Pi x{\star}A.B}\ \mathsf{aot\_lam}$$

$$\frac{\Gamma \vdash M_1 \Rightarrow \Pi x{:}A'.B \qquad \Gamma \vdash M_2 \Rightarrow A \qquad \Gamma^- \vdash A \Longleftrightarrow A' : \mathsf{type}^-}{\Gamma \vdash M_1\ M_2 \Rightarrow [M_2/x]B}\ \mathsf{aot\_app}$$

$$\frac{\Gamma \vdash M_1 \Rightarrow \Pi x{\div}A'.B \qquad \Gamma^{\oplus} \vdash M_2 \Rightarrow A \qquad \Gamma^- \vdash A \Longleftrightarrow A' : \mathsf{type}^-}{\Gamma \vdash M_1 \circ M_2 \Rightarrow [M_2/x]B}\ \mathsf{aot\_iapp}$$

Families

$$\frac{a : K \in \Sigma}{\Gamma \vdash a \Rightarrow K}\ \mathsf{afv\_const}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{type} \qquad \Gamma, x \star A \vdash B \Rightarrow \mathsf{type}}{\Gamma \vdash \Pi x{\star}A.B \Rightarrow \mathsf{type}}\ \mathsf{afv\_pi}$$

$$\frac{\Gamma \vdash A_1 \Rightarrow \Pi x{:}A_2'.K \qquad \Gamma \vdash M \Rightarrow A_2 \qquad \Gamma^- \vdash A_2 \Longleftrightarrow A_2' : \mathsf{type}^-}{\Gamma \vdash A_1\ M \Rightarrow [M/x]K}\ \mathsf{afv\_app}$$

$$\frac{\Gamma \vdash A_1 \Rightarrow \Pi x{\div}A_2'.K \qquad \Gamma^{\oplus} \vdash M \Rightarrow A_2 \qquad \Gamma^- \vdash A_2 \Longleftrightarrow A_2' : \mathsf{type}^-}{\Gamma \vdash A_1 \circ M \Rightarrow [M/x]K}\ \mathsf{afv\_iapp}$$

Kinds

$$\frac{}{\Gamma \vdash \mathsf{type} \Rightarrow \mathsf{kind}}\ \mathsf{akv\_type}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{type} \qquad \Gamma, x \star A \vdash K \Rightarrow \mathsf{kind}}{\Gamma \vdash \Pi x{\star}A.K \Rightarrow \mathsf{kind}}\ \mathsf{akv\_pi}$$

Lemma 5.20 (Algorithmic Equality Lifting)

1. *If* $\Gamma \vdash M' : A$ *and* $(\Gamma^{\oplus})^- \vdash M \Longleftrightarrow M' : A^-$ *then* $\Gamma^- \vdash M \Longleftrightarrow M' : A^-$.
2. *If* $\Gamma \vdash M' : A$ *and* $(\Gamma^{\oplus})^- \vdash M \longleftrightarrow M' : A^-$ *then* $\Gamma^- \vdash M \longleftrightarrow M' : A^-$.

Proof By induction. The case of weak head-reducing the right side for part 1 holds by subject reduction, and the variable case for part 2 follows because by inversion $\Gamma \vdash x : A$ implies that $x : A' \in \Gamma$, for A' such that $\Gamma \vdash A = A' : \mathsf{type}$ so $\Gamma^- \vdash x \longleftrightarrow x : A^-$ by erasure preservation. ■

Lemma 5.21 (Correctness of Algorithmic Type-Checking)

1. *If* $\Gamma \vdash M \Rightarrow A$, *then* $\Gamma \vdash M : A$.
2. *If* $\Gamma \vdash M : A$, *then there is an* A' *such that* $\Gamma \vdash M \Rightarrow A'$ *and* $\Gamma \vdash A = A' : \mathsf{kind}$.
3. *If* $\Gamma \vdash A \Rightarrow K$, *then* $\Gamma \vdash A : K$.
4. *If* $\Gamma \vdash A : K$, *then there is an* K' *such that* $\Gamma \vdash A \Rightarrow K'$ *and* $\Gamma \vdash K = K' : \mathsf{type}$.

5. *If* $\Gamma \vdash K \Rightarrow \mathsf{kind}$*, then* $\Gamma \vdash K : \mathsf{kind}$.

6. *If* $\Gamma \vdash K : \mathsf{kind}$*, then* $\Gamma \vdash K \Rightarrow \mathsf{kind}$.

Proof By induction using established lemmas. We show the irrelevant application cases at the object level.

Case: Part 1, $\mathcal{D} =$

$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash M_1 \Rightarrow \Pi x{\div}A'.B} \quad \overset{\mathcal{D}_2}{\Gamma^{\oplus} \vdash M_2 \Rightarrow A} \quad \overset{\mathcal{D}_3}{\Gamma \vdash A \Longleftrightarrow A' : \mathsf{type}^{-}}}{\Gamma \vdash M_1 \circ M_2 \Rightarrow [M_2/x]B}\,\mathsf{aot\_iapp}$$

By the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2$ we know

$$\Gamma \vdash M_1 : \Pi x{\div}A'.B$$

$$\Gamma^{\oplus} \vdash M_2 : A$$

By inversion of products, $\Gamma \vdash A' : \mathsf{type}$ from which follows $\Gamma^{\oplus} \vdash A' : \mathsf{type}$ by Corollary 4.6. We also have $\Gamma^{\oplus} \vdash A : \mathsf{type}$ by validity. Now $\Gamma^{-} \vdash A \Longleftrightarrow A' : \mathsf{type}^{-}$ can be weakened to $(\Gamma^{\oplus})^{-} \vdash A \Longleftrightarrow A' : \mathsf{type}^{-}$ and soundness of algorithmic typing yields $\Gamma^{\oplus} \vdash A = A' : \mathsf{type}$ and we can derive

$$\frac{\Gamma \vdash M_1 : \Pi x{\div}A'.B \qquad \dfrac{\Gamma \vdash A' : \mathsf{type} \qquad \dfrac{\Gamma^{\oplus} \vdash M_2 : A \qquad \Gamma^{\oplus} \vdash A = A' : \mathsf{type}}{\Gamma^{\oplus} \vdash M_2 : A'}\,\mathsf{ot\_conv}}{\Gamma \vdash M_2 \div A'}\,\mathsf{pot}}{\Gamma \vdash M_1 \circ M_2 : [M_2/x]B}\,\mathsf{ot\_app}$$

Case: Part 2, $\mathcal{D} =$

$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash M_1 : \Pi x{\div}A.B} \quad \overset{\mathcal{D}_2}{\Gamma \vdash M_2 \div A}}{\Gamma \vdash M_1 \circ M_2 : [M_2/x]B}\,\mathsf{ot\_app}$$

By inversion,

$$\mathcal{D}_2 = \frac{\overset{\mathcal{D}_2'}{\Gamma^{\oplus} \vdash M_2 : A} \quad \overset{\mathcal{D}_2''}{\Gamma \vdash A : \mathsf{type}}}{\Gamma \vdash M_2 \div A}\,\mathsf{pot}$$

By the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2'$, we can find C, A' such that $\Gamma \vdash M_1 \Rightarrow C$, $\Gamma^{\oplus} \vdash M_2 \Rightarrow A'$, $\Gamma \vdash C = \Pi x{\div}A.B : \mathsf{type}$, and $\Gamma^{\oplus} \vdash A = A' : \mathsf{type}$. By Lemma 4.16, C must be of the form $\Pi x{\div}A''.B''$ such that $\Gamma \vdash A = A'' : \mathsf{type}$ and $\Gamma, x{\div}A'' \vdash B = B'' : \mathsf{type}$. By Corollary 4.6 and symmetry and transitivity of definitional equality, we get $\Gamma^{\oplus} \vdash A' = A'' : \mathsf{type}$. Completeness of algorithmic equality implies $(\Gamma^{\oplus})^{-} \vdash A' \Longleftrightarrow$

$A'' : \mathsf{type}^-$. By validity applied to $\Gamma \vdash A = A'' : \mathsf{type}$, we have $\Gamma \vdash A' : \mathsf{type}$, so by Lemma 5.20 we have $\Gamma^- \vdash A' = A'' : \mathsf{type}^-$. Therefore we can derive

$$\frac{\Gamma \vdash M_1 \Rightarrow \Pi x{\div}A''.B'' \quad \Gamma^{\oplus} \vdash M_2 \Rightarrow A' \quad \Gamma^- \vdash A' \Longleftrightarrow A'' : \mathsf{type}^-}{\Gamma \vdash M_1 \circ M_2 \Rightarrow [M_2/x]B''}\ \mathsf{aot\_iapp}$$

Now we know from $\Gamma \vdash A = A'' : \mathsf{type}$ that $\vdash \Gamma, x \div A'' = \Gamma, x \div A : \mathsf{ctx}$ by symmetry and $\mathsf{ce\_var}$, so Lemma 4.8 allows us to infer that $\Gamma, x \div A \vdash B = B'' : \mathsf{type}$. By substitution of $\Gamma \vdash M_2 \div A$, we conclude $\Gamma \vdash [M_2/x]B = [M_2/x]B'' : \mathsf{type}$.

■

Theorem 5.22 (Decidability of Type-Checking)

1. *It is decidable whether* $\vdash \Gamma : \mathsf{ctx}$.
2. *Suppose* $\vdash \Gamma : \mathsf{ctx}$ *and* $\Gamma \vdash A : \mathsf{type}$. *Given* M, *it is decidable whether* $\Gamma \vdash M : A$.
3. *Suppose* $\vdash \Gamma : \mathsf{ctx}$ *and* $\Gamma \vdash K : \mathsf{kind}$. *Given* A, *it is decidable whether* $\Gamma \vdash A : K$.
4. *Suppose* $\vdash \Gamma : \mathsf{ctx}$. *Given* K, *it is decidable whether* $\Gamma \vdash K : \mathsf{kind}$.

Proof Since algorithmic equality is decidable and the algorithmic typing judgment is syntax-directed, it is decidable whether there exists A' such that $\Gamma \vdash M \Rightarrow A'$, and this A' is unique if it exists. By correctness of algorithmic typing, we have $\Gamma \vdash A' : \mathsf{type}$, and so $\Gamma \vdash M : A$ iff $\Gamma \vdash A' = A : \mathsf{type}$, and this equality is decidable by Corollary 5.19. The argument is similar for the other cases. ■

The algorithms for typing and equality admit direct proofs of strengthening. Their correctness allows us to lift these properties to the original typing and equality judgments.

Lemma 5.23 (Strengthening of Algorithmic Equality)

1. *Suppose* $M \stackrel{whr}{\longrightarrow} M'$. *If* x *appears free in* M', x *appears free in* M.
2. *If* $\Delta, x \star \tau', \Delta' \vdash M \Longleftrightarrow N : \tau$ *and* x *does not appear free in* M, N, *then* $\Delta, \Delta' \vdash M \Longleftrightarrow N : \tau$.
3. *If* $\Delta, x \star \tau', \Delta' \vdash M \longleftrightarrow N : \tau$ *and* x *does not appear free in* M, N, *then* $\Delta, \Delta' \vdash M \longleftrightarrow N : \tau$.

Proof Straightforward induction. ■

Lemma 5.24 (Strengthening of Algorithmic Typing)

1. *If* $\Gamma, x \star B, \Gamma' \vdash M \Rightarrow A$ *and* x *does not appear free in* Γ', M, *then* $\Gamma, \Gamma' \vdash M \Rightarrow A$ *and* x *does not appear free in* A.

2. *If* $\Gamma, x \star B, \Gamma' \vdash A \Rightarrow K$ *and* x *does not appear free in* Γ', A*, then* $\Gamma, \Gamma' \vdash A \Rightarrow K$ *and* x *does not appear free in* K*.*

3. *If* $\Gamma, x \star B, \Gamma' \vdash K \Rightarrow \textsf{kind}$ *and* x *does not appear free in* Γ', K*, then* $\Gamma, \Gamma' \vdash K \Rightarrow \textsf{kind}$*.*

Proof Straightforward induction, using Lemma 5.23 at the application case. ■

Corollary 5.25 *If* $\Gamma, x \star A, \Gamma' \vdash J$ *and* x *does not appear free in* Γ', J*, then* $\Gamma, \Gamma' \vdash J$*.*

Proof By induction on the structure of the derivation, using Lemmas 5.23 and 5.24, and the correctness of algorithmic equality and typing. ■

6 Strict Definitions

Abbreviations and definitions are important tools in informal as well as formal mathematics for making large proofs understandable. However, it can be inefficient for proof-checking software to expand the meaning of every definition it encounters. One technique for reducing the number of definition expansions required applies to defined functions which are *injective* in the usual mathematical sense. If a defined function can be shown to be injective — if it has a different output for every input — then we need not expand its definition if we want to know if there are any solutions to an equation of the form

$$d\ M_1 \cdots M_n = d\ M'_1 \cdots M'_n$$

If d is known to be injective, then this equation holds only if $M_i = M'_i$ for all $1 \leq i \leq n$. This optimization can make unification of terms using defined functions much more efficient.

Unfortunately, it is undecidable whether a given defined function is injective, but there is a decidable property, *strictness*, introduced by Pfenning and Schürmann [PS98], that implies injectivity. Experience suggests that in practice many definitions are already strict, and in fact the present work has the potential to make an even larger class of definitions easy to modify cleanly so as to be strict. Appel's Twelf Tutorial [App00] describes several ad-hoc techniques for 'cutting in' trivial uses of arguments to make them have strict occurrences. His `lemma57` (in a standard encoding of natural deduction) can be changed from

```
lemma57: pf A -> pf B -> pf (A imp B) =
 [p1: pf A]
 [p2: pf B]
 imp_i [p3: pf A]
 p2.
```

to

```
lemma57: pf A -> pf B -> pf (A imp B) =
 [p1: pf A]
 [p2: pf B]
```

```
 cut p1 [p5: pf A]
 imp_i [p3: pf A]
 p2.
```

for a defined `cut : pf A -> pf (A -> B) -> pf B` to 'cut in' a fake use of the argument of type `pf A`. With irrelevance, however we can simply mark that argument as unused by writing

```
lemma57: pf A -i> pf B -> pf (A imp B) =
 [p1 / pf A]
 [p2: pf B]
 imp_i [p3: pf A]
 p2.
```

(where `-i>` and `/` are concrete syntax for $\stackrel{\div}{\to}$ and $\div$, respectively) and our prototype implementation verifies this as strict. It is injective because If $\texttt{lemma57} \circ M_1 \ M_2 \ M_3 = \texttt{lemma57} \circ M_1' \ M_2' \ M_3'$, then we are guaranteed $M_1 = M_1' \div \texttt{pf A}$ even though the first argument to `lemma57` is never used, exactly because equality at irrelevant types is trivial.

Intuitively, a defined function is strict if it uses all its arguments 'substantially'. If every argument has a substantial use, then changing any argument must change the result, and so the function is injective. Since the concepts of strictness and irrelevance are both in some sense about whether arguments to functions matter (even though they make opposite claims — a strict function definitely *does* use its argument and an irrelevant function definitely doesn't) we would expect they would nontrivially interact. We present a generalization of the definition of strictness that applies to terms in the above type theory that retains the important properties of that in [PS98].

6.1 Spines

An important element in the definition of strictness is the use of *spines* [CP97]. The spine calculus can itself be used to improve implementation efficiency of algorithms which traverse terms 'from the head down'. Instead of a nested application $((\cdots((c\ (M_1\ (M_2\ \cdots))))\cdots))$ which would need to be 'unwound' all the way down to the head, the spine calculus represents it as $c \cdot (M_1; M_2; \cdots; \mathsf{nil})$, a head followed by a list of arguments. Thus the head of an application can be found in constant time, and unification can frequently discover, for instance, constant clashes more quickly. The chief role of spines in strictness is in the fact that an argument to a defined function is said to *have a strict occurrence* if it appears in a rigid position (under application of a constant) applied to a spine of distinct bound (i.e. other than the arguments to the defined function) variables.

For the sake of avoiding introducing another entire variation of LF, we depart from the approach of [PS98], which has spines as first-class citizens in the type theory (and in fact every term involves a spine), and treat spines as a purely notational tool. We define spines syntactically by

$$\text{Spines} \quad S \quad ::= \mathsf{nil} \mid (M; S) \mid (M \div S)$$

and define *head-spine application* recursively as follows:

$$\begin{array}{rcl} M' \cdot \mathsf{nil} & = & M' \\ M' \cdot (M; S) & = & (M'\ M) \cdot S \\ M' \cdot (M \div S) & = & (M' \circ M) \cdot S \end{array}$$

For our purposes, a spine is a list of terms, each followed by a marker ; or $\div$ depending on whether that term is to be under a relevant or irrelevant application, respectively. To emphasize the syntactic intent of these definitions, when we write a head M applied to a spine, say, $(M_1; M_2 \div M_3 \div \mathsf{nil})$ as $M \cdot (M_1; M_2 \div M_3 \div \mathsf{nil})$ we simply mean the actual term $M\ M_1 \circ M_2 \circ M_3$. In no sense does the former reduce to the latter in the theory, nor are they merely equal at some type — the former is simply an abbreviation for the latter.

We will find it useful to define several auxiliary judgments on spines. The *spine typing* judgment $\Gamma \vdash S : A > C$ means that if we had a head M of type A, then $M \cdot S$ would have type C. Equality (resp. algorithmic equality) of spines at $A > C$ is a straightforward requirement that every two corresponding terms in a pair of spines are equal (resp. algorithmically equal) at the appropriate type.

6.1.1 Spine Typing

$$\frac{}{\Gamma \vdash \mathsf{nil} : A > A}\ \mathsf{st\_nil}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash S : [M/x]B > C}{\Gamma \vdash (M; S) : \Pi x{:}A.B > C}\ \mathsf{st\_cons}$$

$$\frac{\Gamma \vdash M \div A \qquad \Gamma \vdash S : [M/x]B > C}{\Gamma \vdash (M \div S) : \Pi x{\div}A.B > C}\ \mathsf{st\_icons}$$

$$\frac{\Gamma \vdash S : A > B \qquad \Gamma \vdash B = B' : \mathsf{type}}{\Gamma \vdash S : A > B'}\ \mathsf{st\_conv}$$

6.1.2 Spine Equality

$$\frac{}{\Gamma \vdash \mathsf{nil} = \mathsf{nil} : A > A}\ \mathsf{se\_nil}$$

$$\frac{\Gamma \vdash M_1 = M_2 : A \qquad \Gamma \vdash S_1 = S_2 : [M_1/x]B > C}{\Gamma \vdash (M_1; S_1) = (M_2; S_2) : \Pi x{:}A.B > C}\ \mathsf{se\_cons}$$

$$\frac{\Gamma \vdash M_1 = M_2 \div A \qquad \Gamma \vdash S_1 = S_2 : [M_1/x]B > C}{\Gamma \vdash (M_1 \div S_1) = (M_2 \div S_2) : \Pi x{\div}A.B > C}\ \mathsf{se\_icons}$$

6.1.3 Algorithmic Spine Equality

$$\frac{}{\Delta \vdash \mathsf{nil} \Longleftrightarrow \mathsf{nil} : \tau > \tau}\ \mathsf{ase\_nil}$$

$$\frac{\Delta \vdash M_1 \Longleftrightarrow M_2 : \tau \qquad \Delta \vdash S_1 = S_2 : \sigma > \rho}{\Delta \vdash (M_1; S_1) \Longleftrightarrow (M_2; S_2) : \tau \rightarrow \sigma > \rho}\ \mathsf{ase\_cons}$$

$$\frac{\Delta \vdash S_1 \Longleftrightarrow S_2 : \sigma > \rho}{\Delta \vdash (M_1 \div S_1) \Longleftrightarrow (M_2 \div S_2) : \tau \stackrel{\div}{\rightarrow} \sigma > \rho}\ \mathsf{ase\_icons}$$

6.2 Strictness

The definition of strictness consists of six judgments:

Pattern Spine	$\Delta \vdash S$ pat	In Δ, S is a pattern spine.
Local Strict Occurrences	$\Gamma; \Delta \Vdash_x M$	x occurs locally strict in M
	$\Gamma; \Delta \Vdash_x A$	x occurs locally strict in A
	$\Gamma; \Delta \Vdash_x S$	x occurs locally strict in S
Strict Occurrences	$\Gamma \Vdash_x M$	x occurs strict in M
Strictness	$\Gamma \Vdash M$	M is strict

6.2.1 Pattern Spines

In [PS98], a spine S is *pattern* in Δ, written $\Delta \vdash S$ pat, if it consists of distinct bound variables from Δ. We can generalize this by allowing any term at all at an irrelevant position in a spine, and requiring that all terms at relevant positions are distinct bound variables. We in fact define a judgment $\Delta \vdash S \sim S'$ pat which intends to mean that S and S' are *similar* pattern spines in the sense that they are syntactically identical except possibly differ at irrelevant positions, and write $\Delta \vdash S$ pat as an abbreviation for $\Delta \vdash S \sim S$ pat.

$$\frac{}{\Delta \vdash \mathsf{nil} \sim \mathsf{nil}\ \mathsf{pat}}\ \mathsf{pat\_nil}$$

$$\frac{\Delta', \Delta'' \vdash S' \sim S''\ \mathsf{pat} \qquad x \notin \Delta', \Delta''}{\Delta', x : \tau, \Delta'' \vdash (x; S') \sim (x; S'')\ \mathsf{pat}}\ \mathsf{pat\_cons}$$

$$\frac{\Delta \vdash S' \sim S''\ \mathsf{pat}}{\Delta \vdash (M \div S') \sim (M' \div S'')\ \mathsf{pat}}\ \mathsf{pat\_icons}$$

This definition forms the base case of the definition of strictness. The importance of pattern spines is that application of a pattern spine to a term is injective, in the sense that $M_1 \cdot S = M_2 \cdot S$ implies that $M_1 = M_2$ for pattern S. Carefully proving this fact requires some auxiliary definitions and several lemmas.

6.2.2 Approximate Typing

We define an approximate typing judgment $\Delta \vdash M : \tau$ in the evident way.

$$\frac{c : A \in \Sigma}{\Delta \vdash c : A^-}\ \textsf{appot\_const}$$

$$\frac{x : \tau \in \Delta}{\Delta \vdash x : \tau}\ \textsf{appot\_var}$$

$$\frac{\Delta, x \star A^- \vdash M : \sigma}{\Delta \vdash \lambda x{\star}A.M : A^- \xrightarrow{\star} \sigma}\ \textsf{appot\_lam}$$

$$\frac{\Delta \vdash M_1 : \tau \to \sigma \qquad \Delta \vdash M_2 : \tau}{\Delta \vdash M_1\ M_2 : \sigma}\ \textsf{appot\_app}$$

$$\frac{\Delta \vdash M_1 : \tau \xrightarrow{\div} \sigma}{\Delta \vdash M_1 \circ M_2 : \sigma}\ \textsf{appot\_iapp}$$

This judgment satisfies some typical properties. All proofs are by straightforward induction.

Lemma 6.1 (Approximate Type Substitution) *If $\Delta, x \star \sigma \vdash M : \tau$ and $\Delta \vdash N \star \sigma$, then $\Delta \vdash [N/x]M : \tau$.*

Lemma 6.2 (Approximate Type Subject Reduction) *If $M \xrightarrow{whr} M'$ and $\Delta \vdash M : \tau$, then $\Delta \vdash M' : \tau$.*

Lemma 6.3 (Approximate Validity)

1. *If $\Delta \vdash M_1 \Longleftrightarrow M_2 : \tau$, then $\Delta \vdash M_1 : \tau$ and $\Delta \vdash M_2 : \tau$.*
2. *If $\Delta \vdash M_1 \longleftrightarrow M_2 : \tau$, then $\Delta \vdash M_1 : \tau$ and $\Delta \vdash M_2 : \tau$.*

6.2.3 Similarity

We also define a relation $\Delta \vdash M \sim M'$ (pronounced M similar to M') on terms which requires M and M' to be syntactically identical and use only relevant variables from Δ, except that both requirements are relaxed under irrelevant application.

$$\frac{\Delta \vdash M \sim M' \qquad \Delta \vdash N \sim N'}{\Delta \vdash M\ N \sim M'\ N'}\ \textsf{sim\_app}$$

$$\frac{\Delta \vdash M \sim M'}{\Delta \vdash M \circ N \sim M' \circ N'}\ \textsf{sim\_iapp}$$

$$\frac{\Delta, x \star A^- \vdash M \sim M'}{\Delta \vdash \lambda x{\star}A.M \sim \lambda x{\star}A.M'}\ \textsf{sim\_lam}$$

$$\frac{c : A \in \Sigma}{\Delta \vdash c \sim c}\,\mathsf{sim\_const}$$

$$\frac{x : \tau \in \Delta}{\Delta \vdash x \sim x}\,\mathsf{sim\_var}$$

This judgment satisfies a functionality property for substitution of similar terms into a relevant variable, and any two terms can be substituted for an irrelevant variable and preserve similarity.

Lemma 6.4 (Similarity Functionality)

1. *If* $\Delta \vdash N \sim N'$ *and* $\Delta, x : \tau, \Delta' \vdash M \sim M'$, *then* $\Delta, \Delta' \vdash [N/x]M \sim [N'/x]M'$.
2. *If* $\Delta, x \div \tau, \Delta' \vdash M \sim M'$, *then* $\Delta, \Delta' \vdash [N/x]M \sim [N'/x]M'$.

Proof By induction on $\mathcal{D} :: \Delta, x \star \tau \vdash M \sim M'$, using inversion on the approximate typing rules to be able to apply the induction hypothesis, and using $\Delta \vdash N \sim N'$ when $\mathcal{D}$ is $\mathsf{sim\_var}$.
■

The key property of similarity is that weak head-reduction and algorithmic and structural equality can be 'pushed' across two pairs of similar terms. For instance, the case for algorithmic equality, in a diagram, is

$$\text{If}\quad \begin{tikzcd} M_1 \arrow[r, Leftrightarrow] \arrow[d, "\sim"'] & M_2 \arrow[d, "\sim"] \\ M_1' & M_2' \end{tikzcd}, \quad \text{then} \quad \begin{tikzcd} M_1 \arrow[r, Leftrightarrow] \arrow[d, "\sim"'] & M_2 \arrow[d, "\sim"] \\ M_1' \arrow[r, Leftrightarrow] & M_2' \end{tikzcd}$$

Lemma 6.5 (Equality Preservation)

1. *If* $\Delta \vdash M_1 \sim M_1'$ *and* $M_1 \xrightarrow{whr} M_2$, *then there exists* M_2' *such that* $\Delta \vdash M_2 \sim M_2'$ *and* $M_1' \xrightarrow{whr} M_2'$.
2. *If* $\Delta \vdash M_1 \sim M_1'$ *and* $M_1' \xrightarrow{whr} M_2'$, *then there exists* M_2 *such that* $\Delta \vdash M_2 \sim M_2'$ *and* $M_1 \xrightarrow{whr} M_2$.
3. *If* $\Delta \vdash M_i \sim M_i'$ *and* $\Delta \vdash M_1 \Longleftrightarrow M_2 : \tau$, *then* $\Delta \vdash M_1' \Longleftrightarrow M_2' : \tau$.
4. *If* $\Delta \vdash M_i \sim M_i'$ *and* $\Delta \vdash M_1' \Longleftrightarrow M_2' : \tau$, *then* $\Delta \vdash M_1 \Longleftrightarrow M_2 : \tau$.
5. *If* $\Delta \vdash M_i \sim M_i'$ *and* $\Delta \vdash M_1 \longleftrightarrow M_2 : \tau$, *then* $\Delta \vdash M_1' \longleftrightarrow M_2' : \tau$.
6. *If* $\Delta \vdash M_i \sim M_i'$ *and* $\Delta \vdash M_1' \longleftrightarrow M_2' : \tau$, *then* $\Delta \vdash M_1 \longleftrightarrow M_2 : \tau$.

Proof By induction on $\mathcal{D} :: M \xrightarrow{whr} M'$ (parts 1 and 2) or $\mathcal{D} :: \Delta \vdash M_1 \Longleftrightarrow M_2 : \tau$ (parts 3 and 4) or $\mathcal{D} :: \Delta \vdash M_1 \longleftrightarrow M_2 : \tau$ (parts 5 and 6). We show a few representative cases.

Case: Part 1, $\mathcal{D} =$

$$\frac{}{(\lambda x{:}A_1.M_2)M_3 \stackrel{whr}{\longrightarrow} [M_3/x]M_2}\ \mathsf{whr\_beta}$$

By inversion on the derivation of $\Delta \vdash (\lambda x{:}A_1.M_2)M_3 \sim M_1'$ we know M_1' is of the form $(\lambda x{:}A_1.M_2')M_3'$ such that $\Delta, x : A_1^- \vdash M_2 \sim M_2'$ and $\Delta \vdash M_3 \sim M_3'$. By whr_beta we know $M_1' \stackrel{whr}{\longrightarrow} [M_3'/x]M_2'$. By Lemma 6.4 part 1 we conclude $[M_3/x]M_2 \sim [M_3'/x]M_2'$.

Case: Part 1, $\mathcal{D} =$

$$\frac{}{(\lambda x{\div}A_1.M_2) \circ M_3 \stackrel{whr}{\longrightarrow} [M_3/x]M_2}\ \mathsf{whr\_beta}$$

By inversion on the derivation of $\Delta \vdash (\lambda x{\div}A_1.M_2) \circ M_3 \sim M_1'$ we know M_1' is of the form $(\lambda x{\div}A_1.M_2') \circ M_3'$ such that $\Delta, x \div A_1^- \vdash M_2 \sim M_2'$. By whr_beta we know $M_1' \stackrel{whr}{\longrightarrow} [M_3'/x]M_2'$. By Lemma 6.4 part 2 we conclude $[M_3/x]M_2 \sim [M_3'/x]M_2'$.

Case: Part 3, $\mathcal{D} =$

$$\frac{M_1 \stackrel{whr}{\longrightarrow} N \qquad \Delta \vdash N \Longleftrightarrow M_2 : \alpha}{\Delta \vdash M_1 \Longleftrightarrow M_2 : \alpha}\ \mathsf{ae\_whrl}$$

By the induction hypothesis (part 1) there is N' such that $M_1' \stackrel{whr}{\longrightarrow} N'$ and $\Delta \vdash N \sim N'$. By the induction hypothesis (part 3) on the fact that $\Delta \vdash N \Longleftrightarrow M_2 : \alpha$, $\Delta \vdash N \sim N'$ and $\Delta \vdash M_2 \sim M_2'$, we know that $\Delta \vdash N' \Longleftrightarrow M_2' : \alpha$. By ae_whrl we can conclude $\Delta \vdash M_1' \Longleftrightarrow M_2' : \alpha$.

Case: Part 3, $\mathcal{D} =$

$$\frac{\Delta, x : \tau_1 \vdash M_1\ x \Longleftrightarrow M_2\ x : \tau_2}{\Delta \vdash M_1 \Longleftrightarrow M_2 : \tau_1 \to \tau_2}\ \mathsf{ae\_ext}$$

We can easily derive $\Delta, x : \tau_1 \vdash M_i\ x \sim M_i'\ x$ from assumptions, so the induction hypothesis gives $\Delta, x : \tau_1 \vdash M_1'\ x \Longleftrightarrow M_2'\ x : \tau_2$. By ae_ext, we conclude $\Delta \vdash M_1' \Longleftrightarrow M_2' : \tau_1 \to \tau_2$.

Case: Part 5, $\mathcal{D} =$

$$\frac{\begin{array}{c}\mathcal{D}_1\\ \Delta \vdash M_1 \longleftrightarrow M_2 : \tau_2 \stackrel{\div}{\to} \tau_1\end{array}}{\Delta \vdash M_1 \circ N_1 \longleftrightarrow M_2 \circ N_2 : \tau_1}\ \mathsf{se\_iapp}$$

Suppose $\Delta \vdash M_1 \circ N_1 \sim M_1' \circ N_1'$ and $\Delta \vdash M_2 \circ N_2 \sim M_2' \circ N_2'$. By inversion on the rules defining $\sim$, we have $\Delta \vdash M_1 \sim M_1'$ and $\Delta \vdash M_2 \sim M_2'$. By the induction hypothesis on $\mathcal{D}_1$, we see $\Delta \vdash M_1' \longleftrightarrow M_2' : \tau_2 \stackrel{\div}{\to} \tau_1$. Therefore by se_iapp we conclude $\Delta \vdash M_1' \circ N_1' \longleftrightarrow M_2' \circ N_2' : \tau_1$.

■

We will also need that approximately well-typed terms are similar to themselves, i.e. don't use irrelevant variables except under irrelevant application.

Lemma 6.6 (Similarity Reflexivity) *If $\Delta \vdash M : \tau$, then $\Delta \vdash M \sim M$.*

Proof Straightforward induction. ■

Finally, we require several inversion principles concerning spines.

Lemma 6.7 (Spine Inversion)

1. *If $\Gamma \vdash (M; S) : A' > C$, then there exist A, B such that A is of the form $\Pi x{:}A.B$, $\Gamma \vdash M : A$, and $\Gamma \vdash S : [M/x]B > C$.*
2. *If $\Gamma \vdash (M \div S) : A' > C$, then there exist A, B such that A is of the form $\Pi x{\div}A.B$, $\Gamma \vdash M \div A$, and $\Gamma \vdash S : [M/x]B > C$.*
3. *If $\Gamma \vdash S : \Pi x{:}A.B > C$ and C is a base type, then S is of the form $(M; S')$ such that $\Gamma \vdash M : A$ and $\Gamma \vdash S' : [M/x]B > C$.*
4. *If $\Gamma \vdash S : \Pi x{\div}A.B > C$ and C is a base type, then S is of the form $(M \div S')$ such that $\Gamma \vdash M \div A$ and $\Gamma \vdash S' : [M/x]B > C$.*
5. *If $\Gamma \vdash \mathsf{nil} : A > B$, then $\Gamma \vdash A = B : \mathsf{type}$.*

Proof By induction on the spine typing rules. ■

Lemma 6.8 (Spine Head Inversion) *If $\Gamma \vdash S : A > C$ and $\Gamma^- \vdash M \cdot S : C^-$, then $\Gamma^- \vdash M : A^-$.*

Proof By induction on $\mathcal{D} :: \Gamma \vdash S : A > C$. ■

We are now ready to show that application of a pattern spine is injective.

Lemma 6.9 (Pattern Spine Injectivity) *Suppose for $i \in \{1, 2\}$ that $\Psi \vdash S_i : A > C$. If Γ, Δ are disjoint subsets of Ψ (We only use Γ^-, Δ^- so we can use exchange without being concerned with dependencies), and*

1. *$\Delta^- \vdash S_1 \sim S_2$ pat,*
2. *$FV(M_1), FV(M_2) \subseteq \Gamma$,*
3. *$\Gamma^-, \Delta^- \vdash M_1 \cdot S_1 \Longleftrightarrow M_2 \cdot S_2 : C^-$, and*

then $\Gamma^- \vdash M_1 \Longleftrightarrow M_2 : A^-$.

Proof By induction on $\mathcal{D} :: \Delta^- \vdash S$ pat.

Case:

$$\mathcal{D} = \frac{}{\Delta^- \vdash \mathsf{nil} \sim \mathsf{nil}\ \mathsf{pat}}\ \mathsf{pat\_nil}$$

By inversion on the spine typing rules, $\Gamma \vdash A = C : \mathsf{type}$. By erasure preservation, A^- is the same as C^-. By strengthening of algorithmic equality, $\Gamma^- \vdash M_1 \Longleftrightarrow M_2 : A^-$.

Case:

$$\mathcal{D} = \cfrac{\begin{matrix}\mathcal{D}_1 \\ (\Delta')^-, (\Delta'')^- \vdash S_1' \sim S_2' \ \mathsf{pat} \quad x \notin \Delta', \Delta''\end{matrix}}{(\Delta')^-, x : \tau, (\Delta'')^- \vdash (x; S_1') \sim (x; S_2') \ \mathsf{pat}} \ \mathsf{pat\_cons}$$

In this case, S_i is $(x; S_i')$ and Δ is $\Delta', x : A_0, \Delta''$ such that A_0^- is τ. By Lemma 6.7, we know A is of the form $\Pi y{:}A'.B$ such that $\Psi \vdash x : A'$ and $\Psi \vdash S_i' : [x/y]B > C$. By Lemma 4.14 and the fact that $x : A_0 \in \Psi$, we have $\Psi \vdash A' = A_0 : \mathsf{type}$. so by erasure preservation $(A')^-$ and A_0^- are both τ. The definition of head-spine application says

$$\Gamma^-, \Delta^- \vdash (M_1\ x) \cdot S_1' \Longleftrightarrow (M_2\ x) \cdot S_2' : C^-$$

So by the induction hypothesis (with $\Gamma, x : A_0$ and Δ', Δ'') we know $\Gamma^-, x : \tau \vdash M_1\ x \Longleftrightarrow M_2\ x : B^-$. By $\mathsf{ae\_ext}$, we conclude $\Gamma^- \vdash M_1 \Longleftrightarrow M_2 : \tau \to B^-$.

Case:

$$\mathcal{D} = \cfrac{\begin{matrix}\mathcal{D}_1 \\ \Delta^- \vdash S_1' \sim S_2' \ \mathsf{pat}\end{matrix}}{\Delta \vdash (M_1' \div S_1') \sim (M_2' \div S_2') \ \mathsf{pat}} \ \mathsf{pat\_icons}$$

In this case, S_i is $(M_i \div S_i')$. By Lemma 6.7, we know A is of the form $\Pi y{\div}A'.B$ such that $\Psi \vdash S_i' : [M/y]B > C$. The definition of head-spine application says

$$\Gamma^-, \Delta^- \vdash (M_1 \circ M_1') \cdot S' \Longleftrightarrow (M_2 \circ M_2') \cdot S' : C^-$$

Since M_1, M_2 are approximately well-typed by Lemmas 6.3 and 6.8, they are similar to themselves by Lemma 6.6. Therefore for some new variable y we can derive $\Gamma^-, y \div (A')^-, \Delta^- \vdash M_i \circ M_i' \sim M_i \circ y$. Also S' is well-typed, so it is easy to show that $\Gamma^-, y \div (A')^-, \Delta^- \vdash (M_i \circ M_i') \cdot S' \sim (M_i \circ y) \cdot S'$. Lemma 6.5 gives us

$$\Gamma^-, y \div (A')^-, \Delta^- \vdash (M_1 \circ y) \cdot S' \Longleftrightarrow (M_2 \circ y) \cdot S' : C^-$$

So by the induction hypothesis (with $\Gamma, y \div A'$ and Δ) we know $\Gamma^-, y \div (A')^- \vdash M_1 \circ y \Longleftrightarrow M_2 \circ y : B^-$. By $\mathsf{ae\_ext}$, we conclude $\Gamma^- \vdash M_1 \Longleftrightarrow M_2 : (A') \overset{\div}{\to} B^-$.

■

We now define the remaining five judgments. At the highest level, $\Gamma \Vdash M$ asserts that M is a strict defined function, and is therefore injective. The judgment $\Gamma \Vdash_x M$ means that x has a strict occurrence in M, intuitively, x is guaranteed to be used by M. Finally, $\Gamma; \Delta \Vdash_x M$ means that x has a strict occurrence in M assuming that Δ contains bound variables.

6.2.4 Local Strict Occurrences

$$\frac{\Delta^- \vdash S \ \mathsf{pat} \qquad \Delta \vdash S : B > C}{\Gamma; \Delta \Vdash_x x \cdot S}\ \mathsf{ls\_pat}$$

$$\frac{\Gamma; \Delta \Vdash_x M}{\Gamma; \Delta \Vdash_x (M; S)}\ \mathsf{ls\_hd} \qquad \frac{\Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x (M; S)}\ \mathsf{ls\_sp}$$

$$\frac{\Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x (M \div S)}\ \mathsf{ls\_isp}$$

$$\frac{y : A \in \Delta \qquad \Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x y \cdot S}\ \mathsf{ls\_var}$$

$$\frac{\Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x a \cdot S}\ \mathsf{ls\_a} \qquad \frac{\Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x c \cdot S}\ \mathsf{ls\_c}$$

$$\frac{d = M : A \in \Sigma \qquad \cdot \Vdash M \qquad \Gamma; \Delta \Vdash_x S}{\Gamma; \Delta \Vdash_x d \cdot S}\ \mathsf{ls\_d}$$

$$\frac{M \xrightarrow{whr} M' \qquad \Gamma; \Delta \Vdash_x M'}{\Gamma; \Delta \Vdash_x M}\ \mathsf{ls\_red}$$

$$\frac{\Gamma; \Delta \Vdash_x A}{\Gamma; \Delta \Vdash_x \lambda y{\star}A.M}\ \mathsf{ls\_ld} \qquad \frac{\Gamma; \Delta, y \star A \Vdash_x M}{\Gamma; \Delta \Vdash_x \lambda y{\star}A.M}\ \mathsf{ls\_lb}$$

$$\frac{\Gamma; \Delta \Vdash_x A_1}{\Gamma; \Delta \Vdash_x \Pi y{\star}A_1.A_2}\ \mathsf{ls\_pd} \qquad \frac{\Gamma; \Delta, y \star A_1 \Vdash_x A_2}{\Gamma; \Delta \Vdash_x \Pi y{\star}A_1.A_2}\ \mathsf{ls\_pb}$$

6.2.5 Strict Occurrences

$$\frac{M \xrightarrow{whr} M' \qquad \Gamma \Vdash_x M'}{\Gamma \Vdash_x M}\ \mathsf{rs\_red}$$

$$\frac{d = M : A \in \Sigma \qquad \cdot \Vdash M \qquad \Gamma; \cdot \Vdash_x S}{\Gamma \Vdash_x d \cdot S}\ \mathsf{rs\_d}$$

$$\frac{\Gamma; \cdot \Vdash_x S}{\Gamma \Vdash_x c \cdot S}\ \mathsf{rs\_c}$$

$$\frac{\Gamma, y \star A \Vdash_x M}{\Gamma \Vdash_x \lambda y \star A.M}\ \mathsf{rs\_lam}$$

6.2.6 Global Strictness

$$\frac{M \xrightarrow{whr} M' \qquad \Gamma \Vdash M'}{\Gamma \Vdash M}\ \textsf{gs\_red}$$

$$\frac{d = M : A \in \Sigma \qquad \cdot \Vdash M \qquad \Gamma \Vdash M \cdot S}{\Gamma \Vdash d \cdot S}\ \textsf{gs\_d}$$

$$\frac{}{\Gamma \Vdash c \cdot S}\ \textsf{gs\_c}$$

$$\frac{\Gamma, x : A \Vdash_x M \qquad \Gamma, x : A \Vdash M}{\Gamma \Vdash \lambda x{:}A.M}\ \textsf{gs\_lam}$$

$$\frac{\Gamma, x \div A \Vdash M}{\Gamma \Vdash \lambda x{\div}A.M}\ \textsf{gs\_ilam}$$

Showing that strict functions — M such that $\Gamma \Vdash M$ — are injective by direct induction fails, so we generalize over substitutions.

Lemma 6.10 (Completeness of Strictness) *Let σ_1, σ_2 be substitutions such that $\Gamma'; \Delta' \vdash \sigma_1, \sigma_2 : \Gamma; \Delta$. That is, σ_1, σ_2 map variables in Γ to objects with variables in Γ', and are the identity on Δ except for types — σ_i maps $y : A \in \Delta$ to $y : (A[\sigma_i]) \in \Delta'$. Assume also that C is a base type, i.e. not of the form $\Pi x \star A.K$.*

1. *If $x : A \in \Gamma$, $\Gamma; \Delta \Vdash_x M$, $\Gamma, \Delta \vdash M : B$, and $(\Gamma')^-, (\Delta')^- \vdash M[\sigma_1] \Longleftrightarrow M[\sigma_2] : B^-$, then $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.*
2. *If $x : A \in \Gamma$, $\Gamma; \Delta \Vdash_x B$, $\Gamma, \Delta \vdash B : \textsf{type}$, and $(\Gamma')^-, (\Delta')^- \vdash B[\sigma_1] = B[\sigma_2] : \textsf{type}^-$, then $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.*
3. *If $x : A \in \Gamma$, $\Gamma; \Delta \Vdash_x S$, $\Gamma, \Delta \vdash S : B > C$, and $\Gamma', \Delta' \vdash S[\sigma_1] \Longleftrightarrow S[\sigma_2] : B^- > C^-$, then $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.*
4. *If $x : A \in \Gamma$, $\Gamma \Vdash_x M$, $\Gamma, \Delta \vdash M : B$, $\Gamma', \Delta' \vdash S : B[\sigma_1] > C$, and $(\Gamma')^- \vdash M[\sigma_1] \cdot S \Longleftrightarrow M[\sigma_2] \cdot S : C^-$, then $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.*
5. *If $\Gamma \Vdash M$, $\Gamma \vdash M : B$, $\Gamma' \vdash S_1 : B[\sigma_1] > C$, $\Gamma' \vdash S_2 : B[\sigma_2] > C$, and $\Gamma' \vdash M[\sigma_1] \cdot S_1 = M[\sigma_2] \cdot S_2 : C$, then $(\Gamma')^- \vdash S_1 \Longleftrightarrow S_2 : B^- > C^-$.*

Proof Simultaneous induction over the rules defining the $\Vdash$ judgments. We show some representative cases.

Case: Part 1,

$$\frac{\Delta^- \vdash S\ \textsf{pat} \qquad \Delta \vdash S : D > C}{\Gamma; \Delta \Vdash_x x \cdot S}\ \textsf{ls\_pat}$$

Observe that $M[\sigma_1]$ is $\sigma_1(x) \cdot S[\sigma_1]$ and $M[\sigma_2]$ is $\sigma_2(x) \cdot S[\sigma_2]$. By a simple induction using the fact that σ_1, σ_2 are the identity on Δ, we have $\Delta^- \vdash S[\sigma_1] \sim S[\sigma_2]\ \textsf{pat}$, so by Lemma 6.9, $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : D^-$. Because σ_1, σ_2 are valid substitutions, D^- and A^- are identical.

Case: Part 3,

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma; \Delta \Vdash_x M}}{\Gamma; \Delta \Vdash_x (M; S)}\ \textsf{ls\_hd}$$

If $(\Gamma')^- \vdash (M;S)[\sigma_1] \Longleftrightarrow (M;S)[\sigma_2] : B^- \to C^-$, then by inversion $(\Gamma')^- \vdash M[\sigma_1] \Longleftrightarrow M[\sigma_2] : D^-$ for some $D-$, and the induction hypothesis (part 1) on $\mathcal{D}_1$ gives $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.

Case: Part 4,

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma; \cdot \Vdash_x S'}}{\Gamma \Vdash_x c \cdot S'}\ \textsf{rs\_c}$$

If $(\Gamma')^- \vdash (c \cdot S'[\sigma_1]) \cdot S \Longleftrightarrow (c \cdot S'[\sigma_2]) \cdot S : C^-$, then since C is a base type and both terms are weak head-normal, we can apply inversion to the algorithmic equality rules to see that $(\Gamma')^- \vdash (c \cdot S'[\sigma_1]) \cdot S \longleftrightarrow (c \cdot S'[\sigma_2]) \cdot S : C^-$. By inversion principles and an easy induction, we have $(\Gamma')^- \vdash S'[\sigma_1] \Longleftrightarrow S'[\sigma_2] : D^- > B^-$ for some D such that $\Gamma \vdash S' : D > B$ and $c : D \in \Sigma$. By the induction hypothesis (part 3) on $\mathcal{D}_1$ we have $(\Gamma')^- \vdash \sigma_1(x) \Longleftrightarrow \sigma_2(x) : A^-$.

Case: Part 5,

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma, x : A \Vdash_x M} \quad \overset{\mathcal{D}_2}{\Gamma, x : A \Vdash M'}}{\Gamma \Vdash \lambda x{:}A.M'}\ \textsf{gs\_lam}$$

By Lemma 4.14, we know from $\Gamma \vdash \lambda x{:}A.M' : B$ that $\Gamma \vdash B = \Pi x{:}A.A' : \textsf{type}$ and $\Gamma, x{:}A \vdash M' : A'$. By Lemma 4.16, B must be of the form $\Pi x{:}B''.B'$ such that $\Gamma \vdash B'' = A : \textsf{type}$ and $\Gamma, x{:}B'' \vdash B' = A' : \textsf{type}$. Since for each $i \in \{1, 2\}$ we have $\Gamma' \vdash S_i : \Pi x{:}B''[\sigma_i].B'[\sigma_i] > C$, Lemma 6.7 tells us that S_i is of the form $(M_i'; S_i')$ such that $\Gamma' \vdash S_i' : B'[\sigma_i, M_i'/x] > C$ and $\Gamma' \vdash M_i' : B''[\sigma_i]$. Using what we know about the structure of M, S_1, S_2, we see that the equation $\Gamma' \vdash M[\sigma_1] \cdot S_1 = M[\sigma_2] \cdot S_2 : C$ in fact says that

$$\Gamma' \vdash M'[\sigma_1, M_1'/x] \cdot S_1' = M'[\sigma_2, M_2'/x] \cdot S_2' : C$$

It is only a matter of converting equal types to see that $\sigma_i, M_i'/x$ are valid substitutions and $\Gamma, x{:}B'' \vdash M' : B'$. By the induction hypothesis, $(\Gamma')^- \vdash S_1' = S_2' : (B')^- > C^-$. Now it can be shown by completeness of algorithmic equality and a simple induction that $(\Gamma')^- \vdash M'[\sigma_1, M_1'/x] \cdot S_1' \Longleftrightarrow M'[\sigma_1, M_2'/x] \cdot S_1' : C^-$. By the induction hypothesis (part 4) on $\mathcal{D}_1$ we infer $(\Gamma')^- \vdash M_1' \Longleftrightarrow M_2' : (B'')^-$. Therefore by $\textsf{ase\_cons}$ we conclude

$$(\Gamma')^- \vdash (M_1'; S_1') \Longleftrightarrow (M_2'; S_2') : (B'')^- \to (B')^- > C^-$$

Case: Part 5,

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_1}{\Gamma, x \div A \Vdash M'}}{\Gamma \Vdash \lambda x{\div}A.M'}\ \textsf{gs\_ilam}$$

By Lemma 4.14, we know from $\Gamma \vdash \lambda x{\div}A.M' : B$ that $\Gamma \vdash B = \Pi x{\div}A.A' : \textsf{type}$ and $\Gamma, x{\div}A \vdash M' : A'$. By Lemma 4.16, B must be of the form $\Pi x{\div}B''.B'$ such that $\Gamma \vdash B'' = A : \textsf{type}$ and $\Gamma, x{\div}B'' \vdash B' = A' : \textsf{type}$. Since for each $i \in \{1, 2\}$ we have $\Gamma' \vdash S_i : \Pi x{\div}B''[\sigma_i].B'[\sigma_i] > C$, Lemma 6.7 tells us that S_i is of the form $(M'_i{\div}S'_i)$ such that $\Gamma' \vdash S'_i : B'[\sigma_i, M'_i/x] > C$ and $\Gamma' \vdash M'_i : B''[\sigma_i]$. Using what we know about the structure of M, S_1, S_2, we see that the equation $\Gamma' \vdash M[\sigma_1] \cdot S_1 = M[\sigma_2] \cdot S_2 : C$ in fact says that

$$\Gamma' \vdash M'[\sigma_1, M'_1/x] \cdot S'_1 = M'[\sigma_2, M'_2/x] \cdot S'_2 : C$$

It is only a matter of converting equal types to see that $\sigma_i, M'_i/x$ are valid substitutions and $\Gamma, x{\div}B'' \vdash M' : B'$. By the induction hypothesis, $(\Gamma')^- \vdash S'_1 = S'_2 : (B')^- > C^-$, so we can derive

$$\cfrac{(\Gamma')^- \vdash S'_1 \Longleftrightarrow S'_2 : (B')^- > C^-}{(\Gamma')^- \vdash (M'_1{\div}S'_1) \Longleftrightarrow (M'_2{\div}S'_2) : (B'')^- \overset{\div}{\to} (B')^- > C^-}\ \textsf{ase\_icons}$$

■

Theorem 6.11 (Injectivity) *If $d : A = M$ is strict, i.e. $\cdot \Vdash M$, then d is injective. That is, if $\Gamma \vdash S_1, S_2 : A > C$, then $\Gamma \vdash d \cdot S_1 = d \cdot S_2 : C$ implies $\Gamma \vdash S_1 = S_2 : A > C$.*

Proof Follows from soundness and Lemma 6.10 (part 5), with σ_1, σ_2 both the identity substitution. ■

References

[AB01] Steve Awodey and Andrej Bauer. Propositions as [Types]. Technical report, Institut Mittag-Leffler, 2001.

[App00] Andrew Appel. Hints on proving theorems in Twelf. Web page. URL: `http://www.cs.princeton.edu/~appel/twelf-tutorial/`, 2000.

[Bar98] G. Barthe. The relevance of proof-irrelevance. In S. Skyum K. Larsen and G. Winskel, editors, *Proceedings of ICALP'98, LNCS*, 1998.

[C+86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[CP97] Iliano Cervesato and Frank Pfenning. A linear spine calculus. Technical Report CMU-CS-97-125, Department of Computer Science, Carnegie Mellon University, April 1997.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HP01] Robert Harper and Frank Pfenning. On the equivalence and canonical forms in the LF type theory. Technical report, Carnegie Mellon University, 2001.

[PE98] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1998.

[Pfe01] Frank Pfenning. Intensionality, extensionality and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, 2001.

[PS98] Frank Pfenning and Carsten Schürmann. Algorithms for equality and unification in the presence of notational definitions. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs*, pages 179–193, Kloster Irsee, Germany, March 1998. Springer-Verlag LNCS 1657.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.