REFINEMENT TYPES FOR LOGICAL FRAMEWORKS AND THEIR INTERPRETATION AS PROOF IRRELEVANCE

WILLIAM LOVAS AND FRANK PFENNING

Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail address: wlovas@cs.cmu.edu

Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail address: fp@cs.cmu.edu

Abstract. Refinement types sharpen systems of simple and dependent types by offering expressive means to more precisely classify well-typed terms. We present a system of refinement types for LF in the style of recent formulations where only canonical forms are well-typed. Both the usual LF rules and the rules for type refinements are bidirectional, leading to a straightforward proof of decidability of typechecking even in the presence of intersection types. Because we insist on canonical forms, structural rules for subtyping can now be derived rather than being assumed as primitive. We illustrate the expressive power of our system with examples and validate its design by demonstrating a precise correspondence with traditional presentations of subtyping.

Proof irrelevance provides a mechanism for selectively hiding the identities of terms in type theories. We show that LF refinement types can be interpreted as predicates using proof irrelevance, establishing a uniform relationship between two previously studied concepts in type theory. The interpretation and its correctness proof are surprisingly complex, lending support to the claim that refinement types are a fundamental construct rather than just a convenient surface syntax for certain uses of proof irrelevance.

1. Introduction

LF was created as a framework for defining logics and programming languages [HHP93]. Since its inception, it has been used to represent and formalize reasoning about a number of deductive systems, which are prevalent in the study of logics and programming languages.[1] In its most recent incarnation as the Twelf metalogic [PS99], it has been used to encode and mechanize the metatheory of programming languages that are prohibitively complex to reason about on paper [Cra03, LCH07].

It has long been recognized that some LF encodings would benefit from the addition of a subtyping mechanism to LF [Pfe93, AC01]. In LF encodings, judgments are represented

1998 ACM Subject Classification: F.3.3 Studies of Program Constructs — *type structure*, F.4.1 Mathematical logic — *lambda calculus and related systems*.

Key words and phrases: Logical frameworks, refinement types, proof irrelevance.

The work was partially supported by the Fundação para a Ciência e Tecnologia (FCT), Portugal, under a grant from the Information and Communications Technology Institute (ICTI) at Carnegie Mellon University.

[1] See [Pfe01b] for an introduction to logical frameworks and further references.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

by type families, and many subsets of data types and judgmental inclusions can be elegantly represented via subtyping.

Prior work has explored adding subtyping and intersection types to LF via *refinement types* [Pfe93]. Many of that system's metatheoretic properties were proven indirectly by translation into other systems, though, giving little insight into notions of adequacy or implementation strategies. We begin this paper by presenting a refinement type system for LF based on the modern *canonical forms* approach [WCPW02, HL07], and by doing so we obtain direct proofs of important properties like decidability. Moreover, the theory of canonical forms provides the basis for a study of adequacy theorems exploiting refinement types.

In canonical forms-based LF, only β-normal η-long terms are well-typed — the syntax restricts terms to being β-normal, while the typing relation forces them to be η-long. Since standard substitution might introduce redexes even when substituting a normal term into a normal term, it is replaced with a notion of *hereditary substitution* that contracts redexes along the way, yielding another normal term. Since only canonical forms are admitted, type equality is just α-equivalence, and typechecking is manifestly decidable.

Canonical forms are exactly the terms one cares about when adequately encoding a language in LF, so this approach loses no expressivity. Since all terms are normal, there is no notion of reduction, and thus the metatheory need not directly treat properties related to reduction, such as subject reduction, Church-Rosser, or strong normalization. All of the metatheoretic arguments become straightforward structural inductions, once the theorems are stated properly.

By introducing a layer of refinements distinct from the usual layer of types, we prevent subtyping from interfering with our extension's metatheory. We also follow the general philosophy of prior work on refinement types [FP91, Fre94, Dav05] in only assigning refined types to terms already well-typed in pure LF, ensuring that our extension is conservative.

As a simple example, we study the representation of natural numbers as well as even and odd numbers. In normal logical discourse, we might define these with the following grammar:

$$\begin{array}{lcll} \text{Natural numbers} & n & ::= & z \mid s(n) \\ \text{Even numbers} & e & ::= & z \mid s(o) \\ \text{Odd numbers} & o & ::= & s(e) \end{array}$$

The first line can be seen as defining the *abstract syntax* of natural numbers in unary form, the second and third lines as defining two subsets of the natural numbers defined in the first line. We will follow this informal convention, and represent the first as a *type* with two constructors.

nat : type.
z : nat.
s : $nat \to nat$.

The second and third line define even and odd numbers as a subset of the natural numbers, which we represent as *refinements* of the type nat.

$even \sqsubset nat$. $odd \sqsubset nat$.
$z :: even$.
$s :: even \to odd \ \wedge \ odd \to even$.

In the above, $even \sqsubset nat$ declares *even* as a refinement of the type *nat*, and the declarations using "::" give more precise sorts for the constructors z and s. Note that since the successor function satisfies two unrelated properties, we give two refinements for it using an intersection sort. We can give similar representations of all regular tree grammars as refinements, which then represent regular tree types [DZ92]. Our language generalizes this further to allow binding operators and dependent types, both of which it inherits from LF, thereby going far beyond what can be recognized with tree automata [CDG+07].

Already in this example we can see that it is natural to use refinements to represent certain subsets of data types. Conversely, refinements can be interpreted as defining subsets. In the second part of this paper, we exhibit an interpretation of LF refinement types which we refer to as the "*subset interpretation*", since a sort refining a type is interpreted as a predicate embodying the refinement, and the set of terms having that sort is simply the subset of terms of the refined type that also satisfy the predicate. For example, under the subset interpretation, we translate the refinements *even* and *odd* to predicates on natural numbers. The refinement declarations for z and s turn into constructors for proofs of these predicates.

$even : nat \to \mathsf{type}.\quad odd : nat \to \mathsf{type}.$
$\widehat{z} \ : even\ z.$
$\widehat{s}_1 : \Pi x{:}nat.\ even\ x \to odd\ (s\ x).$
$\widehat{s}_2 : \Pi x{:}nat.\ odd\ x \to even\ (s\ x).$

The successor function's two unrelated sorts translate to proof constructors for two different predicates.

We show that our interpretation is correct by proving, for instance, that a term N has sort S if and only if its translation $\widehat{N}$ has type $\widehat{S}(N)$, where $\widehat{S}(\ -\ )$ is the translation of the sort S into a type family representing a predicate; thus, an adequate encoding using refinement types remains adequate after translation. The chief complication in proving correctness is the dependency of types on terms, which forces us to deal with a *coherence* problem [BTCGS91, Rey91].

Normally, subset interpretations are not subject to the issue of coherence—that is, of ensuring that the interpretation of a judgment is independent of its derivation—since the terms in the target of the translation are *the same* as the terms in the source, just with the stipulation that a certain property hold of them. The proofs of these properties are computationally immaterial, so they may simply be ignored. But the presence of full dependent types in LF means that the interpretation of a sort might depend on these proofs, potentially violating the adequacy of representations.

In order to solve the coherence problem we employ *proof irrelevance*, a technique used in type theories to selectively hide the identities of terms representing proofs [Pfe01a, AB04]. In the example, the terms whose identity should be irrelevant are those constructing proofs of $odd(n)$ and $even(n)$, that is, those composed from $\widehat{z}$, $\widehat{s}_1$, and $\widehat{s}_2$.

The subset interpretation completes our intuitive understanding of refinement types as representing subsets of types. It turns out that in the presence of variable binding and dependent types, this understanding is considerably more difficult to attain than it might seem from the small example above.

In the remainder of the paper, we describe our refinement type system alongside a few illustrative examples (Section 2). Then we explore its metatheory and sketch proofs of key results, including decidability (Section 3). We note that our approach leads to subtyping

only being defined at base types, but we show that this is no restriction at all: subtyping at higher types is intrinsically present due to the use of canonical forms (Section 4). Next, we take a brief detour to review prior work on proof irrelevance (Section 5), setting the stage for our subset interpretation and proofs of its correctness (Section 6). Finally, we offer some concluding remarks on the broader implications of our work (Section 7).

This paper represents a combination of the developments in a technical report on the basic design of LF with refinement types [LP08a, LP08b] and a conference paper sketching the subset interpretation [LP09].

2. System and Examples

We present our system of LF with Refinements, LFR, through several examples. In what follows, R refers to atomic terms and N to normal terms. Our atomic and normal terms are exactly the terms from canonical presentations of LF.

$$R ::= c \mid x \mid R\ N \qquad \text{atomic terms}$$
$$N, M ::= R \mid \lambda x.\, N \qquad \text{normal terms}$$

In this style of presentation, typing is defined bidirectionally by two judgments: $R \Rightarrow A$, which says atomic term R *synthesizes* type A, and $N \Leftarrow A$, which says normal term N *checks* against type A. Since λ-abstractions are always checked against a given type, they need not be decorated with their domain types.

Types are similarly stratified into atomic and normal types.

$$P ::= a \mid P\ N \qquad \text{atomic type families}$$
$$A, B ::= P \mid \Pi x{:}A.\, B \qquad \text{normal type families}$$

The operation of hereditary substitution, written $[N/x]_A$, is a partial function which computes the normal form of the standard capture-avoiding substitution of N for x. It is indexed by the putative type of x, A, to ensure termination, but neither the variable x nor the substituted term N are required to bear any relation to this type index for the operation to be defined. We show in Section 3 that when N and x *do* have type A, hereditary substitution is a total function on well-formed terms.

As a philosophical aside, we note that restricting our attention to normal terms in this way is similar to the idea of restricting one's attention to cut-free proofs in a sequent calculus [Pfe00]. Showing that hereditary substitution can always compute a canonical form is analogous to showing the cut rule admissible. And just as cut admissibility may be used to prove a *cut elimination* theorem, hereditary substitution may be used to prove a *normalization* theorem relating the canonical approach to traditional formulations. We will not explore the relationship any further in the present work: the canonical terms are the only ones we care about when formalizing deductive systems in a logical framework, so we simply take the canonical presentation as primary.

Our layer of refinements uses metavariables Q for atomic sorts and S for normal sorts. These mirror the definition of types above, except for the addition of intersection and "top" sorts.

$$Q ::= s \mid Q\ N \qquad \text{atomic sort families}$$
$$S, T ::= Q \mid \Pi x{::}S \sqsubset A.\, T \mid \top \mid S_1 \wedge S_2 \qquad \text{normal sort families}$$

Sorts are related to types by a refinement relation, $S \sqsubset A$ ("S refines A"), discussed below. We only sort-check well-typed terms, and a term of type A can be assigned a sort S only when $S \sqsubset A$. These constraints are collectively referred to as the "refinement restriction". We occasionally omit the "$\sqsubset A$" from function sorts when it is clear from context.

Deductive systems are encoded in LF using the *judgments-as-types* principle [HHP93, HL07]: syntactic categories are represented by simple types, and judgments over syntax are represented by dependent type families. Derivations of judgments are inhabitants of those type families, and well-formed derivations correspond to well-typed LF terms. An LF signature is a collection of kinding declarations $a : K$ and typing declarations $c : A$ that establishes a set of syntactic categories, a set of judgments, and inhabitants of both. In LFR, we can represent syntactic subsets or sets of derivations that have certain *properties* using sorts. Thus one might say that the methodology of LFR is *properties-as-sorts*.

2.1. **Example: Natural Numbers.** For the first running example we will use the natural numbers in unary notation. In LF, they would be specified as follows

$$nat : \mathsf{type}.$$
$$z : nat.$$
$$s : nat \to nat.$$

These declarations establish a syntactic category of natural numbers populated by two constructors, a constant constructor representing zero and a unary constructor representing the successor function.

Suppose we would like to distinguish the odd and the even numbers as refinements of the type of all numbers.

$$even \sqsubset nat.$$
$$odd \sqsubset nat.$$

The form of the declaration is $s \sqsubset a$ where a is a type family already declared and s is a new sort family. Sorts headed by s are declared in this way to refine types headed by a. The relation $S \sqsubset A$ is extended through the whole sort hierarchy in a compositional way.

Next we declare the sorts of the constructors. For zero, this is easy:

$$z :: even.$$

The general form of this declaration is $c :: S$, where c is a constant already declared in the form $c : A$, and where $S \sqsubset A$. The declaration for the successor is slightly more difficult, because it maps even numbers to odd numbers and vice versa. In order to capture both properties simultaneously we need to use an *intersection sort*, written as $S_1 \wedge S_2$.[2]

$$s :: even \to odd \ \wedge \ odd \to even.$$

In order for an intersection to be well-formed, both components must refine the same type. The nullary intersection $\top$ can refine any type, and represents the maximal refinement of that type.[3]

$$\frac{s \sqsubset a \in \Sigma}{s\ N_1 \ldots N_k \sqsubset a\ N_1 \ldots N_k} \qquad \frac{S \sqsubset A \qquad T \sqsubset B}{\Pi x{::}S.\, T \sqsubset \Pi x{:}A.\, B} \qquad \frac{S_1 \sqsubset A \qquad S_2 \sqsubset A}{S_1 \wedge S_2 \sqsubset A} \qquad \frac{}{\top \sqsubset A}$$

[2] Intersection has lower precedence than arrow.

[3] As usual in LF, we use $A \to B$ as shorthand for the dependent type $\Pi x{:}A.\, B$ when x does not occur in B.

To show that the declaration for s is well-formed, we establish that $\mathit{even} \to \mathit{odd} \wedge \mathit{odd} \to \mathit{even} \sqsubset \mathit{nat} \to \mathit{nat}$.

The *refinement relation* $S \sqsubset A$ should not be confused with the usual *subtyping relation*. Although each is a kind of subset relation[4], they are quite different: Subtyping relates two types, is contravariant in the domains of function types, and is transitive, while refinement relates a sort to a type, so it does not make sense to consider its variance or whether it is transitive. We will discuss subtyping below and in Section 4.

Now suppose that we also wish to distinguish the strictly positive natural numbers. We can do this by introducing a sort *pos* refining *nat* and declaring that the successor function yields a *pos* when applied to anything, using the maximal sort.

$\mathit{pos} \sqsubset \mathit{nat}$.
$s :: \cdots \wedge \top \to \mathit{pos}$.

Since we only sort-check well-typed programs and s is declared to have type $\mathit{nat} \to \mathit{nat}$, the sort $\top$ here acts as a sort-level reflection of the entire *nat* type.

We can specify that all odds are positive by declaring *odd* to be a subsort of *pos*.

$\mathit{odd} \leq \mathit{pos}$.

Although any ground instance of *odd* is evidently *pos*, we need the subsorting declaration to establish that variables of sort *odd* are also *pos*.

Putting it all together, we have the following:

$\mathit{even} \sqsubset \mathit{nat}$. $\quad \mathit{odd} \sqsubset \mathit{nat}$. $\quad \mathit{pos} \sqsubset \mathit{nat}$.
$\mathit{odd} \leq \mathit{pos}$.
$z :: \mathit{even}$.
$s :: \mathit{even} \to \mathit{odd} \ \wedge \ \mathit{odd} \to \mathit{even} \ \wedge \ \top \to \mathit{pos}$.

Now we should be able to verify that, for example, $s\ (s\ z) \Leftarrow \mathit{even}$. To explain how, we analogize with pure canonical LF. Recall that atomic types have the form $a\ N_1 \ldots N_k$ for a type family a and are denoted by P. Arbitrary types A are either atomic (P) or (dependent) function types ($\Pi x{:}A.\, B$). Canonical terms are then characterized by the rules shown in the left column above.

There are two typing judgments, $N \Leftarrow A$ which means that N checks against A (both given) and $R \Rightarrow A$ which means that R synthesizes type A (R given as input, A produced as output). Both take place in a context Γ assigning types to variables. To force terms to be η-long, the rule for checking an atomic term R only checks it at an atomic type P. It does so by synthesizing a type P' and comparing it to the given type P. In canonical LF, all types are already canonical, so this comparison is just α-equality.

On the right-hand side we have shown the corresponding rules for sorts. First, note that the format of the context Γ is slightly different, because it declares sorts for variables, not just types. The rules for functions and applications are straightforward analogues to the rules in ordinary LF. The rule **switch** for checking atomic terms R at atomic sorts Q replaces the equality check with a subsorting check and is the only place where we appeal to subsorting (defined below). For applications, we use the type A that refines the type S as the index parameter of the hereditary substitution.

[4]It may help to recall the interpretation of $S \sqsubset A$: for a term to be judged to have sort S, it must already have been judged to have type A for some A such that $S \sqsubset A$. Thus, the refinement relation represents an inclusion "by fiat": every term with sort S is also a term of type A, by invariant. By contrast, subsorting $S_1 \leq S_2$ is a more standard sort of inclusion: every term with sort S_1 is also a term of sort S_2, by subsumption (see Section 4).

Canonical LF	LF with Refinements
$\dfrac{\Gamma, x{:}A \vdash N \Leftarrow B}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x{:}A.\, B}$	$\dfrac{\Gamma, x{::}S \sqsubset A \vdash N \Leftarrow T}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x{::}S \sqsubset A.\, T}\ (\Pi\textbf{-I})$
$\dfrac{\Gamma \vdash R \Rightarrow P' \qquad P' = P}{\Gamma \vdash R \Leftarrow P}$	$\dfrac{\Gamma \vdash R \Rightarrow Q' \qquad Q' \leq Q}{\Gamma \vdash R \Leftarrow Q}\ (\textbf{switch})$
$\dfrac{x{:}A \in \Gamma}{\Gamma \vdash x \Rightarrow A}$ $\qquad$ $\dfrac{c{:}A \in \Sigma}{\Gamma \vdash c \Rightarrow A}$	$\dfrac{x{::}S \sqsubset A \in \Gamma}{\Gamma \vdash x \Rightarrow S}\ (\textbf{var})$ $\qquad$ $\dfrac{c :: S \in \Sigma}{\Gamma \vdash c \Rightarrow S}\ (\textbf{const})$
$\dfrac{\Gamma \vdash R \Rightarrow \Pi x{:}A.\, B \qquad \Gamma \vdash N \Leftarrow A}{\Gamma \vdash R\ N \Rightarrow [N/x]_A\, B}$	$\dfrac{\Gamma \vdash R \Rightarrow \Pi x{::}S \sqsubset A.\, T \qquad \Gamma \vdash N \Leftarrow S}{\Gamma \vdash R\ N \Rightarrow [N/x]_A\, T}\ (\Pi\textbf{-E})$

Subsorting is exceedingly simple: it only needs to be defined on atomic sorts, and is just the reflexive and transitive closure of the declared subsorting relationship.

$$\frac{s_1 \leq s_2 \in \Sigma}{s_1\ N_1 \ldots N_k \leq s_2\ N_1 \ldots N_k} \qquad \frac{}{Q \leq Q} \qquad \frac{Q_1 \leq Q' \qquad Q' \leq Q_2}{Q_1 \leq Q_2}$$

The sorting rules do not yet treat intersections. In line with the general bidirectional nature of the system, the introduction rules are part of the *checking* judgment, and the elimination rules are part of the *synthesis* judgment. Binary intersection $S_1 \wedge S_2$ has one introduction and two eliminations, while nullary intersection $\top$ has just one introduction.

$$\frac{\Gamma \vdash N \Leftarrow S_1 \qquad \Gamma \vdash N \Leftarrow S_2}{\Gamma \vdash N \Leftarrow S_1 \wedge S_2}\ (\wedge\textbf{-I}) \qquad \frac{}{\Gamma \vdash N \Leftarrow \top}\ (\top\textbf{-I})$$

$$\frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma \vdash R \Rightarrow S_1}\ (\wedge\textbf{-E}_1) \qquad \frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma \vdash R \Rightarrow S_2}\ (\wedge\textbf{-E}_2)$$

Note that although (canonical forms-style) LF type synthesis is unique, LFR sort synthesis is not, due to the intersection elimination rules.

Now we can see how these rules generate a deduction of $s\ (s\ z) \Leftarrow \mathit{even}$. The context is always empty and therefore omitted. To save space, we abbreviate *even* as e, *odd* as o, and *pos* as p, and we omit reflexive uses of subsorting.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash s \Rightarrow e \to o \wedge (o \to e \wedge \top \to p)}}{\vdash s \Rightarrow o \to e \wedge \top \to p}}{\vdash s \Rightarrow o \to e} \qquad \dfrac{\dfrac{\dfrac{}{\vdash s \Rightarrow e \to o \wedge (\ldots)}}{\vdash s \Rightarrow e \to o} \quad \dfrac{\dfrac{}{\vdash z \Rightarrow e}}{\vdash z \Leftarrow e}}{\vdash s\ z \Rightarrow o}}{\vdash s\ z \Leftarrow o}$$

$$\dfrac{\vdash s\ (s\ z) \Rightarrow e}{\vdash s\ (s\ z) \Leftarrow e}$$

Using the $\wedge$-**I** rule, we can check that $s\ z$ is both odd and positive:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash s\ z \Leftarrow o & \vdash s\ z \Leftarrow p\end{array}}{\vdash s\ z \Leftarrow o \wedge p}$$

Each remaining subgoal now proceeds similarly to the above example.

To illustrate the use of sorts with non-trivial type *families*, consider the definition of the *double* relation in LF. We declare a type family representing the doubling judgment and populate it with two proof rules.

$$\begin{array}{l} \mathit{double} : \mathit{nat} \to \mathit{nat} \to \mathsf{type}. \\ \mathit{dbl/z} : \mathit{double}\ z\ z. \\ \mathit{dbl/s} : \Pi X{:}\mathit{nat}.\, \Pi Y{:}\mathit{nat}.\, \mathit{double}\ X\ Y \to \mathit{double}\ (s\ X)\ (s\ (s\ Y)). \end{array}$$

With sorts, we can now directly express the property that the second argument to *double* must be even. But to do so, we require a notion analogous to *kinds* that may contain sort information. We call these *classes* and denote them by L.

$$\begin{array}{lll} K ::= \mathsf{type} \mid \Pi x{:}A.\, K & & \text{kinds} \\ L ::= \mathsf{sort} \mid \Pi x{::}S \sqsubset A.\, L \mid \top \mid L_1 \wedge L_2 & & \text{classes} \end{array}$$

Classes L mirror kinds K, and they have a refinement relation $L \sqsubset K$ similar to $S \sqsubset A$. (We elide the rules here, but they are included in Appendix A.) Now, the general form of the $s \sqsubset a$ declaration is $s \sqsubset a :: L$, where $a : K$ and $L \sqsubset K$; this declares sort constant s to refine type constant a and to have class L.

For now, we reuse the type name *double* as a sort, as no ambiguity can result. As before, we use $\top$ to represent a *nat* with no additional restrictions.

$$\begin{array}{l} \mathit{double} \sqsubset \mathit{double} :: \top \to \mathit{even} \to \mathsf{sort}. \\ \mathit{dbl/z} :: \mathit{double}\ z\ z. \\ \mathit{dbl/s} :: \Pi X{::}\top.\, \Pi Y{::}\mathit{even}.\, \mathit{double}\ X\ Y \to \mathit{double}\ (s\ X)\ (s\ (s\ Y)). \end{array}$$

After these declarations, it would be a static *sort error* to pose a query such as "`?-` $\mathit{double}\ X\ (s\ (s\ (s\ z)))$." before any search is ever attempted. In LF, queries like this could fail after a long search or even not terminate, depending on the search strategy. One of the important motivations for considering sorts for LF is to avoid uncontrolled search in favor of decidable static properties whenever possible.

The tradeoff for such precision is that now sort checking itself is non-deterministic and has to perform search because of the choice between the two intersection elimination rules. As Reynolds has shown, this non-determinism causes intersection type checking to be PSPACE-hard [Rey96], even for normal terms as we have here [Rey89]. Using techniques such as focusing, we believe that for practical cases they can be analyzed efficiently for the purpose of sort checking.[5]

2.2. **A Second Example: The λ-Calculus.** As a second example, we use an intrinsically typed version of the call-by-value simply-typed λ-calculus. This means every object language expression is indexed by its object language type. We use sorts to distinguish the set of *values* from the set of arbitrary *computations*. While this can be encoded in LF in a variety of ways, it is significantly more cumbersome.

$$\begin{array}{ll} \mathit{tp} : \mathsf{type}. & \%\ \text{the type of object language types} \\ \rightarrowtriangle : \mathit{tp} \to \mathit{tp} \to \mathit{tp}. & \%\ \text{object language function space} \\ \textbf{\%infix}\ \mathit{right}\ 10\ \rightarrowtriangle\ . & \\ & \\ \mathit{exp} : \mathit{tp} \to \mathsf{type}. & \%\ \text{the type of expressions} \end{array}$$

[5]The present paper concentrates primarily on decidability, though, not efficiency.

$cmp \sqsubset exp.$ % the sort of computations
$val \sqsubset exp.$ % the sort of values

$val \leq cmp.$ % every value is a (trivial) computation

$lam :: (val\ A \to cmp\ B) \to val\ (A \Rightarrow B).$
$app :: cmp\ (A \Rightarrow B) \to cmp\ A \to cmp\ B.$

In the last two declarations, we follow Twelf convention and leave the quantification over A and B implicit, to be inferred by type reconstruction. Also, we did not explicitly declare a type for *lam* and *app*. We posit a front end that can recover this information from the refinement declarations for *val* and *cmp*, avoiding redundancy.

The most interesting declaration is the one for the constant *lam*. The argument type $(val\ A \to cmp\ B)$ indicates that *lam* binds a variable which stands for a value of type A and the body is an arbitrary computation of type B. The result type $val\ (A \Rightarrow B)$ indicates that any λ-abstraction is a value. Now we have, for example (parametrically in A and B): $A{::}\top \sqsubset tp, B{::}\top \sqsubset tp \vdash lam\ \lambda x.\, lam\ \lambda y.\, x \Leftarrow val\ (A \Rightarrow (B \Rightarrow A))$.

Now we can express that evaluation must always returns a value. Since the declarations below are intended to represent a logic program, we follow the logic programming convention of reversing the arrows in the declaration of *ev-app*.

$eval :: cmp\ A \to val\ A \to \mathsf{sort}.$
$ev\text{-}lam :: eval\ (lam\ \lambda x.\, E\ x)\ (lam\ \lambda x.\, E\ x).$
$ev\text{-}app :: eval\ (app\ E_1\ E_2)\ V$
$\quad \leftarrow eval\ E_1\ (lam\ \lambda x.\, E_1'\ x)$
$\quad \leftarrow eval\ E_2\ V_2$
$\quad \leftarrow eval\ (E_1'\ V_2)\ V.$

Sort checking the above declarations demonstrates that when evaluation returns at all, it returns a syntactic value. Moreover, if sort reconstruction gives E_1' the "most general" sort $val\ A \to cmp\ B$, the declarations also ensure that the language is indeed call-by-value: it would be a sort error to ever substitute a computation for a *lam*-bound variable, for example, by evaluating $(E_1'\ E_2)$ instead of $(E_1'\ V_2)$ in the *ev-app* rule. An interesting question for future work is whether type reconstruction can always find such a "most general" sort for implicitly quantified metavariables.

A side note: through the use of sort families indexed by object language types, the sort checking not only guarantees that the language is call-by-value and that evaluation, if it succeeds, will always return a value, but also that the object language type of the result remains the same (type preservation).

3. Metatheory

In this section, we present some metatheoretic results about our framework. These follow a similar pattern as previous work using hereditary substitutions [WCPW02, NPP07, HL07]. We give sketches of all proofs. Technically tricky proofs are available from a companion technical report [LP08b].

Judgment:	**Substitution into:**
$[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R'$	Atomic terms (yielding atomic)
$[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R = (N', \alpha')$	Atomic terms (yielding normal)
$[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N = N'$	Normal terms
$[N_0/x_0]^{\mathrm{p}}_{\alpha_0}\, P = P'$	Atomic types
$[N_0/x_0]^{\mathrm{a}}_{\alpha_0}\, A = A'$	Normal types
$[N_0/x_0]^{\mathrm{q}}_{\alpha_0}\, Q = Q'$	Atomic sorts
$[N_0/x_0]^{\mathrm{s}}_{\alpha_0}\, S = S'$	Normal sorts
$[N_0/x_0]^{\mathrm{k}}_{\alpha_0}\, K = K'$	Kinds
$[N_0/x_0]^{\mathrm{l}}_{\alpha_0}\, L = L'$	Classes
$[N_0/x_0]^{\gamma}_{\alpha_0}\, \Gamma = \Gamma'$	Contexts

Table 1: Judgments defining hereditary substitution.

3.1. **Hereditary Substitution.** Recall that we replace ordinary capture-avoiding substitution with *hereditary substitution*, $[N/x]_A$, an operation which substitutes a normal term into a canonical form yielding another canonical form, contracting redexes "in-line". The operation is indexed by the putative type of N and x to facilitate a proof of termination. In fact, the type index on hereditary substitution need only be a simple type to ensure termination. To that end, we denote simple types by α and define an erasure to simple types $(A)^-$.

$$\alpha ::= a \mid \alpha_1 \to \alpha_2 \qquad (a\ N_1 \ldots N_k)^- = a \qquad (\Pi x{:}A.\, B)^- = (A)^- \to (B)^-$$

For clarity, we also index hereditary substitutions by the syntactic category on which they operate, so for example we have $[N/x]^{\mathrm{n}}_A\, M = M'$ and $[N/x]^{\mathrm{s}}_A\, S = S'$; Table 1 lists all of the judgments defining substitution. We write $[N/x]^{\mathrm{n}}_A\, M = M'$ as short-hand for $[N/x]^{\mathrm{n}}_{(A)^-}\, M = M'$.

Our formulation of hereditary substitution is defined judgmentally by inference rules. The only place β-redexes might be introduced is when substituting a normal term N into an atomic term R: N might be a λ-abstraction, and the variable being substituted for may occur at the head of R. Therefore, the judgments defining substitution into atomic terms are the most interesting ones.

We denote substitution into atomic terms by two judgments: $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R'$, for when the head of R is *not* x, and $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R = (N', \alpha')$, for when the head of R *is* x, where α' is the simple type of the output N'. The former is just defined compositionally; the latter is defined by two rules:

$$\frac{}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, x_0 = (N_0, \alpha_0)}\ (\textbf{subst-rn-var})$$

$$\frac{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R_1 = (\lambda x.\, N_1, \alpha_2 \to \alpha_1) \qquad [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2' \qquad [N_2'/x]^{\mathrm{n}}_{\alpha_2}\, N_1 = N_1'}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R_1\ N_2 = (N_1', \alpha_1)}\ (\textbf{subst-rn-}\beta)$$

The rule **subst-rn-var** just returns the substitutend N_0 and its putative type index α_0. The rule **subst-rn-**β applies when the result of substituting into the head of an application

is a λ-abstraction; it avoids creating a redex by hereditarily substituting into the body of the abstraction.

A simple lemma establishes that these two judgments are mutually exclusive by examining the head of the input atomic term.

$$\text{head}(x) = x \qquad \text{head}(c) = c \qquad \text{head}(R\ N) = \text{head}(R)$$

Lemma 3.1.

(1) *If* $[N_0/x_0]^{\text{rr}}_{\alpha_0}\ R = R'$, *then* $\text{head}(R) \neq x_0$.
(2) *If* $[N_0/x_0]^{\text{rn}}_{\alpha_0}\ R = (N', \alpha')$, *then* $\text{head}(R) = x_0$.

Proof. By induction on the given derivation. □

Substitution into normal terms has two rules for atomic terms R, one which calls the "rr" judgment and one which calls the "rn" judgment.

$$\frac{[N_0/x_0]^{\text{rr}}_{\alpha_0}\ R = R'}{[N_0/x_0]^{\text{n}}_{\alpha_0}\ R = R'}\ (\textbf{subst-n-atom}) \qquad \frac{[N_0/x_0]^{\text{rn}}_{\alpha_0}\ R = (R', a')}{[N_0/x_0]^{\text{n}}_{\alpha_0}\ R = R'}\ (\textbf{subst-n-atom-norm})$$

Note that the latter rule requires both the term and the type returned by the "rn" judgment to be atomic.

Every other syntactic category's substitution judgment is defined compositionally, tacitly renaming bound variables to avoid capture. For example, the remaining rule defining substitution into normal terms, the rule for substituting into a λ-abstraction, just recurses on the body of the abstraction.

$$\frac{[N_0/x_0]^{\text{n}}_{\alpha_0}\ N = N'}{[N_0/x_0]^{\text{n}}_{\alpha_0}\ \lambda x.\, N = \lambda x.\, N'}$$

Although we have only defined hereditary substitution relationally, it is easy to show that it is in fact a partial function by proving that there only ever exists one "output" for a given set of "inputs".

Theorem 3.2 (Functionality of Substitution)**.** *Hereditary substitution is a functional relation. In particular:*

(1) *If* $[N_0/x_0]^{\text{rr}}_{\alpha_0}\ R = R_1$ *and* $[N_0/x_0]^{\text{rr}}_{\alpha_0}\ R = R_2$, *then* $R_1 = R_2$,
(2) *If* $[N_0/x_0]^{\text{rn}}_{\alpha_0}\ R = (N_1, \alpha_1)$ *and* $[N_0/x_0]^{\text{rn}}_{\alpha_0}\ R = (N_2, \alpha_2)$, *then* $N_1 = N_2$ *and* $\alpha_1 = \alpha_2$,
(3) *If* $[N_0/x_0]^{\text{n}}_{\alpha_0}\ N = N_1$ *and* $[N_0/x_0]^{\text{n}}_{\alpha_0}\ N = N_2$, *then* $N_1 = N_2$,

and similarly for other syntactic categories.

Proof. Straightforward induction on the first derivation, applying inversion to the second derivation. The cases for rules **subst-n-atom** and **subst-n-atom-norm** require Lemma 3.1 to show that the second derivation ends with the same rule as the first one. □

Additionally, it is worth noting that hereditary substitution behaves just like "ordinary" substitution on terms that do not contain the distinguished free variable.

Theorem 3.3 (Trivial Substitution)**.** *Hereditary substitution for a non-occurring variable has no effect.*

(1) *If* $x_0 \notin \text{FV}(R)$, *then* $[N_0/x_0]^{\text{rr}}_{\alpha_0}\ R = R$,
(2) *If* $x_0 \notin \text{FV}(N)$, *then* $[N_0/x_0]^{\text{n}}_{\alpha_0}\ N = N$,

and similarly for other syntactic categories.

Proof. Straightforward induction on term structure. □

3.2. **Decidability.** A hallmark of the canonical forms/hereditary substitution approach is that it allows a decidability proof to be carried out comparatively early, before proving anything about the behavior of substitution, and without dealing with any complications introduced by β/η-conversions inside types. Ordinarily in a dependently typed calculus, one must first prove a substitution theorem before proving typechecking decidable, since typechecking relies on type equality, type equality relies on β/η-conversion, and β/η-conversions rely on substitution preserving well-formedness. (See for example [HP05] for a typical non-canonical forms-style account of LF definitional equality.)

In contrast, if only canonical forms are permitted, then type equality is just α-convertibility, so one only needs to show *decidability* of substitution in order to show decidability of typechecking. Since LF encodings represent judgments as type families and proof-checking as typechecking, it is comforting to have a decidability proof that relies on so few assumptions.

Lemma 3.4. *If* $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha')$*, then* α' *is a subterm of* α_0*.*

Proof. By induction on the derivation of $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha')$. In rule **subst-rn-var**, α' is the same as α_0. In rule **subst-rn-β**, our inductive hypothesis tells us that $\alpha_2 \to \alpha_1$ is a subterm of α_0, so α_1 is as well. □

By working in a constructive metalogic, we are able to prove decidability of a judgment by proving an instance of the law of the excluded middle; the computational content of the proof then represents a decision procedure.

Theorem 3.5 (Decidability of Substitution)**.** *Hereditary substitution is decidable. In particular:*

1. *Given* N_0*,* x_0*,* α_0*, and* R*, either* $\exists R'.\, [N_0/x_0]^{\mathrm{rr}}_{\alpha_0} R = R'$*, or* $\nexists R'.\, [N_0/x_0]^{\mathrm{rr}}_{\alpha_0} R = R'$*,*
2. *Given* N_0*,* x_0*,* α_0*, and* R*, either* $\exists (N', \alpha').\, [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha')$*, or* $\nexists (N', \alpha').\, [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha')$*,*
3. *Given* N_0*,* x_0*,* α_0*, and* N*, either* $\exists N'.\, [N_0/x_0]^{\mathrm{n}}_{\alpha_0} N = N'$*, or* $\nexists N'.\, [N_0/x_0]^{\mathrm{n}}_{\alpha_0} N = N'$*,*

and similarly for other syntactic categories.

Proof. By lexicographic induction on the type subscript α_0, the main subject of the substitution judgment, and the clause number. For each applicable rule defining hereditary substitution, the premises are at a smaller type subscript, or if the same type subscript, then a smaller term, or if the same term, then an earlier clause. The case for rule **subst-rn-β** relies on Lemma 3.4 to know that α_2 is a strict subterm of α_0. □

Theorem 3.6 (Decidability of Subsorting)**.** *Given* Q_1 *and* Q_2*, either* $Q_1 \leq Q_2$ *or* $Q_1 \not\leq Q_2$*.*

Proof. Since the subsorting relation $Q_1 \leq Q_2$ is just the reflexive, transitive closure of the declared subsorting relation $s_1 \leq s_2$, it suffices to compute this closure, check that the heads of Q_1 and Q_2 are related by it, and ensure that all of the arguments of Q_1 and Q_2 are equal. □

We prove decidability of typing by exhibiting a deterministic algorithmic system that is equivalent to the original. Instead of synthesizing a single sort for an atomic term, the algorithmic system synthesizes an intersection-free list of sorts, Δ.

$$\Delta ::= \cdot \mid \Delta, Q \mid \Delta, \Pi x{::}S{\sqsubset}A.\, T$$

(As usual, we freely overload comma to mean list concatenation, as no ambiguity can result.) One can think of Δ as the intersection of all its elements. Instead of applying

intersection eliminations, the algorithmic system eagerly breaks down intersections using a "split" operator, leading to a deterministic "minimal-synthesis" system.

$$\mathrm{split}(Q) = Q \qquad \mathrm{split}(S_1 \wedge S_2) = \mathrm{split}(S_1), \mathrm{split}(S_2)$$
$$\mathrm{split}(\Pi x{::}S \sqsubset A.\, T) = \Pi x{::}S \sqsubset A.\, T \qquad \mathrm{split}(\top) = \cdot$$

$$\frac{c{::}S \in \Sigma}{\Gamma \vdash c \Rrightarrow \mathrm{split}(S)} \qquad \frac{x{::}S \sqsubset A \in \Gamma}{\Gamma \vdash x \Rrightarrow \mathrm{split}(S)} \qquad \frac{\Gamma \vdash R \Rrightarrow \Delta \qquad \Gamma \vdash \Delta \mathbin{@} N = \Delta'}{\Gamma \vdash R\ N \Rrightarrow \Delta'}$$

The rule for applications uses an auxiliary judgment $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ which computes the possible types of $R\ N$ given that R synthesizes to all the sorts in Δ. It has two key rules:

$$\frac{}{\Gamma \vdash \cdot \mathbin{@} N = \cdot} \qquad \frac{\Gamma \vdash \Delta \mathbin{@} N = \Delta' \qquad \Gamma \vdash N \Lleftarrow S \qquad [N/x]^{\mathrm{s}}_A\, T = T'}{\Gamma \vdash (\Delta, \Pi x{::}S \sqsubset A.\, T) \mathbin{@} N = \Delta', \mathrm{split}(T')}$$

The other rules force the judgment to be defined when neither of the above two rules apply.

$$\frac{\Gamma \vdash \Delta \mathbin{@} N = \Delta' \qquad \Gamma \nvdash N \Lleftarrow S}{\Gamma \vdash (\Delta, \Pi x{::}S \sqsubset A.\, T) \mathbin{@} N = \Delta'} \qquad \frac{\Gamma \vdash \Delta \mathbin{@} N = \Delta' \qquad \nexists T'.\, [N/x]^{\mathrm{s}}_A\, T = T'}{\Gamma \vdash (\Delta, \Pi x{::}S \sqsubset A.\, T) \mathbin{@} N = \Delta'}$$

$$\frac{\Gamma \vdash \Delta \mathbin{@} N = \Delta'}{\Gamma \vdash (\Delta, Q) \mathbin{@} N = \Delta'}$$

Finally, to tie everything together, we define a new checking judgment $\Gamma \vdash N \Lleftarrow S$ that makes use of the algorithmic synthesis judgment; it looks just like $\Gamma \vdash N \Leftarrow S$ except for the rule for atomic terms.

$$\frac{\Gamma \vdash R \Rrightarrow \Delta \qquad Q' \in \Delta \qquad Q' \leq Q}{\Gamma \vdash R \Lleftarrow Q} \qquad \frac{\Gamma, x{::}S \sqsubset A \vdash N \Lleftarrow T}{\Gamma \vdash \lambda x.\, N \Lleftarrow \Pi x{::}S \sqsubset A.\, T}$$

$$\frac{}{\Gamma \vdash N \Lleftarrow \top} \qquad \frac{\Gamma \vdash N \Lleftarrow S_1 \qquad \Gamma \vdash N \Lleftarrow S_2}{\Gamma \vdash N \Lleftarrow S_1 \wedge S_2}$$

This new algorithmic system is manifestly decidable: despite the negative conditions in some of the premises, the definitions of the judgments are well-founded by the ordering used in the following proof. (If we wished, we could also explicitly synthesize a definition of $\Gamma \nvdash N \Lleftarrow S$, but it would not illuminate the algorithm any further.)

Theorem 3.7. *Algorithmic sort checking is decidable. In particular:*

(1) *Given* Γ *and* R*, either* $\exists \Delta.\, \Gamma \vdash R \Rrightarrow \Delta$ *or* $\nexists \Delta.\, \Gamma \vdash R \Rrightarrow \Delta$.
(2) *Given* Γ*,* N*, and* S*, either* $\Gamma \vdash N \Lleftarrow S$ *or* $\Gamma \nvdash N \Lleftarrow S$.
(3) *Given* Γ*,* Δ*, and* N*,* $\exists \Delta'.\, \Gamma \vdash \Delta \mathbin{@} N = \Delta'$.

Proof. By lexicographic induction on the term R or N, the clause number, and the sort S or the list of sorts Δ. For each applicable rule, the premises are either known to be decidable, or at a smaller term, or if the same term, then an earlier clause, or if the same clause, then either a smaller S or a smaller Δ. For clause 3, we must use our inductive hypothesis to argue that the rules cover all possibilities, and so a derivation always exists. □

Note that the algorithmic synthesis system sometimes outputs an empty Δ even when the given term is ill-typed, since the $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ judgment is always defined.

It is straightforward to show that the algorithm is sound and complete with respect to the original bidirectional system.

Lemma 3.8. *If* $\Gamma \vdash R \Rightarrow S$, *then for all* $S' \in \mathrm{split}(S)$, $\Gamma \vdash R \Rightarrow S'$.

Proof. By induction on S, making use of the $\wedge$-$\mathbf{E}_1$ and $\wedge$-$\mathbf{E}_2$ rules. □

Theorem 3.9 (Soundness of Algorithmic Typing)**.**

(1) *If* $\Gamma \vdash R \Rrightarrow \Delta$, *then for all* $S \in \Delta$, $\Gamma \vdash R \Rightarrow S$.
(2) *If* $\Gamma \vdash N \Lleftarrow S$, *then* $\Gamma \vdash N \Leftarrow S$.
(3) *If* $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$, *and for all* $S \in \Delta$, $\Gamma \vdash R \Rightarrow S$, *then for all* $S' \in \Delta'$, $\Gamma \vdash R\ N \Rightarrow S'$.

Proof. By induction on the given derivation, using Lemma 3.8. □

For completeness, we use the notation $\Delta \subseteq \Delta'$ to mean that Δ is a sublist of Δ'.

Lemma 3.10. *If* $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ *and* $\Gamma \vdash R \Rrightarrow \Delta$ *and* $\Pi x{::}S \sqsubset A.\, T \in \Delta$ *and* $\Gamma \vdash N \Lleftarrow S$ *and* $[N/x]^{\mathrm{s}}_A\, T = T'$, *then* $\mathrm{split}(T') \subseteq \Delta'$.

Proof. By straightforward induction on the derivation of $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$. □

Theorem 3.11 (Completeness for Algorithmic Typing)**.**

(1) *If* $\Gamma \vdash R \Rightarrow S$, *then* $\Gamma \vdash R \Rrightarrow \Delta$ *and* $\mathrm{split}(S) \subseteq \Delta$.
(2) *If* $\Gamma \vdash N \Leftarrow S$, *then* $\Gamma \vdash N \Lleftarrow S$.

Proof. By straightforward induction on the given derivation. In the application case, we make use of the fact that $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ is always defined and apply Lemma 3.10. □

Soundness, completeness, and decidability of the algorithmic system gives us a decision procedure for the judgment $\Gamma \vdash N \Leftarrow S$. First, decidability tells us that either $\Gamma \vdash N \Lleftarrow S$ or $\Gamma \nvdash N \Lleftarrow S$. Then soundness tells us that if $\Gamma \vdash N \Lleftarrow S$ then $\Gamma \vdash N \Leftarrow S$, while completeness tells us that if $\Gamma \nvdash N \Lleftarrow S$ then $\Gamma \nvdash N \Leftarrow S$.

Decidability theorems and proofs for other syntactic categories' formation judgments proceed similarly. When all is said and done, we have enough to show that the problem of sort checking an LFR signature is decidable.

Theorem 3.12 (Decidability of Sort Checking)**.** *Sort checking is decidable. In particular:*

(1) *Given* Γ, N, *and* S, *either* $\Gamma \vdash N \Leftarrow S$ *or* $\Gamma \nvdash N \Leftarrow S$,
(2) *Given* Γ, S, *and* A, *either* $\Gamma \vdash S \sqsubset A$ *or* $\Gamma \nvdash S \sqsubset A$, *and*
(3) *Given* Σ, *either* $\vdash \Sigma\ \mathsf{sig}$ *or* $\nvdash \Sigma\ \mathsf{sig}$.

3.3. **Identity and Substitution Principles.** Since well-typed terms in our framework must be canonical, that is β-normal and η-long, it is non-trivial to prove $S \to S$ for non-atomic S, or to compose proofs of $S_1 \to S_2$ and $S_2 \to S_3$. The Identity and Substitution principles ensure that our type theory makes logical sense by demonstrating the reflexivity and transitivity of entailment. Reflexivity is witnessed by η-expansion, while transitivity is witnessed by hereditary substitution.

The Identity principle effectively says that synthesizing (atomic) objects can be made to serve as checking (normal) objects. The Substitution principle dually says that checking objects may stand in for synthesizing assumptions, that is, variables.

3.3.1. *Substitution.* The goal of this section is to give a careful proof of the following substitution theorem.

Suppose $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$. Then:

(1) If
- $\vdash \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}}\ \mathsf{ctx}$, and
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash S \sqsubset A$, and
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$,

then
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$ and $\vdash \Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}}\ \mathsf{ctx}$, and
- $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ and $[N_0/x_0]^{\mathrm{a}}_{A_0}\, A = A'$ and $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash S' \sqsubset A'$, and
- $[N_0/x_0]^{\mathrm{n}}_{A_0}\, N = N'$ and $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash N' \Leftarrow S'$,

(2) If
- $\vdash \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}}\ \mathsf{ctx}$ and
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash R \Rightarrow S$,

then
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$ and $\vdash \Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}}\ \mathsf{ctx}$, and $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$, and either
  - $[N_0/x_0]^{\mathrm{rr}}_{A_0}\, R = R'$ and $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash R' \Rightarrow S'$, or
  - $[N_0/x_0]^{\mathrm{rn}}_{A_0}\, R = (N', \alpha')$ and $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash N' \Leftarrow S'$,

and similarly for other syntactic categories.

(**Theorem 3.19** below.)

To prove the substitution theorem, we require a lemma about how substitutions compose. The corresponding property for a ordinary non-hereditary substitution says that $[N_0/x_0]\,[N_2/x_2]\,N = [[N_0/x_0]\,N_2/x_2]\,[N_0/x_0]\,N$. For hereditary substitutions, the situation is analogous, but we must be clear about which substitution instances we must assume to be defined and which we may conclude to be defined: If the three "inner" substitutions are defined, then the two "outer" ones are also defined, and equal. Note that the composition lemma is something like a diamond property; the notation below is meant to suggest this connection.

Lemma 3.13 (Composition of Substitutions)**.** *Suppose* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2^{\backslash}$ *and* $x_2 \notin \mathrm{FV}(N_0)$. *Then:*

(1) *If* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N = N^{\backslash}$ *and* $[N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, N = N'$, *then for some* N^{v}, $[N_2^{\backslash}/x_2]^{\mathrm{n}}_{\alpha_2}\, N^{\backslash} = N^{\mathsf{v}}$ *and* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N' = N^{\mathsf{v}}$,

(2) *If* $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R^{\backslash}$ *and* $[N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R = R'$, *then for some* R^{v}, $[N_2^{\backslash}/x_2]^{\mathrm{rr}}_{\alpha_2}\, R^{\backslash} = R^{\mathsf{v}}$ *and* $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R' = R^{\mathsf{v}}$,

(3) *If* $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R^{\backslash}$ *and* $[N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, R = (N', \beta)$, *then for some* N^{v}, $[N_2^{\backslash}/x_2]^{\mathrm{rn}}_{\alpha_2}\, R^{\backslash} = (N^{\mathsf{v}}, \beta)$ *and* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N' = N^{\mathsf{v}}$,

(4) *If* $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R = (N^{\backslash}, \beta)$ *and* $[N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R = R'$, *then for some* N^{v}, $[N_2^{\backslash}/x_2]^{\mathrm{n}}_{\alpha_2}\, N^{\backslash} = N^{\mathsf{v}}$ *and* $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R' = (N^{\mathsf{v}}, \beta)$,

and similarly for other syntactic categories.

Proof (sketch). By lexicographic induction on the unordered pair of α_0 and α_2, and on the first substitution derivation in each clause. The cases for rule **subst-rn-**β in clauses 3 and 4 appeal to the induction hypothesis at a smaller type using Lemma 3.4. The case in clause 4 swaps the roles of α_0 and α_2, necessitating the unordered induction metric. □

We also require a simple lemma about substitution into subsorting derivations:

Lemma 3.14 (Substitution into Subsorting)**.** *If* $Q_1 \leq Q_2$ *and* $[N_0/x_0]^{\mathrm{q}}_{\alpha_0}\, Q_1 = Q_1'$ *and* $[N_0/x_0]^{\mathrm{q}}_{\alpha_0}\, Q_2 = Q_2'$*, then* $Q_1' \leq Q_2'$*.*

Proof. Straightforward induction using Theorem 3.2 (Functionality of Substitution), since the subsorting rules depend only on term equalities, and not on well-formedness. □

Next, we must state the substitution theorem in a form general enough to admit an inductive proof. Following previous work on canonical forms-based LF [WCPW02, HL07], we strengthen its statement to one that does not presuppose the well-formedness of the context or the classifying types, but instead merely presupposes that hereditary substitution is defined on them. We call this strengthened theorem "proto-substitution" and prove it in several parts. In order to capture the convention that we only sort-check well-typed terms, proto-substitution includes hypotheses about well-typedness of terms; these hypotheses use an erasure Γ^* that transforms an LFR context into an LF context.

$$\cdot^* = \cdot \qquad\qquad (\Gamma, x{::}S \sqsubset A)^* = \Gamma^*, x{:}A$$

The structure of the proof under this convention requires that we interleave the proof of the core LF proto-substitution theorem. Generally, reasoning related to core LF presuppositions is analogous to refinement-related reasoning and can be dealt with mostly orthogonally, but the presuppositions are necessary in certain cases.

Theorem 3.15 (Proto-Substitution, terms)**.**

(1) *If*
- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma_{\mathrm{L}}^* \vdash N_0 \Leftarrow A_0$*) , and*
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$ *(and* $\Gamma_{\mathrm{L}}^*, x_0{:}A_0, \Gamma_{\mathrm{R}}^* \vdash N \Leftarrow A$*) , and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma_{\mathrm{R}}'$ *, and*
- $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ *(and* $[N_0/x_0]^{\mathrm{a}}_{A_0}\, A = A'$*) ,*

then
- $[N_0/x_0]^{\mathrm{n}}_{A_0}\, N = N'$ *, and*
- $\Gamma_{\mathrm{L}}, \Gamma_{\mathrm{R}}' \vdash N' \Leftarrow S'$ *(and* $\Gamma_{\mathrm{L}}^*, (\Gamma_{\mathrm{R}}')^* \vdash N' \Leftarrow A'$*) .*

(2) *If*
- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma_{\mathrm{L}}^* \vdash N_0 \Leftarrow A_0$*) , and*
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash R \Rightarrow S$ *(and* $\Gamma_{\mathrm{L}}^*, x_0{:}A_0, \Gamma_{\mathrm{R}}^* \vdash R \Rightarrow A$*) , and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma_{\mathrm{R}}'$ *,*

then
- $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ *(and* $[N_0/x_0]^{\mathrm{a}}_{A_0}\, A = A'$ *), and*
- *either*
  - $[N_0/x_0]^{\mathrm{rr}}_{A_0}\, R = R'$ *and*
  - $\Gamma_{\mathrm{L}}, \Gamma_{\mathrm{R}}' \vdash R' \Rightarrow S'$ *(and* $\Gamma_{\mathrm{L}}^*, (\Gamma_{\mathrm{R}}')^* \vdash R' \Rightarrow A'$*),*

  or
  - $[N_0/x_0]^{\mathrm{rn}}_{A_0}\, R = (N', (A')^-)$ *and*
  - $\Gamma_{\mathrm{L}}, \Gamma_{\mathrm{R}}' \vdash N' \Leftarrow S'$ *(and* $\Gamma_{\mathrm{L}}^*, (\Gamma_{\mathrm{R}}')^* \vdash N' \Leftarrow A'$*) .*

Note: We tacitly assume the implicit signature Σ is well-formed. We do *not* tacitly assume that any of the contexts, sorts, or types are well-formed. We *do* tacitly assume that contexts respect the usual variable conventions in that bound variables are always fresh, both with respect to other variables bound in the same context and with respect to other free variables in terms outside the scope of the binding.

Proof (sketch). By lexicographic induction on $(A_0)^-$ and the derivation $\mathcal{D}$ hypothesizing $x_0{::}S_0 \sqsubset A_0$.

The most involved case is that for application $R_1\ N_2$. When $\mathrm{head}(R_1) = x_0$ hereditary substitution carries out a β-reduction, and the proof invokes the induction hypothesis at a smaller type but not a subderivation. This case also requires Lemma 3.13 (Composition): since function sorts are dependent, the typing rule for application carries out a substitution, and we need to compose this substitution with the $[N_0/x_0]^{\mathrm{s}}_{\alpha_0}$ substitution.

In the case where we check a term at sort $\top$, we require the core LF assumptions in order to invoke the core LF proto-substitution theorem. □

Next, we can prove analogous proto-substitution theorems for sorts/types and for classes/kinds.

Theorem 3.16 (Proto-Substitution, sorts and types)**.**

(1) *If*
- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma^*_{\mathrm{L}} \vdash N_0 \Leftarrow A_0$*)* ,
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash S \sqsubset A$ *(and* $\Gamma^*_{\mathrm{L}}, x_0{:}A_0, \Gamma^*_{\mathrm{R}} \vdash A \Leftarrow \mathsf{type}$*)* , *and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$,

then
- $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ *(and* $[N_0/x_0]^{\mathrm{a}}_{A_0}\, A = A'$*)* , *and*
- $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash S' \sqsubset A'$, *(and* $\Gamma^*_{\mathrm{L}}, (\Gamma'_{\mathrm{R}})^* \vdash A' \Leftarrow \mathsf{type}$*)* .

(2) *If*
- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma^*_{\mathrm{L}} \vdash N_0 \Leftarrow A_0$*)* ,
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash Q \sqsubset P \Rightarrow L$ *(and* $\Gamma \vdash P \Rightarrow K$*)* , *and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$,

then
- $[N_0/x_0]^{\mathrm{q}}_{A_0}\, Q = Q'$ *(and* $[N_0/x_0]^{\mathrm{p}}_{A_0}\, P = P'$*)* , *and*
- $[N_0/x_0]^{\mathrm{l}}_{A_0}\, L = L'$ *(and* $[N_0/x_0]^{\mathrm{k}}_{A_0}\, K = K'$*)* , *and*
- $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash Q' \sqsubset P' \Rightarrow L'$ *(and* $\Gamma^*_{\mathrm{L}}, (\Gamma'_{\mathrm{R}})^* \vdash P' \Rightarrow K'$*)* .

Proof. By induction on the derivation hypothesizing $x_0{::}S_0 \sqsubset A_0$, using Theorem 3.15 (Proto-Substitution, terms). The reasoning is essentially the same as the reasoning for Theorem 3.15. □

Theorem 3.17 (Proto-Substitution, classes and kinds)**.**

If
- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma^*_{\mathrm{L}} \vdash N_0 \Leftarrow A_0$*)* ,
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash L \sqsubset K$ *(and* $\Gamma^*_{\mathrm{L}}, x_0{:}A_0, \Gamma^*_{\mathrm{R}} \vdash K \Leftarrow \mathsf{kind}$*)* , *and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$,

then
- $[N_0/x_0]^{\mathrm{l}}_{A_0}\, L = L'$ *(and* $[N_0/x_0]^{\mathrm{k}}_{A_0}\, K = K'$*)* , *and*
- $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash L' \sqsubset K'$, *(and* $\Gamma^*_{\mathrm{L}}, (\Gamma'_{\mathrm{R}})^* \vdash K' \Leftarrow \mathsf{kind}$*)* .

Proof. By induction on the derivation hypothesizing $x_0{::}S_0 \sqsubset A_0$, using Theorem 3.16 (Proto-Substitution, sorts and types). □

Then, we can finish proto-substitution by proving a proto-substitution theorem for contexts.

Theorem 3.18 (Proto-Substitution, contexts)**.**
If

- $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *(and* $\Gamma_{\mathrm{L}}^* \vdash N_0 \Leftarrow A_0$*) , and*
- $\vdash \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0\ \mathsf{ctx}$ *(and* $\vdash \Gamma_{\mathrm{L}}^*, x_0{:}A_0, \Gamma_{\mathrm{R}}^*\ \mathsf{ctx}$*) ,*

then

- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma_{\mathrm{R}}^{\backslash}$ *, and*
- $\vdash \Gamma_{\mathrm{L}}, \Gamma_{\mathrm{R}}^{\backslash}\ \mathsf{ctx}$ *(and* $\vdash \Gamma_{\mathrm{L}}^*, (\Gamma_{\mathrm{R}}^{\backslash})^*\ \mathsf{ctx}$*) .*

Proof. Straightforward induction on Γ_{R}. □

Finally, we have enough obtain a proof of the desired substitution theorem.

Theorem 3.19 (Substitution)**.** *Suppose* $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ *. Then:*

(1) *If*
- $\vdash \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}}\ \mathsf{ctx}$ *, and*
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash S \sqsubset A$ *, and*
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$ *,*

then
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$ *and* $\vdash \Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}}\ \mathsf{ctx}$ *, and*
- $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ *and* $[N_0/x_0]^{\mathrm{a}}_{A_0}\, A = A'$ *and* $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash S' \sqsubset A'$ *, and*
- $[N_0/x_0]^{\mathrm{n}}_{A_0}\, N = N'$ *and* $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash N' \Leftarrow S'$ *,*

(2) *If*
- $\vdash \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}}\ \mathsf{ctx}$ *and*
- $\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash R \Rightarrow S$ *,*

then
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma'_{\mathrm{R}}$ *and* $\vdash \Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}}\ \mathsf{ctx}$ *, and* $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S'$ *, and either*
  - $[N_0/x_0]^{\mathrm{rr}}_{A_0}\, R = R'$ *and* $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash R' \Rightarrow S'$ *, or*
  - $[N_0/x_0]^{\mathrm{rn}}_{A_0}\, R = (N', \alpha')$ *and* $\Gamma_{\mathrm{L}}, \Gamma'_{\mathrm{R}} \vdash N' \Leftarrow S'$ *,*

and similarly for other syntactic categories.

Proof. Straightforward corollary of Proto-Substitution Theorems 3.15, 3.16, 3.17, and 3.18. □

Having proven substitution, we henceforth tacitly assume that all subjects of a judgment are sufficiently well-formed for the judgment to make sense. In particular, we assume that all contexts are well-formed, and whenever we assume $\Gamma \vdash N \Leftarrow S$, we assume that for some well-formed type A, we have $\Gamma \vdash S \sqsubset A$ and $\Gamma \vdash N \Leftarrow A$. These assumptions embody our refinement restriction: we only sort-check a term if it is already well-typed and even then only at sorts that refine its type.

Similarly, whenever we assume $\Gamma \vdash S \sqsubset A$, we tacitly assume that $\Gamma \vdash A \Leftarrow \mathsf{type}$, and whenever we assume $\Gamma \vdash L \sqsubset K$, we tacitly assume that $\Gamma \vdash K \Leftarrow \mathsf{kind}$.

3.3.2. *Identity.* Just as we needed a composition lemma to prove the substitution theorem, in order to prove the identity theorem we need a lemma about how η-expansion commutes with substitution.[6]

[6]The categorically-minded reader might think of this as the right and left unit laws for $\circ$ while thinking of the composition lemma above as the associativity of $\circ$, where $\circ$ in the category represents substitution, as usual.

In stating this lemma, we require a judgment that predicts the simple type output of "rn" substitution. This judgment just computes the simple type as in "rn" substitution, but without computing anything having to do with substitution. Since it resembles a sort of "approximate typing judgment", we write it $x_0{:}\alpha_0 \vdash R : \alpha$. As with "rn" substitution, it is only defined when the head of R is x_0.

$$\frac{}{x_0{:}\alpha_0 \vdash x_0 : \alpha_0} \qquad \frac{x_0{:}\alpha_0 \vdash R : \alpha \to \beta}{x_0{:}\alpha_0 \vdash R\ N : \beta}$$

Lemma 3.20. *If* $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha')$ *and* $x_0{:}\alpha_0 \vdash R : \alpha$, *then* $\alpha' = \alpha$.

Proof. Straightforward induction. □

Lemma 3.21 (Commutativity of Substitution and η-expansion)**.** *Substitution commutes with η-expansion. In particular:*

(1) (a) *If* $[\eta_\alpha(x)/x]^{\mathrm{n}}_{\alpha} N = N'$, *then* $N = N'$,
(b) *If* $[\eta_\alpha(x)/x]^{\mathrm{rr}}_{\alpha} R = R'$, *then* $R = R'$,
(c) *If* $[\eta_\alpha(x)/x]^{\mathrm{rn}}_{\alpha} R = (N, \beta)$, *then* $\eta_\beta(R) = N$,
(2) *If* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} \eta_\alpha(R) = N'$, *then*
(a) *if* $\mathrm{head}(R) \neq x_0$, *then* $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0} R = R'$ *and* $\eta_\alpha(R') = N'$,
(b) *if* $\mathrm{head}(R) = x_0$ *and* $x_0{:}\alpha_0 \vdash R : \alpha$, *then* $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha)$,

and similarly for other syntactic categories.

Proof (sketch). By lexicographic induction on α and the given substitution derivation. The proofs of clauses 1a, 1b, and 1c analyze the substitution derivation, while the proofs of clauses 2a and 2b analyze the simple type α at which R is η-expanded. □

Note: By considering the variable being substituted for to be a bound variable subject to α-conversion[7], we can see that our commutativity theorem is equivalent to an apparently more general one where the η-expanded variable is not the same as the substituted-for variable. For example, in the case of clause (1a), we would have that if $[\eta_\alpha(x)/y]^{\mathrm{n}}_{\alpha} N = N'$, then $[x/y]\, N = N'$. We will freely make use of this fact in what follows when convenient.

Theorem 3.22 (Expansion)**.** *If* $\Gamma \vdash S \sqsubset A$ *and* $\Gamma \vdash R \Rightarrow S$, *then* $\Gamma \vdash \eta_A(R) \Leftarrow S$.

Proof (sketch). By induction on S. The $\Pi x{::}S_1 \sqsubset A_1.\, S_2$ case relies on Theorem 3.19 (Substitution) to show that $[\eta_{A_1}(x)/x]^{\mathrm{s}}_{A_1} S_2$ is defined and on Lemma 3.21 (Commutativity) to show that it is equal to S_2. □

Theorem 3.23 (Identity)**.** *If* $\Gamma \vdash S \sqsubset A$, *then* $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow S$.

Proof. Corollary of Theorem 3.22 (Expansion). □

[7] In other words, by reading $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N = N'$ as something like $\mathrm{subst}^{\mathrm{n}}_{\alpha_0}(N_0, x_0.\, N) = N'$, where x_0 is bound in N.

$$\boxed{S_1 \leq S_2}$$

$$\frac{}{S \leq S}\ (\textbf{refl}) \qquad \frac{S_1 \leq S_2 \qquad S_2 \leq S_3}{S_1 \leq S_3}\ (\textbf{trans}) \qquad \frac{S_2 \leq S_1 \qquad T_1 \leq T_2}{\Pi x{::}S_1.\, T_1 \leq \Pi x{::}S_2.\, T_2}\ (\textbf{S-}\Pi)$$

$$\frac{}{S \leq \top}\ (\top\textbf{-R}) \qquad \frac{T \leq S_1 \qquad T \leq S_2}{T \leq S_1 \wedge S_2}\ (\wedge\textbf{-R})$$

$$\frac{S_1 \leq T}{S_1 \wedge S_2 \leq T}\ (\wedge\textbf{-L}_1) \qquad \frac{S_2 \leq T}{S_1 \wedge S_2 \leq T}\ (\wedge\textbf{-L}_2)$$

$$\frac{}{\top \leq \Pi x{::}S.\, \top}\ (\top/\Pi\textbf{-dist}) \qquad \frac{}{(\Pi x{::}S.\, T_1) \wedge (\Pi x{::}S.\, T_2) \leq \Pi x{::}S.\, (T_1 \wedge T_2)}\ (\wedge/\Pi\textbf{-dist})$$

Figure 1: Derived rules for subsorting at higher sorts.

4. Subsorting at Higher Sorts

Our bidirectional typing discipline limits subsorting checks to a single rule, the **switch** rule when we switch modes from checking to synthesis. Since we insist on typing only canonical forms, this rule is limited to checking at atomic sorts Q, and consequently, subsorting need only be defined on atomic sorts. These observations naturally lead one to ask, what is the status of higher-sort subsorting in LFR? How do our intuitions about things like structural rules, variance, and distributivity—in particular, the rules shown in Figure 1—fit into the LFR picture?

It turns out that despite not *explicitly* including subsorting at higher sorts, LFR *implicitly* includes an intrinsic notion of higher-sort subsorting through the η-expansion associated with canonical forms. The simplest way of formulating this intrinsic notion is as a variant of the identity principle: S is taken to be a subsort of T if $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$. This notion is equivalent to a number of other alternate formulations, including a subsumption-based formulation and a substitution-based formulation.

Theorem 4.1 (Alternate Formulations of Subsorting)**.** *Suppose that for some* Γ_0, $\Gamma_0 \vdash S_1 \sqsubset A$ *and* $\Gamma_0 \vdash S_2 \sqsubset A$, *and define:*

(1) $S_1 \leq_1 S_2 \stackrel{def}{=}$ *for all* Γ *and* R: *if* $\Gamma \vdash R \Rightarrow S_1$, *then* $\Gamma \vdash \eta_A(R) \Leftarrow S_2$.

(2) $S_1 \leq_2 S_2 \stackrel{def}{=}$ *for all* Γ: $\Gamma, x{::}S_1 \sqsubset A \vdash \eta_A(x) \Leftarrow S_2$.

(3) $S_1 \leq_3 S_2 \stackrel{def}{=}$ *for all* Γ *and* N: *if* $\Gamma \vdash N \Leftarrow S_1$, *then* $\Gamma \vdash N \Leftarrow S_2$.

(4) $S_1 \leq_4 S_2 \stackrel{def}{=}$ *for all* Γ_{L}, Γ_{R}, N, *and* S: *if* $\Gamma_{\mathrm{L}}, x{::}S_2 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$ *then* $\Gamma_{\mathrm{L}}, x{::}S_1 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$

(5) $S_1 \leq_5 S_2 \stackrel{def}{=}$ *for all* Γ_{L}, Γ_{R}, N, S, *and* N_1: *if* $\Gamma_{\mathrm{L}}, x{::}S_2 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$ *and* $\Gamma_{\mathrm{L}} \vdash N_1 \Leftarrow S_1$, *then* $\Gamma_{\mathrm{L}}, [N_1/x]^{\gamma}_A\, \Gamma_{\mathrm{R}} \vdash [N_1/x]^{\mathrm{n}}_A\, N \Leftarrow [N_1/x]^{\mathrm{s}}_A\, S$.

Then, $S_1 \leq_1 S_2 \iff S_1 \leq_2 S_2 \iff \cdots \iff S_1 \leq_5 S_2$.

Proof. Using the identity and substitution principles along with Lemma 3.21, the commutativity of substitution with η-expansion.

1 $\Longrightarrow$ **2:** By rule, $\Gamma, x{::}S_1 \sqsubset A \vdash x \Rightarrow S_1$. By 1, $\Gamma, x{::}S_1 \sqsubset A \vdash \eta_A(x) \Leftarrow S_2$.

2 $\Longrightarrow$ **3:** Suppose $\Gamma \vdash N \Leftarrow S_1$. By 2, $\Gamma, x{::}S_1 \sqsubset A \vdash \eta_A(x) \Leftarrow S_2$. By Theorem 3.19 (Substitution), $\Gamma \vdash [N/x]^{\mathrm{n}}_A\, \eta_A(x) \Leftarrow S_2$. By Lemma 3.21 (Commutativity), $\Gamma \vdash N \Leftarrow S_2$.

3 $\Longrightarrow$ **4:** Suppose $\Gamma_{\mathrm{L}}, x{::}S_2 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$. By weakening, $\Gamma_{\mathrm{L}}, y{::}S_1 \sqsubset A, x{::}S_2 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$. By Theorem 3.23 (Identity), $\Gamma_{\mathrm{L}}, y{::}S_1 \sqsubset A \vdash \eta_A(y) \Leftarrow S_1$. By 3, $\Gamma_{\mathrm{L}}, y{::}S_1 \sqsubset A \vdash \eta_A(y) \Leftarrow S_2$. By Theorem 3.19 (Substitution), $\Gamma_{\mathrm{L}}, y{::}S_1 \sqsubset A, [\eta_A(y)/x]^{\gamma}_A\, \Gamma_{\mathrm{R}} \vdash [\eta_A(y)/x]^{\mathrm{n}}_A\, N \Leftarrow [\eta_A(y)/x]^{\mathrm{s}}_A\, S$. By Lemma 3.21 (Commutativity) and α-conversion, $\Gamma_{\mathrm{L}}, x{::}S_1 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$.

4 $\Longrightarrow$ **5:** Suppose $\Gamma_{\mathrm{L}}, x{::}S_2 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$ and $\Gamma_{\mathrm{L}} \vdash N_1 \Leftarrow S_1$. By 4, $\Gamma_{\mathrm{L}}, x{::}S_1 \sqsubset A, \Gamma_{\mathrm{R}} \vdash N \Leftarrow S$. By Theorem 3.19 (Substitution), $\Gamma_{\mathrm{L}}, [N_1/x]^{\gamma}_A\, \Gamma_{\mathrm{R}} \vdash [N_1/x]^{\mathrm{n}}_A\, N \Leftarrow [N_1/x]^{\mathrm{s}}_A\, S$.

5 $\Longrightarrow$ **1:** Suppose $\Gamma \vdash R \Rightarrow S_1$. By Theorem 3.22 (Expansion), $\Gamma \vdash \eta_A(R) \Leftarrow S_1$. By Theorem 3.23 (Identity), $\Gamma, x{::}S_2 \sqsubset A \vdash \eta_A(x) \Leftarrow S_2$. By 5, $\Gamma \vdash [\eta_A(R)/x]^{\mathrm{n}}_A\, \eta_A(x) \Leftarrow S_2$. By Lemma 3.21 (Commutativity), $\Gamma \vdash \eta_A(R) \Leftarrow S_2$. □

If we take "subsorting as η-expansion" to be our *model* of subsorting, we can show the "usual" presentation in Figure 1 to be both sound and complete with respect to this model. In other words, subsorting as η-expansion *really is* subsorting (soundness), and it is *no more than* subsorting (completeness). Alternatively, we can say that completeness demonstrates that there are no subsorting rules missing from the usual declarative presentation: Figure 1 accounts for everything covered intrinsically by η-expansion. By the end of this section, we will have shown both theorems: if $S \leq T$, then $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$, and vice versa.

Soundness is a straightforward inductive argument.

Theorem 4.2 (Soundness of Declarative Subsorting)**.** *If* $S \leq T$*, then* $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$.

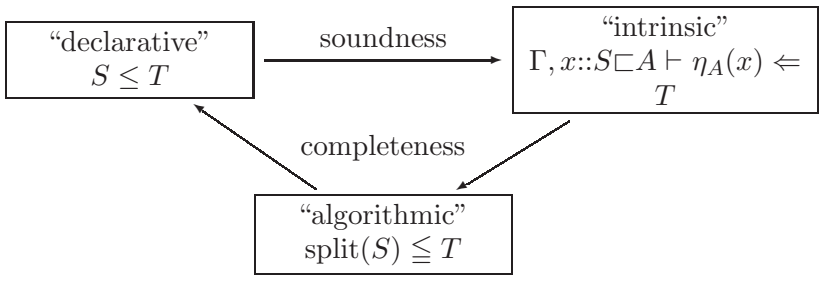
Proof. By induction on the derivation of $S \leq T$. The alternate formulations given by Theorem 4.1 are useful in many cases. □

The proof of completeness is considerably more intricate. We demonstrate completeness via a detour through an algorithmic subsorting system very similar to the algorithmic typing system from Section 3.2, with judgments $\Delta \leqq S$ and $\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$. To show completeness, we show that intrinsic subsorting implies algorithmic subsorting and that algorithmic subsorting implies declarative subsorting; the composition of these theorems is our desired completeness result.

If $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$, then $\mathrm{split}(S) \leqq T$. (**Theorem 4.15** below.)

If $\mathrm{split}(S) \leqq T$, then $S \leq T$. (**Theorem 4.7** below.)

The following schematic representation of soundness and completeness may help the reader to understand the key theorems.



$$\boxed{\Delta \leqq S}$$

$$\frac{}{\Delta \leqq \top} \qquad \frac{\Delta \leqq S_1 \qquad \Delta \leqq S_2}{\Delta \leqq S_1 \wedge S_2} \qquad \frac{Q' \in \Delta \qquad Q' \leq Q}{\Delta \leqq Q}$$

$$\frac{\Delta \mathbin{@} x{::}\mathrm{split}(S_1) \sqsubset A_1 = \Delta_2 \qquad \Delta_2 \leqq S_2}{\Delta \leqq \Pi x{::}S_1 \sqsubset A_1 .\, S_2}$$

$$\boxed{\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2}$$

$$\frac{}{\cdot \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \cdot} \qquad \frac{\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2 \qquad \Delta_1 \leqq S_1 \qquad [\eta_{A_1}(x)/y]^{\mathrm{n}}_{A_1}\, S_2 = S_2'}{(\Delta, \Pi y{::}S_1 \sqsubset A_1 .\, S_2) \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2, \mathrm{split}(S_2')}$$

$$\frac{\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2 \qquad \Delta_1 \not\leqq S_1}{(\Delta, \Pi y{::}S_1 \sqsubset A_1 .\, S_2) \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2}$$

$$\frac{\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2 \qquad \nexists S_2'.\, [\eta_{A_1}(x)/y]^{\mathrm{s}}_{A_1}\, S_2 = S_2'}{(\Delta, \Pi y{::}S_1 \sqsubset A_1 .\, S_2) \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2} \qquad \frac{\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2}{(\Delta, Q) \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2}$$

Figure 2: Algorithmic subsorting.

As mentioned above, the algorithmic subsorting system system is characterized by two judgments: $\Delta \leqq S$ and $\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$; rules defining them are shown in Figure 2. As in Section 3.2, Δ represents an intersection-free list of sorts. The interpretation of the judgment $\Delta \leqq S$, made precise below, is roughly that the intersection of all the sorts in Δ is a subsort of the sort S.

The rule for checking whether Δ is a subsort of a function type makes use of the application judgment $\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$ to extract all of the applicable function codomains from the list Δ. As in Section 3.2, care is taken to ensure that this latter judgment is defined even in seemingly "impossible" scenarios that well-formedness preconditions would rule out, like Δ containing atomic sorts or hereditary substitution being undefined.

Our first task is to demonstrate that the algorithm has the interpretation alluded to above. To that end, we define an operator $\bigwedge(-)$ that transforms a list Δ into a sort S by "folding" $\wedge$ over Δ with unit $\top$.

$$\bigwedge(\cdot) = \top \qquad\qquad \bigwedge(\Delta, S) = \bigwedge(\Delta) \wedge S$$

Now our goal is to demonstrate that if the algorithm says $\Delta \leqq S$, then declaratively $\bigwedge(\Delta) \leq S$. First, we prove some useful properties of the $\bigwedge(-)$ operator.

Lemma 4.3. $\bigwedge(\Delta_1) \wedge \bigwedge(\Delta_2) \leq \bigwedge(\Delta_1, \Delta_2)$

Proof. Straightforward induction on Δ_2. □

Lemma 4.4. $S \leq \bigwedge(\mathrm{split}(S))$.

Proof. Straightforward induction on S. □

Lemma 4.5. *If* $Q' \in \Delta$ *and* $Q' \leq Q$, *then* $\bigwedge(\Delta) \leq Q$.

Proof. Straightforward induction on Δ. □

$$\frac{S_1 \leq T_1 \qquad S_2 \leq T_2}{S_1 \wedge S_2 \leq T_1 \wedge T_2}\ (\textbf{S-}\wedge) \qquad\qquad \frac{}{S_1 \wedge (S_2 \wedge S_3) \leq (S_1 \wedge S_2) \wedge S_3}\ (\wedge\textbf{-assoc})$$

$$\frac{S \leq \Pi x{::}T_1.\, T_2 \qquad T_1 \leq S_1}{S \wedge \Pi x{::}S_1.\, S_2 \leq \Pi x{::}T_1.\, (T_2 \wedge S_2)}\ (\wedge/\Pi\textbf{-dist}')$$

Figure 3: Useful rules derivable from those in Figure 1.

Theorem 4.6 (Generalized Algorithmic ⇒ Declarative)**.**

(1) *If* $\mathcal{D} :: \Delta \leqq T$, *then* $\bigwedge(\Delta) \leq T$.
(2) *If* $\mathcal{D} :: \Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$, *then* $\bigwedge(\Delta) \leq \Pi x{::}\bigwedge(\Delta_1) \sqsubset A_1.\ \bigwedge(\Delta_2)$.

Proof (sketch). By induction on $\mathcal{D}$, using Lemmas 4.3, 4.4, and 4.5. The derivable rules from Figure 3 come in handy in the proof of clause 2. □

Theorem 4.6 is sufficient to prove that algorithmic subsorting implies declarative subsorting.

Theorem 4.7 (Algorithmic ⇒ Declarative)**.** *If* $\text{split}(S) \leqq T$, *then* $S \leq T$.

Proof. Suppose $\text{split}(S) \leqq T$. Then,

$\bigwedge(\text{split}(S)) \leq T$	By Theorem 4.6.
$S \leq \bigwedge(\text{split}(S))$	By Lemma 4.4.
$S \leq T$	By rule **trans**.

□

Now it remains only to show that intrinsic subsorting implies algorithmic. To do so, we require some lemmas. First, we extend our notion of a sort S refining a type A to an entire list of sorts Δ refining a type A in the obvious way.

$$\frac{}{\Gamma \vdash \cdot \sqsubset A} \qquad\qquad \frac{\Gamma \vdash \Delta \sqsubset A \qquad \Gamma \vdash S \sqsubset A}{\Gamma \vdash (\Delta, S) \sqsubset A}$$

This new notion has the following important properties.

Lemma 4.8. *If* $\Gamma \vdash \Delta_1 \sqsubset A$ *and* $\Gamma \vdash \Delta_2 \sqsubset A$, *then* $\Gamma \vdash \Delta_1, \Delta_2 \sqsubset A$.

Proof. Straightforward induction on Δ_2. □

Lemma 4.9. *If* $\Gamma \vdash S \sqsubset A$, *then* $\Gamma \vdash \text{split}(S) \sqsubset A$.

Proof. Straightforward induction on S. □

Lemma 4.10. *If* $\mathcal{D} :: \Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ *and* $\mathcal{E} :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2$ *and* $[N/x]^{\text{a}}_{A_1} A_2 = A_2'$, *then* $\Gamma \vdash \Delta_2 \sqsubset A_2'$.

Proof (sketch). By induction on $\mathcal{E}$, using Theorem 3.9 (Soundness of Algorithmic Typing) to appeal to Theorem 3.19 (Substitution), along with Lemmas 4.8 and 4.9. □

We will also require an analogue of subsumption for our algorithmic typing system, which relies on two lemmas about lists of sorts.

Lemma 4.11. *If* $\Gamma \vdash \Delta \sqsubset A$, *then for all* $S \in \Delta$, $\Gamma \vdash S \sqsubset A$.

Proof. Straightforward induction on Δ. □

Lemma 4.12. *If for all* $S \in \Delta$, $\Gamma \vdash N \Leftarrow S$, *then* $\Gamma \vdash N \Leftarrow \bigwedge(\Delta)$.

Proof. Straightforward induction on Δ. □

Theorem 4.13 (Algorithmic Subsumption)**.** *If* $\Gamma \vdash R \Rrightarrow \Delta$ *and* $\Gamma \vdash \Delta \sqsubset A$ *and* $\Delta \leqq S$, *then* $\Gamma \vdash \eta_A(R) \Leftarrow S$.

Proof. Straightforward deduction, using soundness and completeness of algorithmic typing.

$\forall S' \in \Delta. \Gamma \vdash R \Rightarrow S'$	By Theorem 3.9 (Soundness of Alg. Typing).
$\forall S' \in \Delta. \Gamma \vdash S' \sqsubset A$	By Lemma 4.11.
$\forall S' \in \Delta. \Gamma \vdash \eta_A(R) \Leftarrow S'$	By Theorem 3.22 (Expansion).
$\Gamma \vdash \eta_A(R) \Leftarrow \bigwedge(\Delta)$	By Lemma 4.12.
$\Delta \leqq S$	By assumption.
$\bigwedge(\Delta) \leq S$	By Theorem 4.6 (Generalized Alg. $\Rightarrow$ Decl.).
$\Gamma \vdash \eta_A(R) \Leftarrow S$	By Theorem 4.2 (Soundness of Decl. Subsorting) and Theorem 4.1 (Alternate Formulations of Subsorting).
$\Gamma \vdash \eta_A(R) \Leftarrow S$	By Theorem 3.11 (Completeness of Alg. Typing).

□

Now we can prove the following main theorem, which generalizes our desired "Intrinsic $\Rightarrow$ Algorithmic" theorem:

Theorem 4.14 (Generalized Intrinsic $\Rightarrow$ Algorithmic)**.**

(1) *If* $\Gamma \vdash R \Rrightarrow \Delta$ *and* $\mathcal{E} :: \Gamma \vdash \eta_A(R) \Leftarrow S$ *and* $\Gamma \vdash \Delta \sqsubset A$ *and* $\Gamma \vdash S \sqsubset A$, *then* $\Delta \leqq S$.

(2) *If* $\Gamma \vdash x \Rrightarrow \Delta_1$ *and* $\mathcal{E} :: \Gamma \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2$ *and* $\Gamma \vdash \Delta_1 \sqsubset A_1$ *and* $\Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$, *then* $\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$.

Proof (sketch). By induction on A, S, and $\mathcal{E}$.

Clause 1 is most easily proved by case analyzing the sort S and applying inversion to the derivation $\mathcal{E}$. The case when $S = \Pi x{::}S_1 \sqsubset A_1.\, S_2$ appeals to the induction hypothesis at an unrelated derivation but at a smaller type, and Lemmas 4.8 and 4.9 are used to satisfy the preconditions of the induction hypotheses.

Clause 2 is most easily proved by case analyzing the derivation $\mathcal{E}$. In one case, we require the contrapositive of Theorem 4.13 (Algorithmic Subsumption) to convert a derivation of $\Gamma \not\vdash \eta_{A_1}(x) \Leftarrow S_1$ into a derivation of $\Delta_1 \not\leqq S_1$.

□

Theorem 4.14 along with Theorem 3.11, the Completeness of Algorithmic Typing, gives us our desired result:

Theorem 4.15 (Intrinsic $\Rightarrow$ Algorithmic)**.** *If* $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$, *then* $\mathrm{split}(S) \leqq T$.

Proof. Suppose $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$. Then,

$\Gamma, x{::}S \sqsubset A \vdash x \Rrightarrow \mathrm{split}(S)$	By rule.
$\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$	By Theorem 3.11 (Completeness of Alg. Typing).
$\mathrm{split}(S) \leqq T$	By Theorem 4.14.

□

Finally, we have completeness as a simple corollary:

Theorem 4.16 (Completeness of Declarative Subsorting)**.** *If* $\Gamma, x{::}S \sqsubset A \vdash \eta_A(x) \Leftarrow T$, *then* $S \leq T$.

Proof. Corollary of Theorems 4.15 and 4.7. □

5. Proof Irrelevance

When constructive type theory is used as a foundation for verified functional programming, we notice that many parts of proofs are *computationally irrelevant*, that is, their structure does not affect the returned value we are interested in. The role of these proofs is only to guarantee that the returned value satisfies the desired specification. For example, from a proof of $\forall x{:}A.\, \exists y{:}B.\, C(x, y)$ we may choose to extract a function $f : A \to B$ such that $C(x, f(x))$ holds for every $x{:}A$, but ignore the proof that this is the case. The proof must be present, but its identity is irrelevant. Proof-checking in this scenario has to ascertain that such a proof is indeed not needed to compute the relevant result.

A similar issue arises when a type theory such as λ^Π is used as a logical framework. For example, assume we would like to have an adequate representation of prime numbers, that is, to have a bijection between prime numbers p and closed terms $M : \mathit{primenum}$. It is relatively easy to define a type family $\mathit{prime} : \mathit{nat} \to \mathsf{type}$ such that there exists a closed $M : \mathit{prime}\ N$ if and only if N is prime. Then $\mathit{primenum} = \Sigma n{:}\mathit{nat}.\, \mathit{prime}\ n$ is a candidate (with members $\langle N, M\rangle$), but it is not actually in bijective correspondence with prime numbers unless the proof M that a number is prime is always unique. Again, we need the existence of M, but would like to ignore its identity. This can be achieved with *subset types* [C+86, SS88] $\{x{:}\mathit{nat} \mid \mathit{prime}(x)\}$ whose members are just the prime numbers p, but if the restricting predicate is undecidable then type-checking would be undecidable, which is not acceptable for a logical framework.

For LF, we further note that Σ is not available as a type constructor, so we instead introduce a new type *primenum* with exactly one constructor, *primenum*/*i*:

$\mathit{primenum} : \mathsf{type}.$
$\mathit{primenum}/i : \Pi N{:}\mathit{nat}.\ \mathit{prime}\ N \div\!\!\to \mathit{primenum}.$

Here the second arrow $\div\!\!\to$ represents a function that ignores the identity of its argument. The inhabitants of *primenum*, all of the form $\mathit{primenum}/i\ N\ [M]$, are now in bijective correspondence with prime numbers since $\mathit{primenum}/i\ N\ [M] = \mathit{primenum}/i\ N\ [M']$ for all M and M'.

In the extension of LF with proof irrelevance [Pfe01a, RP08], or LFI, we have a new form of hypothesis $x \div A$ (x has type A, but the identity of x should be irrelevant). In

the non-dependent case (the only one important for the purposes of this paper), such an assumption is introduced by a λ-abstraction:

$$\frac{\Gamma, x \div A \vdash M \Leftarrow B}{\Gamma \vdash \lambda x.\, M \Leftarrow A \div B}\ .$$

We can use such variables only in places where their identity doesn't matter, e.g., in the second argument to the constructor *primenum/i* in the prime number example. More generally, we can only use it in arguments to constructor functions that do not care about the identity of their argument:

$$\frac{\Gamma \vdash R \Rightarrow A \div B \qquad \Gamma^{\oplus} \vdash N \Leftarrow A}{\Gamma \vdash R\ [N] \Rightarrow B}\ .$$

Here, $\Gamma^{\oplus}$ is the *promotion* operator which converts any assumption $x \div A$ to $x{:}A$, thereby making x usable in N. Note that there is no direct way to use an assumption $x \div A$.

The underlying definitional equality "=" (usually just α-conversion on canonical forms) is extended so that $R\ [N] = R'\ [N']$ if $R = R'$, no matter what N and N' are.

The substitution principle (shown here only in its simplest, non-dependent form) captures the proper typing as well as the irrelevance of assumptions $x \div A$:

Principle 5.1 (Irrelevant Substitution)**.** If $\Gamma, x \div A \vdash N \Leftarrow B$ and $\Gamma^{\oplus} \vdash M \Leftarrow A$ then $\Gamma \vdash [M/x]\, N \Leftarrow B$ and $[M/x]\, N = N$ (under definitional equality).

One typical use of proof irrelevance in type theory is to render the typechecking of subset types [C+86, SS88] decidable. A subset type $\{x{:}A \mid B(x)\}$ represents the set of terms of type A which also satisfy B; typechecking is undecidable because to determine if a term M has this type, you must search for a proof of $B(M)$. One might attempt to recover decidability by using a dependent sum $\Sigma x{:}A.\, B(x)$, representing the set of terms M of type A paired with proofs of $B(M)$; typechecking is decidable, since a proof of $B(M)$ is provided, but equality of terms is overly fine-grained: if there are two proofs of $B(M)$, the two pairs will be considered unequal. Using proof irrelevance, one can find a middle ground with the type $\Sigma x{:}A.\, [B(x)]$, where $[-]$ represents the proof irrelevance modality. Type checking is decidable for such terms, since a proof of the property B is always given, but the identity of that proof is ignored, so all pairs with the same first component will be considered equal.

Our situation with the subset interpretation is similar: we would like to represent proofs of sort-checking judgments without depending on the identities of those proofs. By carefully using proof irrelevance to hide the identities of sort-checking proofs, we are able to make a translation that is sound and complete, preserving the adequacy of representations.

6. Interpretation

6.1. **Overview.** We interpret LFR into LFI by representing sorts as predicates and derivations of sorting as proofs of those predicates. In this section, we endeavor to explain our general translation by way of examples of it in action. The translation is derivation-directed and compositional: for each judgment $\Gamma \vdash \mathcal{J}$, there is a corresponding judgment $\Gamma \vdash \mathcal{J} \rightsquigarrow X$ whose rules mimic the rules of $\Gamma \vdash \mathcal{J}$. The syntactic class of X and its precise interpretation vary from judgment to judgment. For reference, the various forms are listed in Table 2, but we will explain them in turn as they arise in our examples.

Judgment:	**Result:**
$\Gamma \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}_{\text{f}}(-)$	Type of proofs of the formation family
$K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_{\text{p}}(-,-)$	Kind of the predicate family
$K \overset{\leq}{\rightsquigarrow} \widehat{K}_{\text{s}}(-,-,-,-,-)$	Type of coercions between families of kind K
$\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}(-)$	Metafunction representing predicate
$\Gamma \vdash Q \sqsubset P \Rightarrow L \rightsquigarrow \widehat{Q}$	Proof that Q is well-formed
$\Gamma \vdash N \Leftarrow S \rightsquigarrow \widehat{N}$	Proof that N has sort S
$\Gamma \vdash R \Rightarrow S \rightsquigarrow \widehat{R}$	Proof that R has sort S
$\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow F(-,-)$	Metacoercion from proofs of Q_1 to proofs of Q_2
$Q_1 \leq Q_2 \rightsquigarrow \widehat{Q_1\text{-}Q_2}$	Coercion from proofs of Q_1 to proofs of Q_2
$\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}$	Translated context
$\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}$	Translated signature

Table 2: Judgments of the translation.

Recall our simplest example of refinement types: the natural numbers, where the even and odd numbers are isolated as refinements.

$nat : \mathsf{type}.$
$z : nat.$
$s : nat \to nat.$

$even \sqsubset nat.$
$odd \sqsubset nat.$
$z :: even.$
$s :: even \to odd \ \wedge \ odd \to even.$

As described in the introduction, our translation represents *even* and *odd* as *predicates* on natural numbers, and the refinement declarations for z and s become declarations for constants for constructing proofs of those predicates.

$even : nat \to \mathsf{type}.$
$odd : nat \to \mathsf{type}.$
$\widehat{z} \ : even\ z.$
$\widehat{s}_1 : \Pi x{:}nat.\ even\ x \to odd\ (s\ x).$
$\widehat{s}_2 : \Pi x{:}nat.\ odd\ x \to even\ (s\ x).$

Starting simple, the proof constructor declaration for $\widehat{z}$ can be read as an assertion that the constant z satisfies a certain predicate, namely that of being even.

In fact, every sort S will have a representation as a predicate, not just the base sorts like *even* and *odd*. Generally, a predicate is just a *type* with a hole for a term; conventionally, we write the predicate representation of S as a meta-level function $\widehat{S}(-)$, and we say that a term N satisfies such a predicate if the type $\widehat{S}(N)$ is inhabited. Predicates will be the output of the sort translation judgment, $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$, which mirrors the sort formation judgment, adding a translation as an output.

For example, the predicate corresponding to the sort $\mathit{even} \to \mathit{odd}$ is the meta-function $(\Pi x{:}\mathit{nat}.\ \mathit{even}\ x \to \mathit{odd}\ ((-)\ x))$, and we see this predicate applied to the successor constant s in the type of the proof constructor $\widehat{s}_1$. Thus the proof constructor declaration for $\widehat{s}_1$ can *also* be read as an assertion: the constant s satisfies the predicate that, when applied to an even natural number, it yields an odd one.

Our analysis suggests a general strategy for translating a refinement type declaration: translate its sort into a predicate, and yield a declaration of a proof constructor asserting that the predicate holds of the original constant.

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad c{:}A \in \Sigma \qquad \cdot \vdash_\Sigma S \sqsubset A \rightsquigarrow \widehat{S}}{\vdash \Sigma,\ c{::}S\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma},\ \widehat{c}{:}\widehat{S}(\eta_A(c))}$$

As a reflection of the fact that in general these predicates may be applied to arbitrary terms, not just atomic ones, we fully η-expand the constant before applying the predicate.

How do arrow sorts like $\mathit{even} \to \mathit{odd}$ translate in general? Recall that $S \to T$ is just shorthand for the dependent function sort $\Pi x{::}S.\, T$ when x does not occur in T. The general rule for translating dependent function sorts is:

$$\frac{\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S} \qquad \Gamma, x{::}S \sqsubset A \vdash T \sqsubset B \rightsquigarrow \widehat{T}}{\Gamma \vdash \Pi x{::}S \sqsubset A.\, T \sqsubset \Pi x{:}A.\, B \rightsquigarrow \boldsymbol{\lambda} N\mathbf{.}\ \Pi x{:}A.\, \Pi \widehat{x}{:}\widehat{S}(\eta_A(x)).\, \widehat{T}(N@x)}\ (\Pi\textbf{-F})$$

There are two points of note in this rule. First, writing predicates as types with holes becomes cumbersome, so we instead write metafunctions explicitly using meta-level abstraction, written as a bold $\boldsymbol{\lambda}$; we continue to write meta-level application using bold (parens). Second, since as we noted above, the term argument of a predicate is in general a canonical term, and canonical terms may not appear in application position, we appeal to an auxiliary judgment that applies a canonical term to an atomic one, $N@R$. It is defined by the single clause,

$$(\lambda x.\, N)@R = [R/x]\, N,$$

where the right-hand side is an ordinary non-hereditary substitution. Now we can read the translation output as the predicate of a term N which holds if there is a function from objects $x : A$ satisfying predicate $\widehat{S}$ to proofs that N applied to x satisfies predicate $\widehat{T}$.

But what about the fact that s only had one declaration in the original signature, but there are two proof constructor declarations asserting predicates that hold of it? For compositionality's sake, we would like to translate the single refinement declaration for s into a single proof constructor declaration, but one that can effectively serve the roles of both $\widehat{s}_1$ and $\widehat{s}_2$. To this end, we use a product type.

$$\begin{aligned}&\widehat{s} : (\Pi x{:}\mathit{nat}.\ \mathit{even}\ x \to \mathit{odd}\ (s\ x))\\ &\quad \times (\Pi x{:}\mathit{nat}.\ \mathit{odd}\ x \to \mathit{even}\ (s\ x)).\end{aligned}$$

Now $\pi_i\, \widehat{s}$ may be used anywhere $\widehat{s}_i$ was used before. Generally, an intersection sort will translate to a conjunction of predicates, represented as a type-theoretic product. Similarly,

the nullary intersection $\top$ will translate to a unit type.[8]

$$\frac{\Gamma \vdash S_1 \sqsubset A \rightsquigarrow \widehat{S_1} \qquad \Gamma \vdash S_2 \sqsubset A \rightsquigarrow \widehat{S_2}}{\Gamma \vdash S_1 \wedge S_2 \sqsubset A \rightsquigarrow \boldsymbol{\lambda} N.\ \widehat{S_1}(N) \times \widehat{S_2}(N)}\ (\wedge\textbf{-F}) \qquad \frac{}{\Gamma \vdash \top \sqsubset A \rightsquigarrow \boldsymbol{\lambda} N.\ 1}\ (\top\textbf{-F})$$

What kinds of proofs inhabit these predicates? Such proofs are the output of the term translation judgment $\Gamma \vdash N \Leftarrow S \rightsquigarrow \widehat{N}$, which mirrors the sort checking judgment, adding a translation as an output. Generally, a derivation that a term N has sort S will translate to a proof $\widehat{N}$ that the predicate $\widehat{S}$ holds of N (where $\widehat{S}$ is as usual the interpretation of S as a predicate), or symbolically, if $S \sqsubset A \rightsquigarrow \widehat{S}$ and $N \Leftarrow S \rightsquigarrow \widehat{N}$, then $\widehat{N} \Leftarrow \widehat{S}(N)$—ignoring for a moment the question of what happens to the contexts. This expectation begins to hint at the soundness theorem we will demonstrate below, but for now we will use it just to guide our intuitions.

For example, since an intersection sort is represented by a product of predicates, we should expect that a term judged to have an intersection sort should translate to a proof of a product, or a pair. Similarly, since the sort $\top$ translates to a trivially true unit predicate, a term judged to have sort $\top$ should translate to a trivial unit element.

$$\frac{\Gamma \vdash N \Leftarrow S_1 \rightsquigarrow \widehat{N_1} \qquad \Gamma \vdash N \Leftarrow S_2 \rightsquigarrow \widehat{N_2}}{\Gamma \vdash N \Leftarrow S_1 \wedge S_2 \rightsquigarrow \langle \widehat{N_1}, \widehat{N_2} \rangle}\ (\wedge\textbf{-I}) \qquad \frac{}{\Gamma \vdash N \Leftarrow \top \rightsquigarrow \langle\rangle}\ (\top\textbf{-I})$$

Intuitively, knowing that a term has an intersection sort $S_1 \wedge S_2$ gives us two pieces of information about it, while knowing that a term has sort $\top$ tells us nothing new. This aspect of our translation is similar in spirit to Liquori and Ronchi Della Rocca's $\Lambda^{\mathsf{t}}_{\wedge}$ [LRDR07], a Church-style type system for intersections in which derivations are explicitly represented as proofs and intersections as products, though in their setting the proofs are viewed as part of a program rather than the output of a translation.

We can similarly intuit the appropriate proof for an implication predicate by examining the rule for translating $\Pi x{::}S.\, T$ above. We start from the sort-checking rule Π**-I**, which shows that a term $\lambda x.\, N$ has sort $\Pi x{::}S.\, T$. To prove that the corresponding Π predicate holds of $\lambda x.\, N$, we will have to produce a function taking an object x of type A and a proof that x satisfies $\widehat{S}$ and yielding a proof that $(\lambda x.\, N)@x = [x/x]\, N = N$ satisfies $\widehat{T}$. This is easily done: the translation of the body N is precisely the proof we require about N, and we wrap this in two λ-abstractions to get a proof of the Π predicate.

$$\frac{\Gamma, x{::}S \sqsubset A \vdash N \Leftarrow T \rightsquigarrow \widehat{N}}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x{::}S \sqsubset A.\, T \rightsquigarrow \lambda x.\, \lambda \widehat{x}.\, \widehat{N}}\ (\Pi\textbf{-I})$$

Careful examination of the Π**-I** rule reveals a subtlety: it is clear from our understanding of the sort-checking part of the rule that the free variables of N and T may include x, but we seem to have indicated by our λ-abstraction that the proof $\widehat{N}$ may depend not only on the variable x, but also on a variable $\widehat{x}$. Where did this second variable come from?

The answer—as hinted above—is that we have not yet specified with respect to what context the translation of a term is to be interpreted. This context should in fact be the translation of the context Γ associated with the original term N, and by convention we write

[8] Strictly speaking, this means our translation targets an extension of LFI with product and unit types. Such an extension is orthogonal to the addition of proof irrelevance, and has been studied by many people over the years, including Schürmann [Sch03] and Sarkar [Sar09]. Alternatively, products may be eliminated after translation by a simple currying transformation, but that is beyond the scope of this article.

$$\boxed{\Gamma \vdash_\Sigma R^+ \Rightarrow S^- \rightsquigarrow \widehat{R}^-}$$

$$\frac{c{::}S \in \Sigma}{\Gamma \vdash c \Rightarrow S \rightsquigarrow \widehat{c}}\ (\textbf{const}) \qquad \frac{x{::}S{\sqsubset}A \in \Gamma}{\Gamma \vdash x \Rightarrow S \rightsquigarrow \widehat{x}}\ (\textbf{var})$$

$$\frac{\Gamma \vdash R_1 \Rightarrow \Pi x{::}S_2{\sqsubset}A_2.\, S \rightsquigarrow \widehat{R_1} \qquad \Gamma \vdash N_2 \Leftarrow S_2 \rightsquigarrow \widehat{N_2} \qquad [N_2/x]^{\mathrm{s}}_{A_2}\, S = S'}{\Gamma \vdash R_1\ N_2 \Rightarrow S' \rightsquigarrow \widehat{R_1}\ N_2\ \widehat{N_2}}\ (\Pi\textbf{-E})$$

$$\frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2 \rightsquigarrow \widehat{R}}{\Gamma \vdash R \Rightarrow S_1 \rightsquigarrow \pi_1\, \widehat{R}}\ (\wedge\textbf{-E}_1) \qquad \frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2 \rightsquigarrow \widehat{R}}{\Gamma \vdash R \Rightarrow S_2 \rightsquigarrow \pi_2\, \widehat{R}}\ (\wedge\textbf{-E}_2)$$

Figure 4: Translation rules for atomic term sort synthesis

it as $\widehat{\Gamma}$. The judgment translating contexts is an annotated version of the context-formation judgment, written $\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}$.

$$\frac{}{\vdash \cdot\ \mathsf{ctx} \rightsquigarrow \cdot} \qquad \frac{\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma} \qquad \Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}}{\vdash \Gamma, x{::}S{\sqsubset}A\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}, x{:}A, \widehat{x}{:}\widehat{S}(\eta_A(x))}$$

The second rule is quite similar to the translation rule we have seen for signature declarations $c{:}A$: each declaration $x{::}S{\sqsubset}A$ splits into a typing declaration $x{:}A$ and a proof declaration $\widehat{x}{:}\widehat{S}(\eta_A(x))$. Now it is easily seen why the proof $\widehat{N}$ in the translation rule Π-**I** may depend on $\widehat{x}$: our soundness criterion will tell us that $\widehat{\Gamma}, x{:}A, \widehat{x}{:}\widehat{S}(\eta_A(x)) \vdash \widehat{N} \Leftarrow \widehat{T}(N)$.

There is just one sort checking rule remaining: the **switch** rule for checking an atomic term at a base sort. This rule appeals to subsorting, so we postpone discussion of it until we discuss the translation of subsorting judgments in Section 6.3. For now, the reader may think of the rule as simply returning the result of the sort synthesis translation judgment, $\Gamma \vdash R \Rightarrow S \rightsquigarrow \widehat{R}$. At the base cases, this judgment returns the hatted proof constants $\widehat{c}$ and variables $\widehat{x}$ we have seen in the translations of signature declarations and contexts. The other rules correspond to elimination forms, and they follow straightforwardly by the same intuitions we used to derive the introduction rules in the sort checking translation. All the rules for this judgment are shown in Figure 4.

There is also just one sort formation rule remaining: the rule for translating base sorts Q. Although this translation seems straightforward in the case of simple sorts like *even* and *odd*, it is rather subtle when it comes to dependent sort families due to a problem of coherence. To explain, we return to another early example, the doubling relation on natural numbers.

6.2. **Dependent Base Sorts.** Recall the double relation defined as a type family in LF:

double : *nat* → *nat* → type.
dbl/z : *double z z*.
dbl/s : Π*N*:*nat*. Π*N2*:*nat*. *double N N2* → *double* (*s N*) (*s* (*s N2*)).

As we saw earlier, we can use LFR *refinement kinds*, or *classes*, to express and enforce the property that the second subject of any doubling relation is always even, no matter what

properties hold of the first subject. To do so we define a sort *double** which is isomorphic to *double*, but has a more precise class.[9]

$double^* \sqsubset double :: \top \to even \to \mathsf{sort}.$
$dbl/z :: double^*\ z\ z.$
$dbl/s :: \Pi N{::}\top.\ \Pi N2{::}even.\ double^*\ N\ N2 \to double^*\ (s\ N)\ (s\ (s\ N2)).$

Successfully sort-checking the declarations for dbl/z and dbl/s demonstrates that whenever $double^*\ M\ N$ is inhabited, the second argument, N, is even.

There is a crucial difference between refinements like *even* or *odd* and refinements like *double**: while *even* and *odd* denote particular subsets of the natural numbers, the inhabitants of the refinement $double^*\ M\ N$ are identical to those of the ordinary type $double\ M\ N$. What is important is not whether a particular instance $double^*\ M\ N$ is inhabited, but rather whether it is *well-formed at all*.

For this reason, we separate the *formation* of a dependent refinement type family from its *inhabitation*. Simple sorts like *even* and *odd* are always well-formed, but we would like a way to explicitly represent the formation of an indexed sort like $double^*\ M\ N$. Therefore, we translate *double** into two parts: a *formation family*, written $\widehat{double}^*$, and a *predicate family*, written using the original name of the sort, *double**.

There are two declarations involving the formation family. First, the declaration of the formation family itself:

$\widehat{double}^* : nat \to nat \to \mathsf{type}.$

The formation family has the same kind as the original refined type. Intuitively, the formation family $\widehat{double}^*\ M\ N$ should be inhabited whenever the sort $double^*\ M\ N$ would have been a well-formed sort pre-translation. For example, $\widehat{double}^*\ z\ z$ will be inhabited, since $double^*\ z\ z$ was a well-formed sort.

Next, we have a constructor for the formation family:

$\widehat{double}^*/i : \Pi x{:}nat.\ \Pi y{:}nat.\ even\ y \to \widehat{double}^*\ x\ y.$

The constructor takes all the arguments to *double** along with evidence that they have the appropriate sorts and yields a member of the formation family, i.e. a proof that *double** applied to those arguments was well-formed pre-translation. For example, $\widehat{double}^*/i\ z\ z\ \widehat{z}$ is a proof that $double^*\ z\ z$ was well-formed, since it contains the necessary evidence: a proof that the second argument z is *even*.

Finally, we have a declaration for the predicate family itself:

$double^* : \Pi x{:}nat.\ \Pi y{:}nat.\ \widehat{double}^*\ x\ y \div double\ x\ y \to \mathsf{type}.$

For any M and N, the predicate family will be inhabited by proofs that derivations of $double\ M\ N$ have the refinement $double^*\ M\ N$, provided that $double^*\ M\ N$ is well-formed in the first place. In our doubling example, all derivations of $double\ M\ N$ satisfy the refinement $double^*\ M\ N$, so the predicate family will have one inhabitant for each of them. As before, these inhabitants come from the translation of the refinement declarations for dbl/z and dbl/s. Writing arguments in irrelevant position in [*square brackets*], we get:

[9]Earlier, we used the name *double* for both the type family and the sort family refining it, but in what follows it will be important to distinguish the two.

$$\begin{aligned}
&\widehat{dbl/z} : double^*\ z\ z \\
&\qquad [\ \widehat{double}^*/i\ z\ z\ \widehat{z}\ ] \\
&\qquad dbl/z. \\
&\widehat{dbl/s} : \Pi N{:}nat.\ \Pi N2{:}nat.\ \Pi \widehat{N2}{:}even\ N2.\ \Pi D{:}double\ N\ N2. \\
&\qquad double^*\ N\ N2\ [\ \widehat{double}^*/i\ N\ N2\ \widehat{N2}\ ]\ D \\
&\quad \to double^*\ (s\ N)\ (s\ (s\ N2)) \\
&\qquad [\ \widehat{double}^*/i\ (s\ N)\ (s\ (s\ N2))\ (\widehat{s}_2\ (s\ N2)\ (\widehat{s}_1\ N2\ \widehat{N2}))\ ] \\
&\qquad (dbl/s\ N\ N2\ D).
\end{aligned}$$

As is evident even from this short and abbreviated example, the interpretation leads to a significant blowup in the size and complexity of a signature, underscoring the importance of a primitive understanding of refinement types.

Note that in the declaration of the predicate family *double**, the proof of well-formedness is made irrelevant using a proof-irrelevant function space $A \div B$, representing functions from A to B that are insensitive to the identity of their argument. Using irrelevance ensures that a given sort has a unique translation, up to equivalence. We elaborate on this below.

Generalizing from the above example, a sort declaration translates into three declarations: one for the *formation family*, one for the *proof constructor* for the formation family, and one for the *predicate family*.

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad a{:}K \in \Sigma \qquad \cdot \vdash_\Sigma L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}_\mathrm{f} \qquad K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_\mathrm{p}}{\vdash \Sigma, s{\sqsubset}a{::}L\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma},\ \widehat{s}{:}K,\ \widehat{s}/i{:}\widehat{L}_\mathrm{f}(\widehat{s}),\ s{:}\widehat{K}_\mathrm{p}(\widehat{s}, a)}$$

The class formation judgment $\Gamma \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}_\mathrm{f}$ yields a metafunction describing the type of proofs of formation family, while an auxiliary kind translation judgment $K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_\mathrm{p}$ yields a metafunction describing the kind of the predicate family. As in the example, the kind of the formation family is the same as the kind of the refined type, K.

The metafunction $\widehat{L}_\mathrm{f}$ takes as input the formation family so far, initially just $\widehat{s}$. The translation of Π classes adds an argument, and the base case returns the formation family so constructed.

$$\frac{\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S} \qquad \Gamma, x{::}S{\sqsubset}A \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}}{\Gamma \vdash \Pi x{::}S{\sqsubset}A.\ L \sqsubset \Pi x{:}A.\ K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_\mathrm{f}\mathbf{.}\ \Pi x{:}A.\ \Pi \widehat{x}{:}\widehat{S}(\eta_A(x)).\ \widehat{L}(Q_\mathrm{f}\ \eta_A(x))}$$

$$\frac{}{\Gamma \vdash \mathsf{sort} \sqsubset \mathsf{type} \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_\mathrm{f}\mathbf{.}\ Q_\mathrm{f}}$$

Employing a similar trick as we did with intersection sorts, we will translate intersection and $\top$ classes to unit and product types.

$$\frac{\Gamma \vdash L_1 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L_1} \qquad \Gamma \vdash L_2 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L_2}}{\Gamma \vdash L_1 \wedge L_2 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_\mathrm{f}\mathbf{.}\ \widehat{L_1}(Q_\mathrm{f}) \times \widehat{L_2}(Q_\mathrm{f})} \qquad \frac{}{\Gamma \vdash \top \sqsubset K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_\mathrm{f}\mathbf{.}\ 1}$$

Intersection classes give multiple ways for a sort to be well-formed, and a product of formation families gives multiple ways to project out a proof of well-formedness.

The metafunction $\widehat{K}_\mathrm{p}$ takes two arguments: one for the formation family so far (initially $\widehat{s}$) and one for the refined type so far (initially a). The rule for Π kinds just adds an argument

$$\boxed{\Gamma \vdash_\Sigma Q^+ \sqsubset P^- \Rightarrow L^- \rightsquigarrow \widehat{Q}^-}$$

$$\frac{s{\sqsubset}a{::}L \in \Sigma}{\Gamma \vdash s \sqsubset a \Rightarrow L \rightsquigarrow \widehat{s}/i}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow \Pi x{::}S{\sqsubset}A.\, L \rightsquigarrow \widehat{Q} \qquad \Gamma \vdash N \Leftarrow S \rightsquigarrow \widehat{N} \qquad [N/x]^{\mathrm{l}}_A\, L = L'}{\Gamma \vdash Q\ N \sqsubset P\ N \Rightarrow L' \rightsquigarrow \widehat{Q}\ N\ \widehat{N}}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2 \rightsquigarrow \widehat{Q}}{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \rightsquigarrow \pi_1\, \widehat{Q}} \qquad \frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2 \rightsquigarrow \widehat{Q}}{\Gamma \vdash Q \sqsubset P \Rightarrow L_2 \rightsquigarrow \pi_2\, \widehat{Q}}$$

Figure 5: Translation rules for base sort class synthesis

to each:

$$\frac{K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}}{\Pi x{:}A.\, K \overset{\text{pred}}{\rightsquigarrow} \boldsymbol{\lambda}(Q_{\mathrm{f}}, P).\ \Pi x{:}A.\, \widehat{K}(Q_{\mathrm{f}}\ \eta_A(x), P\ \eta_A(x))}$$

while the translation is really characterized by its behavior on the base kind, type:

$$\frac{}{\mathsf{type} \overset{\text{pred}}{\rightsquigarrow} \boldsymbol{\lambda}(Q_{\mathrm{f}}, P).\ Q_{\mathrm{f}} \div P \to \mathsf{type}}$$

The kind of the predicate family for a base sort Q refining P is essentially a one-place judgment on terms of type P, along with an irrelevant argument belonging to the formation family of Q.

Finally, we are able to make sense of the rule for translating base sorts:

$$\frac{\Gamma \vdash Q \sqsubset P' \Rightarrow L \rightsquigarrow \widehat{Q} \qquad P' = P \qquad L = \mathsf{sort}}{\Gamma \vdash Q \sqsubset P \rightsquigarrow \boldsymbol{\lambda} N.\ Q\ [\widehat{Q}]\ N}\ (Q\textbf{-F})$$

The class synthesis translation judgment $\Gamma \vdash Q \sqsubset P \Rightarrow L \rightsquigarrow \widehat{Q}$ (similar to the sort synthesis judgment; see Figure 5) yields a proof of Q's formation family; thus the predicate for a base sort Q, given an argument N, is simply the predicate family Q applied to an irrelevant proof $\widehat{Q}$ that Q is well-formed and the argument itself, N.

What if we hadn't made the proofs of formation irrelevant? Then if there were more than one proof that Q were well-formed, a soundness problem would arise. To see how, let us return to the doubling example. Imagine extending our encoding of natural numbers with a sort distinguishing zero as a refinement.

$\mathit{zero} \sqsubset \mathit{nat}.$

$z :: \mathit{even} \ \wedge \ \mathit{zero}.$

As with *even* and *odd*, the sort *zero* turns into a predicate. Now that z has two sorts, it translates to two proof constructors.[10]

[10]For the sake of simplicity, we will continue our example with the slightly unfaithful assumptions we've been making all along. Strictly speaking, *zero* should also have a formation family with a single trivial member, and the two declarations $\widehat{z}_1$ and $\widehat{z}_2$ should be one declaration of product type. The point we wish to make will be the same nonetheless.

$zero : nat \to \mathsf{type}.$
$\widehat{z}_1 : even\ z.$
$\widehat{z}_2 : zero\ z.$

Next, we can observe that zero always doubles to itself and augment the declaration of $double^*$ using an intersection class:

$$double^* \sqsubset double :: \top \to even \to \mathsf{sort} \\ \wedge\ zero \to zero \to \mathsf{sort}.$$

After translation, since there are potentially two ways for $double^*\ x\ y$ to be well-formed, there are two introduction constants for the formation family.

$\widehat{double}^*/i_1 : \Pi x{:}nat.\ \Pi y{:}nat.\ even\ y \to \widehat{double}^*\ x\ y.$
$\widehat{double}^*/i_2 : \Pi x{:}nat.\ zero\ x \to \Pi y{:}nat.\ zero\ y \to \widehat{double}^*\ x\ y.$

The declarations for $\widehat{double}^*$ and $double^*$ remain the same.

Now recall the refinement declaration for doubling zero,

$dbl/z :: double^*\ z\ z\ ,$

and observe that it is valid for two reasons, since $double^*\ z\ z$ is well-formed for two reasons. Consequently, after translation, there will be two proofs inhabiting the formation family $\widehat{double}^*\ z\ z$, but only one of them will be used in the translation of the dbl/z declaration. Supposing it is the first one, we'll have

$\widehat{dbl/z} : double^*\ z\ z\ [\ \widehat{double}^*/i_1\ z\ z\ \widehat{z}_1\ ]\ dbl/z\ ,$

but our soundness criterion will still require that the constant $\widehat{dbl/z}$ check at the type $double^*\ z\ z\ [\ \widehat{double}^*/i_2\ z\ \widehat{z}_2\ z\ \widehat{z}_2\ ]\ dbl/z$, the other possibility. The apparent mismatch is resolved by the fact that the formation proofs are irrelevant, and so the two types are considered equal. Without proof irrelevance, the two types would be distinct and we would have a counterexample to the soundness theorem (Theorem 6.1) we prove below.

6.3. **Subsorting.** We now return to the question of how the translation handles subsorting. Recall that an LFR signature can include subsorting declarations between sort family constants, $s_1 \leq s_2$. For instance, continuing with our running example of the natural numbers, we might note that any *nat* that is *zero* is *even* by declaring:

$zero \leq even.$

Such a declaration may seem redundant, since the only thing declared to have sort *zero* has *already* been declared to have sort *even*, but it may be necessary given the inherently open-ended nature of an LF signature. We may find ourselves later in a situation where we have a new hypothesis $x : zero$, and without the inclusion, we would not be able to conclude that $x : even$. For example the derivation of $\cdot \vdash \lambda x.\, x \Leftarrow zero \to even$ requires the inclusion to satisfy the second premise of the **switch** rule.

$$\dfrac{\dfrac{}{x{:}zero \vdash x \Rightarrow zero}\ \mathbf{var} \quad \dfrac{zero \leq even \in \Sigma}{zero \leq even}}{\dfrac{x{:}zero \vdash x \Leftarrow even}{\cdot \vdash \lambda x.\, x \Leftarrow zero \to even}\ \Pi\text{-}\mathbf{I}}\ \mathbf{switch}$$

How should we translate that derivation into a proof? As we saw earlier, the representation of *zero* → *even* as a predicate is $\boldsymbol{\lambda} N.\ \Pi x{:}\mathit{nat}.\ \mathit{zero}\ x \to \mathit{even}\ (N\ @\ x)$, and applying this predicate to $\lambda x.\, x$ yields the type we need the proof to have: $\Pi x{:}\mathit{nat}.\ \mathit{zero}\ x \to \mathit{even}\ x$. It is not much of a leap of the imagination to see that one solution is simply to posit a constant of the appropriate type:

$\mathit{zero}\text{-}\mathit{even} : \Pi x{:}\mathit{nat}.\ \mathit{zero}\ x \to \mathit{even}\ x.$

Now the translation of $\lambda x.\, x \Leftarrow \mathit{zero} \to \mathit{even}$ can be simply the η-expansion of this constant: $\lambda x.\, \lambda \widehat{x}.\, \mathit{zero}\text{-}\mathit{even}\ x\ \widehat{x}$. This makes intuitive sense: the constant *zero*-*even* witnesses the meaning of the declaration $\mathit{zero} \leq \mathit{even}$ under the subset interpretation.

Our example leads us to a rule: a subsorting declaration $s_1 \leq s_2$ will will translate into a declaration for a coercion constant $s_1\text{-}s_2$.

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad s_1 \sqsubset a{::}L \in \Sigma \qquad s_2 \sqsubset a{::}L \in \Sigma \qquad a{:}K \in \Sigma \qquad K \overset{\leq}{\rightsquigarrow} \widehat{K}_{\mathrm{s}}}{\vdash \Sigma, s_1 \leq s_2\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}, s_1\text{-}s_2{:}\widehat{K}_{\mathrm{s}}(a, \widehat{s_1}, s_1, \widehat{s_2}, s_2)}$$

The auxiliary judgment $K \overset{\leq}{\rightsquigarrow} \widehat{K}_{\mathrm{s}}$ yields a metafunction describing the type of proof coercions between sorts that refine a type family of kind K. The metafunction $\widehat{K}_{\mathrm{s}}$ takes five arguments: the refined type, the formation family and predicate family for the domain of the coercion, and the formation family and predicate family for the codomain of the coercion. As before, the Π translation adds an argument to each of the meta-arguments.

$$\frac{K \overset{\leq}{\rightsquigarrow} \widehat{K}}{\Pi x{:}A.\, K \overset{\leq}{\rightsquigarrow} \boldsymbol{\lambda}(P, Q_{1\mathrm{f}}, Q_1, Q_{2\mathrm{f}}, Q_2).\ \Pi x{:}A.\, \widehat{K}(P', Q_{1\mathrm{f}}', Q_1', Q_{2\mathrm{f}}', Q_2')}$$
(where, for each P, $P' = P\ \eta_A(x)$)

At the base kind type, the rule outputs the type of the coercion:

$$\frac{}{\mathsf{type} \overset{\leq}{\rightsquigarrow} \boldsymbol{\lambda}(P, Q_{1\mathrm{f}}, Q_1, Q_{2\mathrm{f}}, Q_2).\ \Pi f_1{:}Q_{1\mathrm{f}}.\, \Pi f_2{:}Q_{2\mathrm{f}}.\, \Pi x{:}P.\, Q_1\ [f_1]\ x \to Q_2\ [f_2]\ x}$$

Essentially, this is the type of coercions, given x, from proofs of $Q_1\ x$ to proofs of $Q_2\ x$, but in the general case, we must pass the predicates Q_1 and Q_2 evidence that they are well-formed, so the coercion requires formation proofs as inputs as well.

How do these coercions work? Recall that subsorting need only be defined at base sorts Q, and there, it is simply the application-compatible, reflexive, transitive closure of the declared relation. For the purposes of the translation, we employ an equivalent algorithmic formulation of subsorting. Following the inspiration of bidirectional typing, there are two judgments: a *checking* judgment that takes two base sorts as inputs and a *synthesis* judgment that takes one base sort as input and outputs another base sort that is one step higher in the subsort hierarchy.

The synthesis judgment constructs a coercion from the new coercion constants in the signature.

$$\frac{s_1 \leq s_2 \in \Sigma}{s_1 \leq s_2 \rightsquigarrow s_1\text{-}s_2} \qquad \frac{Q_1 \leq Q_2 \rightsquigarrow \widehat{Q_1\text{-}Q_2}}{Q_1\ N \leq Q_2\ N \rightsquigarrow \widehat{Q_1\text{-}Q_2}\ N}$$

The checking judgment, on the other hand, constructs a *meta-level* coercion between proofs of the two sorts. It is defined by two rules: a rule of reflexivity and a rule to climb the

subsort hierarchy.

$$\frac{Q_1 = Q_2}{\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow \boldsymbol{\lambda}(R, R_1).\ R_1}\ (\textbf{refl})$$

$$\frac{\begin{array}{cc} Q_1 \leq Q' \rightsquigarrow \widehat{Q_1\text{-}Q'} & \Gamma \vdash Q_1 \sqsubset P \Rightarrow \mathsf{sort} \rightsquigarrow \widehat{Q_1} \\ \Gamma \vdash Q' \leq Q_2 \rightsquigarrow F & \Gamma \vdash Q' \sqsubset P \Rightarrow \mathsf{sort} \rightsquigarrow \widehat{Q'} \end{array}}{\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow \boldsymbol{\lambda}(R, R_1).\ F(R, \widehat{Q_1\text{-}Q'}\ \widehat{Q_1}\ \widehat{Q'}\ R\ R_1)}\ (\textbf{climb})$$

The reflexivity rule's metacoercion simply returns the proof it is given, while the climb rule composes the actual coercion $\widehat{Q_1\text{-}Q'}$ with the metacoercion F. Two extra premises generate the necessary formation proofs.

Finally, we have described enough of the translation to explain the rule most central to the design of LFR, the **switch** rule.

$$\frac{\Gamma \vdash R \Rightarrow Q' \rightsquigarrow \widehat{R} \qquad \Gamma \vdash Q' \leq Q \rightsquigarrow F}{\Gamma \vdash R \Leftarrow Q \rightsquigarrow F(R, \widehat{R})}\ (\textbf{switch})$$

The first premise produces a proof $\widehat{R}$ that R satisfies property Q', and the second premise generates the meta-level proof coercion that transforms such a proof into a proof that R satisfies property Q.

Having sketched the translation and the role of proof irrelevance, we now review some metatheoretic results.

6.4. **Correctness.** Our translation is both sound and complete with respect to the original system of LF with refinement types, and so our correctness criteria will come in two flavors.

Soundness theorems tell us that the result of a translation is well-formed. But even more importantly than telling us that our translation is on some level correct, they serve as an independent means of understanding the translation. In a sense, a soundness theorem can be read as the meta-level type of a translation judgment—a specification of its intended behavior—and just as types serve as an organizing principle for the practicing programmer, so too do soundness theorems serve the thoughtful theoretician. We explain our soundness theorems, then, not only to demonstrate the sensibility of our translation, but also to aid the reader in understanding its purpose.

In what follows, $\mathrm{form}(Q)$ represents the formation family for a base sort Q.

$$\mathrm{form}(s) = \widehat{s} \qquad\qquad \mathrm{form}(Q\ N) = \mathrm{form}(Q)\ N$$

Theorem 6.1 (Soundness)**.** *Suppose* $\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}$ *and* $\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}$*. Then:*

(1) *If* $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *and* $\Gamma \vdash N \Leftarrow S \rightsquigarrow \widehat{N}$*, then* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{N} \Leftarrow \widehat{S}(N)$*.*

(2) *If* $\Gamma \vdash R \Rightarrow S \rightsquigarrow \widehat{R}$*, then* $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *and* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{R} \Rightarrow \widehat{S}(\eta_A(R))$ *(for some* A *and* $\widehat{S}$*).*

(3) *If* $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *and* $\Gamma \vdash N \Leftarrow A$*, then* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{S}(N) \Leftarrow \mathsf{type}$*.*

(4) *If* $\Gamma \vdash Q \sqsubset P \Rightarrow L \rightsquigarrow \widehat{Q}$*, then for some* K*,* $\widehat{L}_{\mathrm{f}}$*, and* $\widehat{K}_{\mathrm{p}}$*,*
   - $\Gamma \vdash L \sqsubset K \overset{\mathrm{form}}{\rightsquigarrow} \widehat{L}_{\mathrm{f}}$ *and* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{Q} \Rightarrow \widehat{L}_{\mathrm{f}}(\,\mathrm{form}(Q))$*, and*
   - $K \overset{\mathrm{pred}}{\rightsquigarrow} \widehat{K}_{\mathrm{p}}$ *and* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} Q \Rightarrow \widehat{K}_{\mathrm{p}}(\,\mathrm{form}(Q), P)$*.*

(5) *If* $\Gamma \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}_{\text{f}}$ *and* $\Gamma \vdash P \Rightarrow K$, *then* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{L}_{\text{f}}(P) \Leftarrow \mathsf{type}$.

(6) *If* $K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_{\text{p}}$, $\Gamma \vdash Q_{\text{f}} \Rightarrow K$, *and* $\Gamma \vdash P \Rightarrow K$, *then* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{K}_{\text{p}}(Q_{\text{f}}, P) \Leftarrow \mathsf{kind}$.

(7) *If* $Q_1 \leq Q_2 \rightsquigarrow \widehat{Q_1\text{-}Q_2}$, $\Gamma \vdash Q_1 \sqsubset P \Rightarrow L$, $\Gamma \vdash P \Rightarrow K$, *and* $K \overset{\leq}{\rightsquigarrow} \widehat{K}_s$, *then* $\Gamma \vdash Q_2 \sqsubset P \Rightarrow L$ *and* $\widehat{\Gamma} \vdash \widehat{Q_1\text{-}Q_2} \Rightarrow \widehat{K}_s(P, \text{form}(Q_1), Q_1, \text{form}(Q_2), Q_2)$.

(8) *If* $\Gamma \vdash R \Rightarrow P$, $\Gamma \vdash Q_i \sqsubset P \rightsquigarrow \widehat{Q_i}$, $\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow F$, *and* $\widehat{\Gamma} \vdash R_1 \Rightarrow \widehat{Q_1}(R)$, *then* $\widehat{\Gamma} \vdash F(R, R_1) \Rightarrow \widehat{Q_2}(R)$.

(9) *If* $K \overset{\leq}{\rightsquigarrow} \widehat{K}_s$, $K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_{\text{p}}$, $\Gamma \vdash P \Rightarrow K$, $\Gamma \vdash Q_{i\text{f}} \Rightarrow K$, *and* $\widehat{\Gamma} \vdash Q_i \Rightarrow \widehat{K}_{\text{p}}(Q_{i\text{f}}, P)$, *then* $\widehat{\Gamma} \vdash \widehat{K}_s(P, Q_{1\text{f}}, Q_1, Q_{2\text{f}}, Q_2) \Leftarrow \mathsf{type}$.

Proof. By induction on each clause's main input derivation. Several clauses must be proved mutually; for instance, clauses 1, 2, 8, and 4 are all mutual, since the rules for translating terms refer to the translation of subsorting, the rules for translating subsorting refer to the class synthesis translation, and since sorts may be dependent, the rules for class synthesis translation refer back to the term translation. □

The proofs use entirely standard syntactic methods, but they appeal to several key lemmas about the structure of the translation.

Lemma 6.2 (Erasure)**.** *If* $\Gamma \vdash \mathcal{J} \rightsquigarrow X$, *then* $\Gamma \vdash \mathcal{J}$.

Proof. Straightforward induction on the structure of the translation derivation. The translation rules are premise-wise strictly more restrictive than the original LFR rules, except for the subsorting rules, which are also more restrictive in the sense that they force rules to be applied in a certain order. □

Lemma 6.3 (Reconstruction)**.** *If* $\Gamma \vdash \mathcal{J}$, *then for some* X, $\Gamma \vdash \mathcal{J} \rightsquigarrow X$.

Proof. By induction on the structure of the LFR derivation. The cases for the subsorting rules require us to demonstrate that an LFR subsorting derivation can be put into "algorithmic form", with all uses of reflexivity and transitivity outermost and right-nested, like the algorithmic translation rules **refl** and **climb**. We also make use of the tacit assumption that the judgment $\Gamma \vdash \mathcal{J}$ itself is well-formed, e.g. if $\mathcal{J} = N \Leftarrow S$, then $\Gamma \vdash S \sqsubset A$, which ensures that we will have the necessary formation premises when we need to apply the **climb** rule. □

Erasure and reconstruction substantiate the claim that our translation is derivation-directed by allowing us to move freely between translation judgments and ordinary ones. Using erasure and reconstruction, we can leverage all of the LFR metatheory without re-proving it for translation judgments. For example, several cases require us to substitute into a translation derivation: we can apply erasure, appeal to LFR's substitution theorem, and invoke reconstruction to get the output we require.

But since reconstruction only gives us *some* output X, we may not know that it is the one that suits our needs. Therefore, we usually require another lemma, compositionality, to tell us that the translation commutes with substitution. There are several such lemmas; we show here the one for sort translation.

Lemma 6.4 (Compositionality)**.** *Let* σ *denote* $[M/x]_A$.

(1) *If* $\Gamma_{\text{L}}, x{::}\_, \Gamma_{\text{R}} \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *and* $\Gamma_{\text{L}}, \sigma\Gamma_{\text{R}} \vdash \sigma S \sqsubset \sigma A \rightsquigarrow \widehat{S'}$, *then* $\sigma\widehat{S}(N) = \widehat{S'}(\sigma N)$,

(2) *If* $\Gamma_{\mathrm{L}}, x{::}\_, \Gamma_{\mathrm{R}} \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}$ *and* $\Gamma, \sigma\Gamma_{\mathrm{R}} \vdash \sigma L \sqsubset \sigma K \overset{\text{form}}{\rightsquigarrow} \widehat{L}'$, *then* $\sigma\widehat{L}(P) = \widehat{L}'(\sigma P)$,

and similarly for $K \overset{\leq}{\rightsquigarrow} \widehat{K}_s$ *and* $K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}_{\mathrm{p}}$.

Proof. Straightforward induction using functionality of hereditary substitution. The base case of the first clause leverages the irrelevance introduced in the Q-**F** translation rule: both sort formation derivations will have a premise outputting evidence for the well-formedness of the sort, and there is no guarantee they will output the *same* evidence, but since the evidence is relegated to an irrelevant position, its identity is ignored. The second clause's Π case appeals to the first clause, since Π classes contain sorts. □

Finally, there is a lemma demonstrating that proof variables only ever occur irrelevantly, so substituting for them cannot change the identity of a sort or class meta-function output by the translation.

Lemma 6.5 (Proof Variable Substitution)**.**

(1) *If* $\Gamma_{\mathrm{L}}, x{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *then* $[M/\widehat{x}]^{\mathrm{a}}_{A_0}\, \widehat{S}(N) = \widehat{S}([M/\widehat{x}]^{\mathrm{n}}_{A_0}\, N)$.

(2) *If* $\Gamma_{\mathrm{L}}, x{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}$ *then* $[M/\widehat{x}]^{\mathrm{a}}_{A_0}\, \widehat{L}(P) = \widehat{L}([M/\widehat{x}]^{\mathrm{p}}_{A_0}\, P)$.

Proof. Straightforward induction, noting in the base case, the Q-**F** rule, the only term that could depend on $\widehat{x}$ is in an irrelevant position. □

Completeness theorems tell us that our target is not too rich: that everything we find evidence of in the codomain of the translation actually holds true in its domain. While important for establishing general correctness, completeness theorems are not as informative as soundness theorems, so we give here only the cases for terms—and in any case, those are the only cases we require to fulfill our goal of preserving adequacy.

In stating completeness, we syntactically isolate the set of terms that could represent proofs using metavariables $\widehat{R}$ and $\widehat{N}$.

$$\begin{aligned}
\widehat{R} &::= \widehat{c} \mid \widehat{x} \mid \widehat{R}\ N\ \widehat{N} \mid \pi_1\, \widehat{R} \mid \pi_2\, \widehat{R} \\
\widehat{N} &::= \widehat{F} \mid \lambda x.\, \lambda \widehat{x}.\, \widehat{N} \mid \langle \widehat{N_1}, \widehat{N_2} \rangle \mid \langle\rangle \\
\widehat{F} &::= \widehat{R} \mid \widehat{Q_1\text{-}Q_2}\ \widehat{Q_1}\ \widehat{Q_2}\ R\ F \\
\widehat{Q_1\text{-}Q_2} &::= s_1\text{-}s_2 \mid \widehat{Q_1\text{-}Q_2}\ N \\
\widehat{Q} &::= \widehat{s}/i \mid \widehat{Q}\ N\ \widehat{N} \mid \pi_1\, \widehat{Q} \mid \pi_2\, \widehat{Q}
\end{aligned}$$

Theorem 6.6 (Completeness)**.** *Suppose* $\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}$ *and* $\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}$. *Then:*

(1) *If* $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$ *and* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{N} \Leftarrow \widehat{S}(N)$, *then* $\Gamma \vdash N \Leftarrow S$.

(2) *If* $\widehat{\Gamma} \vdash_{\widehat{\Sigma}} \widehat{R} \Rightarrow B$, *then* $\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}$, $B = \widehat{S}(\eta_A(R))$, *and* $\Gamma \vdash R \Rightarrow S$ *(for some* S, A, $\widehat{S}$, *and* R*).*

(3) *If* $\widehat{\Gamma} \vdash \widehat{F} \Rightarrow Q\ [\widehat{Q}]\ R$, *then* $\widehat{\Gamma} \vdash \widehat{R} \Leftarrow Q$.

(4) *If* $\widehat{\Gamma} \vdash \widehat{Q_1\text{-}Q_2} \Rightarrow B$, *then* $K \overset{\leq}{\rightsquigarrow} \widehat{K}_s$, $B = \widehat{K}_s(P, \mathrm{form}(Q_1), Q_1, \mathrm{form}(Q_2), Q_2)$, *and* $Q_1 \leq Q_2$ *(for some* K, $\widehat{K}_s$, P, Q_1, *and* Q_2*).*

(5) *If* $\widehat{\Gamma} \vdash \widehat{Q} \Rightarrow B$, *then* $\Gamma \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}_{\mathrm{f}}$, $B = \widehat{L}_{\mathrm{f}}(\mathrm{form}(Q))$, *and* $\Gamma \vdash Q \sqsubset P \Rightarrow L$ *(for some* L, K, $\widehat{L}_{\mathrm{f}}$, *and* Q*).*

Proof. By induction over the structure of the proof term. □

Adequacy of a representation is generally shown by exhibiting a compositional bijection between informal entities and terms of certain LFR sorts. Since we have undertaken a subset interpretation, the set of terms of any LFR sort are unchanged by translation, and so any bijective correspondence between those terms and informal entities remains after translation. Furthermore, soundness and completeness tell us that our interpretation preserves and reflects the derivability of any refinement type judgments over those terms. Thus, we have achieved our main goal: any adequate LFR representation can be translated to an adequate LFI representation.

7. Conclusion

Logical frameworks are metalanguages specifically designed so that common concepts and notations in logic and the theory of programming languages can be represented elegantly and concisely. LF [HHP93] intrinsically supports α-conversion, capture-avoiding substitution, and hypothetical and parametric judgments, but as with any such enterprise, certain patterns fall out of its scope and must be encoded indirectly. One pattern is the ability to form regular subsets of types already defined. We address this by extending LF with type refinements, leveraging the modern view of LF as a calculus of canonical forms to obtain a metatheoretically simple yet expressive system, LFR. Another pattern is to ignore the identities of proofs, relying only on their existence. This is addressed in LF extended with proof irrelevance, LFI [Pfe01a, RP08]. We have shown that our system of refinement types can be mapped into LFI in a bijective manner, preserving adequacy theorems for LFR representations in LFI.

In the methodology of logical frameworks research, it is important to understand the cost of such a translation: how much more complicated are encodings in the target framework, and how much more difficult is it to work with them? We cannot measure this cost precisely, but we hope it is evident from the definition of the translation and the examples that the price is considerable. Even if in special cases more direct encodings are possible, we believe our general translation could not be simplified much, given the explicit goal to preserve the adequacy of representations. Other translations from programming languages, such as coercion interpretations where sorts are translated to distinct types and subsorting to coercions, appear even more complex because adequacy depends on certain functional equalities between coercions. Our preliminary conclusion is that refinement types in logical frameworks provide elegant and immediate representations that are not easy to simulate without them, providing a solid argument for their inclusion in the next generation of frameworks.

Of course, much work remains to be done before refinement types can be considered a practical addition. First, it will be necessary to develop a sufficiently complete algorithm for reconstructing the sorts of implicitly Π-quantified metavariables in order to allow the elegant encodings we imagine without burdensome redundancy. Furthermore, it would be useful to have a logic programming interpretation of LFR declarations and the ability to perform analyses like coverage and termination checking on declarations *qua* programs; to enable such an interpretation, we will have to develop an algorithm for sorted unification, generalizing existing work on pattern unification in the context of logical frameworks. It may also be a worthwhile endeavor to formalize the metatheory of LFR and its subset interpretation in a metalogical framework or proof assistant; although we have avoided doing so due to the high cost of working around current technological limitations in proof

assistants, the present work has been carried out in sufficient detail that formalization should not be particularly difficult beyond the technical challenge of representing a dependently typed calculus.

Refinement types have been also been proposed for functional programming [Fre94, DP04, Dav05], most recently in conjunction with a limited form of dependent types [Dun07]. Proof irrelevance is already integrated in this setting, and also available in general type theories such as NuPrl or Coq. One can ask the same question here: Can we simply eliminate refinement types and just work with dependent types and proof irrelevance? The results in this paper lend support to the conjecture that this can be accomplished by a uniform translation. On the other hand, just as here, it seems there would likely be a high cost in terms of brevity in order to maintain a bijection between well-sorted data in the source and dependently well-typed data in the target of the translation.

Acknowledgements. Thanks to Jason Reed for many fruitful discussions on the topic of proof irrelevance. Thanks to the anonymous referees for offering insightful commentary on how to clarify our presentation.

References

[AB04] Steven Awodey and Andrej Bauer. Propositions as [types]. *Journal of Logic and Computation*, 14(4):447–471, 2004.

[AC01] David Aspinall and Adriana B. Compagnoni. Subtyping dependent types. *Theoretical Computer Science*, 266(1-2):273–309, 2001.

[BTCGS91] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.

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

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[Cra03] Karl Crary. Toward a foundational typed assembly language. In G. Morrisett, editor, *Proceedings of the 30th Annual Symposium on Principles of Programming Languages (POPL '03)*, pages 198–212, New Orleans, Louisiana, January 2003. ACM Press.

[Dav05] Rowan Davies. *Practical Refinement-Type Checking*. PhD thesis, Carnegie Mellon University, May 2005. Available as Technical Report CMU-CS-05-110.

[DP04] Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In Xavier Leroy, editor, *ACM Symp. Principles of Programming Languages (POPL '04)*, pages 281–292, Venice, Italy, January 2004.

[Dun07] Joshua Dunfield. *A Unified System of Type Refinements*. PhD thesis, Carnegie Mellon University, August 2007. Available as Technical Report CMU-CS-07-129.

[DZ92] Philip W. Dart and Justin Zobel. A regular type language for logic programs. In Frank Pfenning, editor, *Types in Logic Programming*, pages 157–187. MIT Press, Cambridge, Massachusetts, 1992.

[FP91] Tim Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the SIGPLAN '91 Symposium on Language Design and Implementation*, pages 268–277, Toronto, Ontario, June 1991. ACM Press.

[Fre94] Tim Freeman. *Refinement Types for ML*. PhD thesis, Carnegie Mellon University, March 1994. Available as Technical Report CMU-CS-94-110.

[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.

[HL07] Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4–5):613–673, July 2007.

[HP05] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *Transactions on Computational Logic*, 6:61–101, January 2005.

[LCH07] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of Standard ML. In Matthias Felleisen, editor, *Proceedings of the 34th Annual Symposium on Principles of Programming Languages (POPL '07)*, pages 173–184, Nice, France, January 2007. ACM Press.

[LP08a] William Lovas and Frank Pfenning. A bidirectional refinement type system for LF. *Electronic Notes in Theoretical Computer Science*, 196:113–128, January 2008.

[LP08b] William Lovas and Frank Pfenning. A bidirectional refinement type system for LF. Technical Report CMU-CS-08-129, Department of Computer Science, Carnegie Mellon University, 2008.

[LP09] William Lovas and Frank Pfenning. Refinement types as proof irrelevance. In Pierre-Louis Curien, editor, *Proceedings of 9th International Conference on Typed Lambda Calculi and Applications (TLCA 2009)*, number 5608 in Lecture Notes in Computer Science, pages 157–171. Springer, 2009.

[LRDR07] Luigi Liquori and Simona Ronchi Della Rocca. Intersection-types à la Church. *Information and Computation*, 205(9):1371–1386, 2007.

[NPP07] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *Transactions on Computational Logic*, 2007. To appear.

[Pfe93] Frank Pfenning. Refinement types for logical frameworks. In Herman Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, pages 285–299, Nijmegen, The Netherlands, May 1993.

[Pfe00] Frank Pfenning. Structural cut elimination: I. intuitionistic and classical logic. *Information and Computation*, 157(1-2):84–141, 2000.

[Pfe01a] Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In J. Halpern, editor, *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, pages 221–230, Boston, Massachusetts, June 2001. IEEE Computer Society Press.

[Pfe01b] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 17, pages 1063–1147. Elsevier Science and MIT Press, 2001.

[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.

[Rey89] John C. Reynolds. Even normal forms can be hard to type. Unpublished, marked Carnegie Mellon University, December 1, 1989.

[Rey91] John C. Reynolds. The coherence of languages with intersection types. In Takayasu Ito and Albert R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 675–700, Berlin, 1991. Springer-Verlag.

[Rey96] John C. Reynolds. Design of the programming language Forsythe. Report CMU–CS–96–146, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 28, 1996.

[RP08] Jason Reed and Frank Pfenning. Proof irrelevance in a logical framework. Unpublished draft, July 2008.

[Sar09] Susmit Sarkar. *A Dependently Typed Programming Language, with Applications to Foundational Certified Code Systems*. PhD thesis, Carnegie Mellon University, May 2009. Available as Technical Report CMU-CS-09-128.

[Sch03] Carsten Schürmann. Towards practical functional programming with logical frameworks. Unpublished, available at http://cs-www.cs.yale.edu/homes/carsten/delphin/, July 2003.

[SS88] Anne Salvesen and Jan M. Smith. The strength of the subset type in Martin-Löf's type theory. In *Proceedings of LICS'88*, pages 384–391. IEEE Computer Society Press, 1988.

[WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

Appendix A. Complete LFR Rules

In the judgment forms below, superscript + and − indicate a judgment's "inputs" and "outputs", respectively.

A.1. **Grammar.**

Kind level

$$K ::= \mathsf{type} \mid \Pi x{:}A.\, K \qquad \text{kinds}$$
$$L ::= \mathsf{sort} \mid \Pi x{::}S \sqsubset A.\, L \mid \top \mid L_1 \wedge L_2 \qquad \text{classes}$$

Type level

$$P ::= a \mid P\ N \qquad \text{atomic type families}$$
$$A ::= P \mid \Pi x{:}A_1.\, A_2 \qquad \text{canonical type families}$$

$$Q ::= s \mid Q\ N \qquad \text{atomic sort families}$$
$$S ::= Q \mid \Pi x{::}S_1 \sqsubset A_1.\, S_2 \mid \top \mid S_1 \wedge S_2 \qquad \text{canonical sort families}$$

Term level

$$R ::= c \mid x \mid R\ N \qquad \text{atomic terms}$$
$$N ::= R \mid \lambda x.\, N \qquad \text{canonical terms}$$

Signatures and contexts

$$\Sigma ::= \cdot \mid \Sigma, D \qquad \text{signatures}$$
$$D ::= a{:}K \mid c{:}A \mid s \sqsubset a{::}L \mid s_1 \leq s_2 \mid c{::}S \qquad \text{declarations}$$
$$\Gamma ::= \cdot \mid \Gamma, x{::}S \sqsubset A \qquad \text{contexts}$$

A.2. **Expansion and Substitution.** All bound variables are tacitly assumed to be sufficiently fresh.

$\boxed{(A)^- = \alpha}$

$$\alpha, \beta ::= a \mid \alpha_1 \to \alpha_2$$

$$(a)^- = a$$
$$(P\ N)^- = (P)^-$$
$$(\Pi x{:}A.\, B)^- = (A)^- \to (B)^-$$

$\boxed{\eta_\alpha(R) = N}$

$$\eta_a(R) = R$$
$$\eta_{\alpha \to \beta}(R) = \lambda x.\, \eta_\beta(R\ \eta_\alpha(x))$$

$$\boxed{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N = N'}$$

$$\frac{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R = (N, a)}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, R = N} \qquad \frac{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R'}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, R = R'} \qquad \frac{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N = N'}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, \lambda x.\, N = \lambda x.\, N'}$$

$$\boxed{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R'}$$

$$\frac{x \neq x_0}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, x = x} \qquad \frac{}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, c = c} \qquad \frac{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_1 = R_1' \qquad [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2'}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_1\ N_2 = R_1'\ N_2'}$$

$$\boxed{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R = (N', \alpha')}$$

$$\frac{}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, x_0 = (N_0, \alpha_0)}\ (\textbf{subst-rn-var})$$

$$\frac{\begin{array}{c}[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R_1 = (\lambda x.\, N_1, \alpha_2 \to \alpha_1) \\ [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2' \qquad [N_2'/x]^{\mathrm{n}}_{\alpha_2}\, N_1 = N_1'\end{array}}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0}\, R_1\ N_2 = (N_1', \alpha_1)}\ (\textbf{subst-rn-}\beta)$$

(Substitution for other syntactic categories (q, p, s, a, l, k, γ) is compositional.)

A.3. **Kinding.**

$$\boxed{\Gamma \vdash_\Sigma L^+ \sqsubset K^+}$$

$$\frac{}{\Gamma \vdash \mathsf{sort} \sqsubset \mathsf{type}} \qquad \frac{\Gamma \vdash S \sqsubset A \qquad \Gamma, x{::}S{\sqsubset}A \vdash L \sqsubset K}{\Gamma \vdash \Pi x{::}S{\sqsubset}A.\, L \sqsubset \Pi x{:}A.\, K}$$

$$\frac{}{\Gamma \vdash \top \sqsubset K} \qquad \frac{\Gamma \vdash L_1 \sqsubset K \qquad \Gamma \vdash L_2 \sqsubset K}{\Gamma \vdash L_1 \wedge L_2 \sqsubset K}$$

$$\boxed{\Gamma \vdash_\Sigma Q^+ \sqsubset P^- \Rightarrow L^-}$$

$$\frac{s{\sqsubset}a{::}L \in \Sigma}{\Gamma \vdash s \sqsubset a \Rightarrow L}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow \Pi x{::}S{\sqsubset}A.\, L \qquad \Gamma \vdash N \Leftarrow S \qquad [N/x]^1_A\, L = L'}{\Gamma \vdash Q\ N \sqsubset P\ N \Rightarrow L'}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2}{\Gamma \vdash Q \sqsubset P \Rightarrow L_1} \qquad \frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2}{\Gamma \vdash Q \sqsubset P \Rightarrow L_2}$$

$$\boxed{\Gamma \vdash_\Sigma S^+ \sqsubset A^+}$$

$$\frac{\Gamma \vdash Q \sqsubset P' \Rightarrow L \qquad P' = P \qquad L = \mathsf{sort}}{\Gamma \vdash Q \sqsubset P}\ (Q\textbf{-F})$$

$$\frac{\Gamma \vdash S \sqsubset A \qquad \Gamma, x{::}S{\sqsubset}A \vdash S' \sqsubset A'}{\Gamma \vdash \Pi x{::}S{\sqsubset}A.\, S' \sqsubset \Pi x{:}A.\, A'}\ (\Pi\textbf{-F})$$

$$\frac{}{\Gamma \vdash \top \sqsubset A}\ (\top\textbf{-F}) \qquad \frac{\Gamma \vdash S_1 \sqsubset A \qquad \Gamma \vdash S_2 \sqsubset A}{\Gamma \vdash S_1 \wedge S_2 \sqsubset A}\ (\wedge\textbf{-F})$$

Note: no intro rules for classes $\top$ and $L_1 \wedge L_2$.

A.4. **Typing.**

$\boxed{\Gamma \vdash_\Sigma R^+ \Rightarrow S^-}$

$$\frac{c{::}S \in \Sigma}{\Gamma \vdash c \Rightarrow S}\ (\textbf{const}) \qquad \frac{x{::}S{\sqsubset}A \in \Gamma}{\Gamma \vdash x \Rightarrow S}\ (\textbf{var})$$

$$\frac{\Gamma \vdash R_1 \Rightarrow \Pi x{::}S_2{\sqsubset}A_2.\, S \qquad \Gamma \vdash N_2 \Leftarrow S_2 \qquad [N_2/x]^{\mathrm{s}}_{A_2}\, S = S'}{\Gamma \vdash R_1\ N_2 \Rightarrow S'}\ (\Pi\textbf{-E})$$

$$\frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma \vdash R \Rightarrow S_1}\ (\wedge\textbf{-E}_1) \qquad \frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma \vdash R \Rightarrow S_2}\ (\wedge\textbf{-E}_2)$$

$\boxed{\Gamma \vdash_\Sigma N^+ \Leftarrow S^+}$

$$\frac{\Gamma \vdash R \Rightarrow Q' \qquad Q' \leq Q}{\Gamma \vdash R \Leftarrow Q}\ (\textbf{switch})$$

$$\frac{\Gamma, x{::}S{\sqsubset}A \vdash N \Leftarrow S'}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x{::}S{\sqsubset}A.\, S'}\ (\Pi\textbf{-I})$$

$$\frac{}{\Gamma \vdash N \Leftarrow \top}\ (\top\textbf{-I}) \qquad \frac{\Gamma \vdash N \Leftarrow S_1 \qquad \Gamma \vdash N \Leftarrow S_2}{\Gamma \vdash N \Leftarrow S_1 \wedge S_2}\ (\wedge\textbf{-I})$$

$\boxed{Q_1^+ \leq Q_2^+}$

$$\frac{Q_1 = Q_2}{Q_1 \leq Q_2} \qquad \frac{Q_1 \leq Q' \qquad Q' \leq Q_2}{Q_1 \leq Q_2} \qquad \frac{s_1{\leq}s_2 \in \Sigma}{s_1 \leq s_2} \qquad \frac{Q_1 \leq Q_2}{Q_1\ N \leq Q_2\ N}$$

A.5. **Signatures and Contexts.**

$\boxed{\vdash \Sigma\ \mathsf{sig}}$

$$\frac{}{\vdash \cdot\ \mathsf{sig}} \qquad \frac{\vdash \Sigma\ \mathsf{sig} \qquad \cdot \vdash_{\Sigma^*} K \Leftarrow \mathsf{kind} \qquad a{:}K' \notin \Sigma}{\vdash \Sigma, a{:}K\ \mathsf{sig}}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad \cdot \vdash_{\Sigma^*} A \Leftarrow \mathsf{type} \qquad c{:}A' \notin \Sigma}{\vdash \Sigma, c{:}A\ \mathsf{sig}}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad a{:}K \in \Sigma \qquad \cdot \vdash_{\Sigma} L \sqsubset K \qquad s \sqsubset a'{::}L' \notin \Sigma}{\vdash \Sigma, s \sqsubset a{::}L\ \mathsf{sig}}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad c{:}A \in \Sigma \qquad \cdot \vdash_{\Sigma} S \sqsubset A \qquad c{::}S' \notin \Sigma}{\vdash \Sigma, c{::}S\ \mathsf{sig}}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad s_1 \sqsubset a{::}L \in \Sigma \qquad s_2 \sqsubset a{::}L \in \Sigma}{\vdash \Sigma, s_1 \leq s_2\ \mathsf{sig}}$$

$\boxed{\vdash_\Sigma \Gamma\ \mathsf{ctx}}$

$$\frac{}{\vdash \cdot\ \mathsf{ctx}} \qquad \frac{\vdash \Gamma\ \mathsf{ctx} \qquad \Gamma \vdash S \sqsubset A}{\vdash \Gamma, x{::}S \sqsubset A\ \mathsf{ctx}}$$

Appendix B. Complete Translation Rules

In the judgment forms below, superscript + and − indicate a judgment's "inputs" and "outputs", respectively.

B.1. **Kinding.**

$$\boxed{\Gamma \vdash_\Sigma L^+ \sqsubset K^+ \overset{\text{form}}{\rightsquigarrow} \widehat{L}^-}$$

$$\frac{}{\Gamma \vdash \mathsf{sort} \sqsubset \mathsf{type} \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_{\mathrm{f}}.\ Q_{\mathrm{f}}}$$

$$\frac{\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S} \qquad \Gamma, x{::}S \sqsubset A \vdash L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L}}{\Gamma \vdash \Pi x{::}S \sqsubset A.\ L \sqsubset \Pi x{:}A.\ K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_{\mathrm{f}}.\ \Pi x{:}A.\ \Pi \widehat{x}{:}\widehat{S}(\eta_A(x)).\ \widehat{L}(Q_{\mathrm{f}}\ \eta_A(x))}$$

$$\frac{}{\Gamma \vdash \top \sqsubset K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_{\mathrm{f}}.\ 1} \qquad \frac{\Gamma \vdash L_1 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L_1} \qquad \Gamma \vdash L_2 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L_2}}{\Gamma \vdash L_1 \wedge L_2 \sqsubset K \overset{\text{form}}{\rightsquigarrow} \boldsymbol{\lambda} Q_{\mathrm{f}}.\ \widehat{L_1}(Q_{\mathrm{f}}) \times \widehat{L_2}(Q_{\mathrm{f}})}$$

$$\boxed{\Gamma \vdash_\Sigma Q^+ \sqsubset P^- \Rightarrow L^- \rightsquigarrow \widehat{Q}^-}$$

$$\frac{s \sqsubset a{::}L \in \Sigma}{\Gamma \vdash s \sqsubset a \Rightarrow L \rightsquigarrow \widehat{s}/i}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow \Pi x{::}S \sqsubset A.\ L \rightsquigarrow \widehat{Q} \qquad \Gamma \vdash N \Leftarrow S \rightsquigarrow \widehat{N} \qquad [N/x]^{\mathrm{l}}_A\, L = L'}{\Gamma \vdash Q\ N \sqsubset P\ N \Rightarrow L' \rightsquigarrow \widehat{Q}\ N\ \widehat{N}}$$

$$\frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2 \rightsquigarrow \widehat{Q}}{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \rightsquigarrow \pi_1\, \widehat{Q}} \qquad \frac{\Gamma \vdash Q \sqsubset P \Rightarrow L_1 \wedge L_2 \rightsquigarrow \widehat{Q}}{\Gamma \vdash Q \sqsubset P \Rightarrow L_2 \rightsquigarrow \pi_2\, \widehat{Q}}$$

$$\boxed{\Gamma \vdash_\Sigma S^+ \sqsubset A^+ \rightsquigarrow \widehat{S}^-}$$

$$\frac{\Gamma \vdash Q \sqsubset P' \Rightarrow L \rightsquigarrow \widehat{Q} \qquad P' = P \qquad L = \mathsf{sort}}{\Gamma \vdash Q \sqsubset P \rightsquigarrow \boldsymbol{\lambda} N.\ Q\ [\widehat{Q}]\ N}\ (Q\textbf{-F})$$

$$\frac{\Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S} \qquad \Gamma, x{::}S \sqsubset A \vdash S' \sqsubset A' \rightsquigarrow \widehat{S'}}{\Gamma \vdash \Pi x{::}S \sqsubset A.\ S' \sqsubset \Pi x{:}A.\ A' \rightsquigarrow \boldsymbol{\lambda} N.\ \Pi x{:}A.\ \Pi \widehat{x}{:}\widehat{S}(\eta_A(x)).\ \widehat{S'}(N @ x)}\ (\Pi\textbf{-F})$$

$$\frac{}{\Gamma \vdash \top \sqsubset A \rightsquigarrow \boldsymbol{\lambda} N.\ 1}\ (\top\textbf{-F}) \qquad \frac{\Gamma \vdash S_1 \sqsubset A \rightsquigarrow \widehat{S_1} \qquad \Gamma \vdash S_2 \sqsubset A \rightsquigarrow \widehat{S_2}}{\Gamma \vdash S_1 \wedge S_2 \sqsubset A \rightsquigarrow \boldsymbol{\lambda} N.\ \widehat{S_1}(N) \times \widehat{S_2}(N)}\ (\wedge\textbf{-F})$$

Note: no intro rules for classes $\top$ and $L_1 \wedge L_2$.

$\boxed{K^+ \overset{\text{pred}}{\rightsquigarrow} \widehat{K}^-}$

$$\frac{}{\mathsf{type} \overset{\text{pred}}{\rightsquigarrow} \boldsymbol{\lambda}(Q_{\mathrm{f}}, P).\ Q_{\mathrm{f}} \div P \to \mathsf{type}}$$

$$\frac{K \overset{\text{pred}}{\rightsquigarrow} \widehat{K}}{\Pi x{:}A.\, K \overset{\text{pred}}{\rightsquigarrow} \boldsymbol{\lambda}(Q_{\mathrm{f}}, P).\ \Pi x{:}A.\, \widehat{K}(Q_{\mathrm{f}}\ \eta_A(x), P\ \eta_A(x))}$$

$\boxed{K \overset{\leq}{\rightsquigarrow} \widehat{K}}$

$$\frac{}{\mathsf{type} \overset{\leq}{\rightsquigarrow} \boldsymbol{\lambda}(P, Q_{1\mathrm{f}}, Q_1, Q_{2\mathrm{f}}, Q_2).\ \Pi f_1{:}Q_{1\mathrm{f}}.\, \Pi f_2{:}Q_{2\mathrm{f}}.\, \Pi x{:}P.\, Q_1\ [f_1]\ x \to Q_2\ [f_2]\ x}$$

$$\frac{K \overset{\leq}{\rightsquigarrow} \widehat{K}}{\Pi x{:}A.\, K \overset{\leq}{\rightsquigarrow} \boldsymbol{\lambda}(P, Q_{1\mathrm{f}}, Q_1, Q_{2\mathrm{f}}, Q_2).\ \Pi x{:}A.\, \widehat{K}(P', Q_{1\mathrm{f}}', Q_1', Q_{2\mathrm{f}}', Q_2')}$$
(where, for each P, $P' = P\ \eta_A(x)$)

B.2. **Typing.**

$$\boxed{\Gamma \vdash_\Sigma R^+ \Rightarrow S^- \rightsquigarrow \widehat{R}^-}$$

$$\frac{c{::}S \in \Sigma}{\Gamma \vdash c \Rightarrow S \rightsquigarrow \widehat{c}}\ (\textbf{const}) \qquad \frac{x{::}S{\sqsubset}A \in \Gamma}{\Gamma \vdash x \Rightarrow S \rightsquigarrow \widehat{x}}\ (\textbf{var})$$

$$\frac{\Gamma \vdash R_1 \Rightarrow \Pi x{::}S_2{\sqsubset}A_2.\, S \rightsquigarrow \widehat{R_1} \qquad \Gamma \vdash N_2 \Leftarrow S_2 \rightsquigarrow \widehat{N_2} \qquad [N_2/x]^{\mathrm{s}}_{A_2}\, S = S'}{\Gamma \vdash R_1\ N_2 \Rightarrow S' \rightsquigarrow \widehat{R_1}\ N_2\ \widehat{N_2}}\ (\Pi\textbf{-E})$$

$$\frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2 \rightsquigarrow \widehat{R}}{\Gamma \vdash R \Rightarrow S_1 \rightsquigarrow \pi_1\, \widehat{R}}\ (\wedge\textbf{-E}_1) \qquad \frac{\Gamma \vdash R \Rightarrow S_1 \wedge S_2 \rightsquigarrow \widehat{R}}{\Gamma \vdash R \Rightarrow S_2 \rightsquigarrow \pi_2\, \widehat{R}}\ (\wedge\textbf{-E}_2)$$

$$\boxed{\Gamma \vdash_\Sigma N^+ \Leftarrow S^+ \rightsquigarrow \widehat{N}^-}$$

$$\frac{\Gamma \vdash R \Rightarrow Q' \rightsquigarrow \widehat{R} \qquad \Gamma \vdash Q' \leq Q \rightsquigarrow F}{\Gamma \vdash R \Leftarrow Q \rightsquigarrow F(R, \widehat{R})}\ (\textbf{switch})$$

$$\frac{\Gamma, x{::}S{\sqsubset}A \vdash N \Leftarrow S' \rightsquigarrow \widehat{N}}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x{::}S{\sqsubset}A.\, S' \rightsquigarrow \lambda x.\, \lambda\widehat{x}.\, \widehat{N}}\ (\Pi\textbf{-I})$$

$$\frac{}{\Gamma \vdash N \Leftarrow \top \rightsquigarrow \langle\rangle}\ (\top\textbf{-I}) \qquad \frac{\Gamma \vdash N \Leftarrow S_1 \rightsquigarrow \widehat{N_1} \qquad \Gamma \vdash N \Leftarrow S_2 \rightsquigarrow \widehat{N_2}}{\Gamma \vdash N \Leftarrow S_1 \wedge S_2 \rightsquigarrow \langle \widehat{N}_1, \widehat{N}_2 \rangle}\ (\wedge\textbf{-I})$$

$$\boxed{\Gamma \vdash Q_1^+ \leq Q_2^+ \rightsquigarrow F}$$

$$\frac{Q_1 = Q_2}{\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow \boldsymbol{\lambda}(R, R_1).\ R_1}\ (\textbf{refl})$$

$$\frac{\begin{array}{cc} Q_1 \leq Q' \rightsquigarrow \widehat{Q_1\text{-}Q'} & \Gamma \vdash Q_1 \sqsubset P \Rightarrow \mathsf{sort} \rightsquigarrow \widehat{Q_1} \\ \Gamma \vdash Q' \leq Q_2 \rightsquigarrow F & \Gamma \vdash Q' \sqsubset P \Rightarrow \mathsf{sort} \rightsquigarrow \widehat{Q'} \end{array}}{\Gamma \vdash Q_1 \leq Q_2 \rightsquigarrow \boldsymbol{\lambda}(R, R_1).\ F(R, \widehat{Q_1\text{-}Q'}\ \widehat{Q_1}\ \widehat{Q'}\ R\ R_1)}\ (\textbf{climb})$$

$$\boxed{Q_1^+ \leq Q_2^- \rightsquigarrow \widehat{Q_1\text{-}Q_2}}$$

$$\frac{s_1{\leq}s_2 \in \Sigma}{s_1 \leq s_2 \rightsquigarrow s_1\text{-}s_2} \qquad \frac{Q_1 \leq Q_2 \rightsquigarrow \widehat{Q_1\text{-}Q_2}}{Q_1\ N \leq Q_2\ N \rightsquigarrow \widehat{Q_1\text{-}Q_2}\ N}$$

B.3. **Signatures and Contexts.**

$$\boxed{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}}$$

$$\frac{}{\vdash \cdot\ \mathsf{sig} \rightsquigarrow \cdot} \qquad \frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad \cdot \vdash_{\Sigma^*} K \Leftarrow \mathsf{kind} \qquad a{:}K' \notin \Sigma}{\vdash \Sigma, a{:}K\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}, a{:}K}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad \cdot \vdash_{\Sigma^*} A \Leftarrow \mathsf{type} \qquad c{:}A' \notin \Sigma}{\vdash \Sigma, c{:}A\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}, c{:}A}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad a{:}K \in \Sigma \qquad \cdot \vdash_{\Sigma} L \sqsubset K \overset{\text{form}}{\rightsquigarrow} \widehat{L_{\mathrm{f}}} \qquad K \overset{\text{pred}}{\rightsquigarrow} \widehat{K_{\mathrm{p}}} \qquad s{\sqsubset}a'{::}L' \notin \Sigma}{\vdash \Sigma, s{\sqsubset}a{::}L\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma},\ \widehat{s}{:}K,\ \widehat{s}/i{:}\widehat{L_{\mathrm{f}}}(\widehat{s}),\ s{:}\widehat{K_{\mathrm{p}}}(\widehat{s}, a)}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad c{:}A \in \Sigma \qquad \cdot \vdash_{\Sigma} S \sqsubset A \rightsquigarrow \widehat{S} \qquad c{::}S' \notin \Sigma}{\vdash \Sigma, c{::}S\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}, \widehat{c}{:}\widehat{S}(\eta_A(c))}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma} \qquad s_1{\sqsubset}a{::}L \in \Sigma \qquad s_2{\sqsubset}a{::}L \in \Sigma \qquad a{:}K \in \Sigma \qquad K \overset{\leq}{\rightsquigarrow} \widehat{K}}{\vdash \Sigma, s_1{\leq}s_2\ \mathsf{sig} \rightsquigarrow \widehat{\Sigma}, s_1\text{-}s_2{:}\widehat{K}(a, \widehat{s_1}, s_1, \widehat{s_2}, s_2)}$$

$$\boxed{\vdash_{\Sigma} \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}}$$

$$\frac{}{\vdash \cdot\ \mathsf{ctx} \rightsquigarrow \cdot} \qquad \frac{\vdash \Gamma\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma} \qquad \Gamma \vdash S \sqsubset A \rightsquigarrow \widehat{S}}{\vdash \Gamma, x{::}S{\sqsubset}A\ \mathsf{ctx} \rightsquigarrow \widehat{\Gamma}, x{:}A, \widehat{x}{:}\widehat{S}(\eta_A(x))}$$