A Bidirectional Refinement Type System for LF

William Lovas[1] **Frank Pfenning**[2]

May 2007
CMU-CS-08-129

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

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 several examples in the domain of logics and programming languages.

This document is an extended version of a paper originally presented at the Second International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP '07) and published in Electronic Notes in Theoretical Computer Science [LP08]. It supercedes Technical Report CMU-CS-07-127, which was unpublished.

[1]Email: wlovas@cs.cmu.edu
[2]Email: fp@cs.cmu.edu

Keywords: LF, refinement types, subtyping, dependent types, intersection types

1 Introduction

LF was created as a framework for defining logics [HHP93]. Since its inception, it has been used to formalize reasoning about a number of deductive systems (see [Pfe01] for an introduction). 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, judgements are represented by type families, and many subsyntactic relations and judgemental 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 a notion of adequacy or an implementation strategy. We present here a refinement type system for LF based on the modern *canonical forms* approach, and by doing so we obtain direct proofs of important properties like decidability.

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 [Fre94, Dav05] in only assigning refined types to terms already well-typed in pure LF, ensuring that our extension is conservative.

In the remainder of the paper, we describe our refinement type system alongside several illustrative examples (Section 2). Then we explore its metatheory and give proof sketches of important results, including decidability (Section 3). We note that our approach leads to subtyping only being defined on atomic types, but we show that subtyping at higher types is already present in our system by proving that the usual declarative rules are sound and complete with respect to an intrinsic notion of subtyping (Section 4). Finally, we discuss some related work (Section 5) and summarize our results (Section 6).

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.

$$\begin{array}{rl} R ::= c \mid x \mid R\ N & \text{atomic terms} \\ N, M ::= R \mid \lambda x.\, N & \text{normal terms} \end{array}$$

In this style of presentation, typing is defined bidirectionally by two judgements: $R \Rightarrow A$, 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.

$$\begin{array}{rl} P ::= a \mid P\ N & \text{atomic type families} \\ A, B ::= P \mid \Pi x{:}A.\, B & \text{normal type families} \end{array}$$

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.

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.

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

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.

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 : type.
$zero$: nat.
$succ$: $nat \to nat$.

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

$$\mathit{even} \sqsubset \mathit{nat}.$$
$$\mathit{odd} \sqsubset \mathit{nat}.$$

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:

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

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 *intersection sorts*, written as $S_1 \wedge S_2$.[1]

$$\mathit{succ} :: \mathit{even} \to \mathit{odd} \ \wedge \ \mathit{odd} \to \mathit{even}.$$

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.[2]

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

To show that the declaration for *succ* 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[3], 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.

[1] Intersection has lower precedence than arrow.

[2] 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.

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

$pos \sqsubset nat.$
$succ :: \cdots \wedge \top \rightarrow pos.$

Since we only sort-check well-typed programs and *succ* is declared to have type $nat \rightarrow nat$, 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*.

$odd \leq pos.$

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:

$even \sqsubset nat. \quad odd \sqsubset nat. \quad pos \sqsubset nat.$
$odd \leq pos.$
$zero :: even.$
$succ :: even \rightarrow odd \ \wedge \ odd \rightarrow even \ \wedge \ \top \rightarrow pos.$

Now we should be able to verify that, for example, $succ\ (succ\ zero) \Leftarrow even$. 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 judgements, $N \Leftarrow A$ 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.

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* judgement, and the elimination rules are part of the *synthesis* judgement. 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 *succ* (*succ zero*) $\Leftarrow$ *even*. The context is always empty and therefore omitted. To save space, we abbreviate *even* as *e*, *odd* as *o*, *pos* as *p*, *zero* as *z*, and *succ* as *s*, and we omit reflexive uses of subsorting.

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

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

$$\frac{\vdots \qquad \vdots}{\vdash s\ z \Leftarrow o \qquad \vdash s\ z \Leftarrow p}$$
$$\vdash s\ z \Leftarrow o \wedge p$$

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.

$$\begin{array}{l} \mathit{double} : \mathit{nat} \to \mathit{nat} \to \mathsf{type}. \\ \mathit{dbl\text{-}zero} : \mathit{double}\ \mathit{zero}\ \mathit{zero}. \\ \mathit{dbl\text{-}succ} : \Pi X{:}\mathit{nat}.\, \Pi Y{:}\mathit{nat}.\, \mathit{double}\ X\ Y \to \mathit{double}\ (\mathit{succ}\ X)\ (\mathit{succ}\ (\mathit{succ}\ Y)). \end{array}$$

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.) 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.

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.

$$\begin{array}{l} \mathit{double} \sqsubset \mathit{double} :: \top \to \mathit{even} \to \mathsf{sort}. \\ \mathit{dbl\text{-}zero} :: \mathit{double}\ \mathit{zero}\ \mathit{zero}. \\ \mathit{dbl\text{-}succ} :: \Pi X{::}\top.\, \Pi Y{::}\mathit{even}.\, \mathit{double}\ X\ Y \to \mathit{double}\ (\mathit{succ}\ X)\ (\mathit{succ}\ (\mathit{succ}\ Y)). \end{array}$$

After these declarations, it would be a *sort error* to pose a query such as "?- *double X* (*succ* (*succ* (*succ zero*)))." 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.[4]

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.

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

tp : type. % the type of object language types
$\rightarrow$: $tp \rightarrow tp \rightarrow tp$. % object language function space
%infix *right* 10 $\rightarrow$.

exp : $tp \rightarrow$ type. % the type of expressions
$cmp \sqsubset exp$. % the sort of computations
$val \sqsubset exp$. % the sort of values

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

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

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 $(val\ A \rightarrow cmp\ B)$ 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*.

$eval :: cmp\ A \rightarrow val\ A \rightarrow$ 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 evaluation always returns a value. Moreover, if type reconstruction gives E_1' the "most general" sort $val\ A \rightarrow cmp\ A$, 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).

2.3 A Final Example: The Calculus of Constructions

As a final example, we present the Calculus of Constructions. Usually, there is a great deal of redundancy in its presentation because of repeated constructs at the level of objects, families, and kinds. Using sorts, we can enforce the stratification and write typing rules that are as simple as if we assumed the infamous *type* : *type*.

$$
\begin{array}{ll}
\mathit{term} : \mathsf{type}. & \% \text{ terms at all levels} \\
\\
\mathit{hyp} \sqsubset \mathit{term}. & \% \text{ hyperkinds (the classifier of ``kind'')} \\
\mathit{knd} \sqsubset \mathit{term}. & \% \text{ kinds} \\
\mathit{fam} \sqsubset \mathit{term}. & \% \text{ families} \\
\mathit{obj} \sqsubset \mathit{term}. & \% \text{ objects}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{tp} :: \mathit{hyp} \wedge \mathit{knd}. & \\
\mathit{pi} :: \mathit{fam} \to (\mathit{obj} \to \mathit{fam}) \to \mathit{fam} \wedge & \% \text{ dependent function types, } \Pi x{:}A.\, B \\
\quad \mathit{fam} \to (\mathit{obj} \to \mathit{knd}) \to \mathit{knd} \wedge & \% \text{ type family kinds, } \Pi x{:}A.\, K \\
\quad \mathit{knd} \to (\mathit{fam} \to \mathit{fam}) \to \mathit{fam} \wedge & \% \text{ polymorphic function types, } \forall \alpha{:}K.\, A \\
\quad \mathit{knd} \to (\mathit{fam} \to \mathit{knd}) \to \mathit{knd}. & \% \text{ type operator kinds, } \Pi \alpha{:}K_1.\, K_2 \\
\mathit{lm} :: \mathit{fam} \to (\mathit{obj} \to \mathit{obj}) \to \mathit{obj} \wedge & \% \text{ functions, } \lambda x{:}A.\, M \\
\quad \mathit{fam} \to (\mathit{obj} \to \mathit{fam}) \to \mathit{fam} \wedge & \% \text{ type families, } \lambda x{:}A.\, B \\
\quad \mathit{knd} \to (\mathit{fam} \to \mathit{obj}) \to \mathit{obj} \wedge & \% \text{ polymorphic abstractions, } \Lambda \alpha{:}K.\, M \\
\quad \mathit{knd} \to (\mathit{fam} \to \mathit{fam}) \to \mathit{fam}. & \% \text{ type operators, } \lambda \alpha{:}K.\, A \\
\mathit{ap} :: \mathit{obj} \to \mathit{obj} \to \mathit{obj} \wedge & \% \text{ ordinary application, } M\ N \\
\quad \mathit{fam} \to \mathit{obj} \to \mathit{fam} \wedge & \% \text{ type family application, } A\ M \\
\quad \mathit{obj} \to \mathit{fam} \to \mathit{obj} \wedge & \% \text{ polymorphic instantiation, } M\ [A] \\
\quad \mathit{fam} \to \mathit{fam} \to \mathit{fam}. & \% \text{ type operator instantiation, } A\ B
\end{array}
$$

The typing rules can now be given non-redundantly, illustrating the implicit overloading afforded by the use of intersections. We omit the type conversion rule and auxiliary judgements for brevity.

$$
\begin{array}{l}
\mathit{of} :: \mathit{knd} \to \mathit{hyp} \to \mathsf{sort} \wedge \\
\quad \mathit{fam} \to \mathit{knd} \to \mathsf{sort} \wedge \\
\quad \mathit{obj} \to \mathit{fam} \to \mathsf{sort}.
\end{array}
$$

$$
\mathit{of\text{-}tp} :: \mathit{of}\ \mathit{tp}\ \mathit{tp}.
$$

$$
\begin{array}{l}
\mathit{of\text{-}pi} :: \mathit{of}\ (\mathit{pi}\ T_1\ \lambda x.\, T_2\ x)\ \mathit{tp} \\
\qquad \leftarrow \mathit{of}\ T_1\ \mathit{tp} \\
\qquad \leftarrow (\Pi x{:}\mathit{term}.\, \mathit{of}\ x\ T_1 \to \mathit{of}\ (T_2\ x)\ \mathit{tp}). \\
\mathit{of\text{-}lm} :: \mathit{of}\ (\mathit{lm}\ U_1\ \lambda x.\, T_2\ x)\ (\mathit{pi}\ U_1\ \lambda x.\, U_2\ x) \\
\qquad \leftarrow \mathit{of}\ U_1\ \mathit{tp}
\end{array}
$$

$$\leftarrow (\Pi x{:}\mathit{term}.\, \mathit{of}\ x\ U_1 \to \mathit{of}\ (T_2\ x)\ (U_2\ x)).$$
$$\mathit{of\text{-}ap} :: \mathit{of}\ (\mathit{ap}\ T_1\ T_2)\ (U_1\ T_2)$$
$$\leftarrow \mathit{of}\ T_1\ (\mathit{pi}\ U_2\ \lambda x.\, U_1\ x)$$
$$\leftarrow \mathit{of}\ T_2\ U_2.$$

Intersection types also provide a degree of modularity: by deleting some conjuncts from the declarations of *pi*, *lm*, and *ap* above, we can obtain an encoding of any point on the λ-cube.

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 straightforward proofs; for technically tricky proofs, we include all cases in Appendix B.

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 judgements 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 judgementally 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 judgements defining substitution into atomic terms are the most interesting ones.

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

Judgement:	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: Judgements defining hereditary substitution.

compositionally; the latter is defined by two rules:

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

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

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 judgements are mutually exclusive by examining the head of the input atomic term.

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

Lemma 3.1.

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

Proof. By induction on the given derivation. □

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

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

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

Every other syntactic category's substitution judgement 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.

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

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

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 judgements 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 judgement 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 judgement, 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 judgement $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ 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 judgement 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 judgement $\Gamma \vdash N \Lleftarrow S$ that makes use of the algorithmic synthesis judgement; 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.

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 $\Gamma \vdash \Delta \mathbin{@} N = \Delta'$ judgement 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 $\Gamma \vdash \Delta \ @ \ N = \Delta'$ is always defined and apply Lemma 3.10. □

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

Decidability theorems and proofs for other syntactic categories' formation judgements 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* $\vdash \Sigma$ sig *or* $\nvdash \Sigma$ 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.

Theorem (Substitution). *Suppose* $\Gamma_L \vdash N_0 \Leftarrow S_0$. *Then:*

1. *If*
   - $\vdash \Gamma_L, x_0 :: S_0 \sqsubset A_0, \Gamma_R$ ctx , *and*
   - $\Gamma_L, x_0 :: S_0 \sqsubset A_0, \Gamma_R \vdash S \sqsubset A$, *and*
   - $\Gamma_L, x_0 :: S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S$,

   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.

To prove the Substitution Theorem, we require a lemma about how substitutions compose. The corresponding property for a 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* $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\ N_2 = N_2^{\backprime}$ *and* $x_2 \notin \mathrm{FV}(N_0)$. *Then:*

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

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. (The full proof may be found in Appendix B.1.) □

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. In order to highlight the essential content of the theorem—the part that relates to refinements—we write the core LF assumptions and conclusions in grey, and in the proof itself, we elide reasoning related to these grey assumptions. (It is always either straightforward bookkeeping or follows by analogy with the refinement-related reasoning.)

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*

- $\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S$ *and* $\Gamma_L^*, x_0{:}A_0, \Gamma_R^* \vdash R \Rightarrow A$, *and*
- $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_R = \Gamma_R`$,

then

- $[N_0/x_0]^{s}_{A_0}\, S = S`$ *and* $[N_0/x_0]^{a}_{A_0}\, A = A`$, *and*
- *either*
  - $[N_0/x_0]^{rr}_{A_0}\, R = R`$ *and*
  - $\Gamma_L, \Gamma_R` \vdash R` \Rightarrow S`$ *and* $\Gamma_L^*, (\Gamma_R`)^* \vdash R` \Rightarrow A`$,

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

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 $[N_0/x_0]^{s}_{\alpha_0}$ substitution.

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

(The full proof may be found in Appendix B.2.) □

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*
   - $\Gamma_L \vdash N_0 \Leftarrow S_0$ *and* $\Gamma_L^* \vdash N_0 \Leftarrow A_0$,
   - $\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash S \sqsubset A$ *and* $\Gamma_L^*, x_0{:}A_0, \Gamma_R^* \vdash A \Leftarrow \mathsf{type}$, *and*
   - $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_R = \Gamma_R`$,

   then
   - $[N_0/x_0]^{s}_{A_0}\, S = S`$ *and* $[N_0/x_0]^{a}_{A_0}\, A = A`$, *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

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

Proof. Straightforward induction on Γ_R. □

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

Theorem 3.19 (Substitution). *Suppose* $\Gamma_L \vdash N_0 \Leftarrow S_0$. *Then:*

1. *If*
   - $\vdash \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R\ \mathsf{ctx}$, *and*
   - $\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash S \sqsubset A$, *and*
   - $\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S$,

   then
   - $[N_0/x_0]^{\gamma}_{A_0}\ \Gamma_R = \Gamma'_R$ *and* $\vdash \Gamma_L, \Gamma'_R\ \mathsf{ctx}$, *and*
   - $[N_0/x_0]^{\mathrm{s}}_{A_0}\ S = S'$ *and* $[N_0/x_0]^{\mathrm{a}}_{A_0}\ A = A'$ *and* $\Gamma_L, \Gamma'_R \vdash S' \sqsubset A'$, *and*
   - $[N_0/x_0]^{\mathrm{n}}_{A_0}\ N = N'$ *and* $\Gamma_L, \Gamma'_R \vdash N' \Leftarrow S'$,
2. *If*
   - $\vdash \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R\ \mathsf{ctx}$ *and*
   - $\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S$,

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

and similarly for other syntactic categories.

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

Having proven Substitution, we henceforth tacitly assume that all subjects of a judgement are sufficiently well-formed for the judgement 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 whetever 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.[5]

In stating this lemma, we require a judgement that predicts the simple type output of "rn" substitution. This judgement 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 judgement", 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.

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

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. (The full proof may be found in Appendix B.3.) □

Note: By considering the variable being substituted for to be a bound variable subject to α-conversion[6], 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.

[5] 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.

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

$\boxed{S_1 \leq S_2}$

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

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

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

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

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

Figure 1: Derived rules for subsorting at higher sorts.

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. (The full proof may be found in Appendix B.4.) □

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). □

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 Fig. 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 Fig. 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: Fig. 1 accounts for everything covered intrinsically by η-expansion. By the end of this section, we will have shown the following theorems:

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

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

Soundness is a straightforward inductive argument.

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

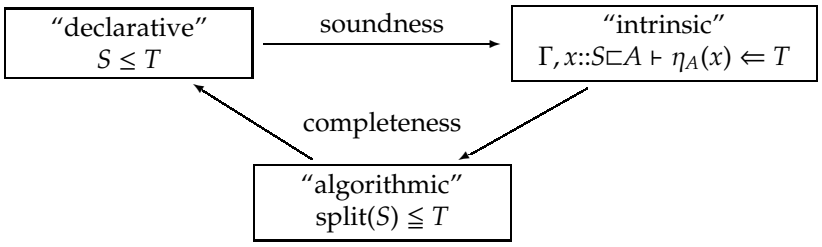
Proof. By induction, making use of the alternate formulations given by Theorem 4.1. □

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 judgements $\Delta \leqq S$ 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.

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

Theorem (Algorithmic $\Rightarrow$ Declarative). *If* $\mathrm{split}(S) \leqq T$, *then* $S \leq T$.

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



As mentioned above, the algorithmic subsorting system system is characterized by two judgements: $\Delta \leqq S$ 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 judgement $\Delta \leqq S$, made precise below, is roughly that the intersection of all the sorts in Δ is a subsort of the sort S.

$\boxed{\Delta \leqq S}$

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

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

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

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

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

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

Figure 2: Algorithmic subsorting.

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

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

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

The rule for checking whether Δ is a subsort of a function type makes use of the application judgement $\Delta \mathbin{@} x{::}\Delta_1 \sqsubset A_1 = \Delta_2$ 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 judgement 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. $S \leq \bigwedge(\text{split}(S))$.

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. □

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. (The full proof may be found in Appendix B.5.) □

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.

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

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. (The full proof may be found in Appendix B.6.) □

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* $\Gamma \vdash \eta_A(R) \Lleftarrow S$.

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 $\Gamma \nvdash \eta_{A_1}(x) \Leftarrow S_1$ into a derivation of $\Delta_1 \not\leqq S_1$.

(The full proof may be found in Appendix B.7.) □

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 Related Work

The works most closely related to ours extend various LF-like dependent type theories with subtyping or intersection types, and we treat these first.

Pfenning described in an unrefereed workshop paper a proposed extension of LF with refinement types [Pfe93]. The present work can be seen as a reconstruction, reformulation, and extension of his ideas, with a focus on canonical forms, decidability, and good proof-theoretic properties.

Aspinall and Compagnoni [AC01] studied a type theory $\lambda P_{\leq}$ with both dependent types and subtyping, but they treated subtyping directly rather than introducing a refinement layer. Their chief difficulty was breaking the cycle that arises between subtyping, kinding, and typing in order to show decidability, which they did by splitting ordinary β-reduction into two levels, one that reduces terms and one that reduces types. In our setting, the restriction of attention to canonical forms obviates the need to consider β-reduction and its properties (e.g. subject reduction, Church-Rosser, etc.) at the cost of a more involved Substitution theorem, an arguably simpler development.

Aspinall [Asp00] also studied an unconventional system of subtyping with dependent types using "power types", a type-level analogue of power sets. Aspinall's system λ_{Power} has uniform "subtyping" at all levels since power "types" can in fact classify type families; although the system remains predicative, this generalization complicates the system's metatheory. Aspinall's use of a "rough typing" judgement in formulating the metatheory of λ_{Power} is somewhat related to our use of simple types in the metatheory of hereditary substitution and η-expansion.

Both Aspinall and Compagnoni's $\lambda P_{\leq}$ and Aspinall's λ_{Power} are more general than LFR in a certain sense, since they allow subtyping declarations between atomic families whose arities and indices might be different. So far in the development of LFR, no examples have wanted for such declarations. The primary shortcoming of both $\lambda P_{\leq}$ and λ_{Power} is their lack of intersection types, which are essential for even the simplest of our examples.

Kopylov [Kop03] studied a dependent intersection $\bigwedge x{::}A.\, B$, a generalization of ordinary intersection $A \wedge B$ where the second type may depend on the element that has both types.[7] His motivation was finding a simple way to define dependent records in NuPRL in terms of only single-field records (following Reynolds's trick in the design of Forsythe [Rey96]). It is tempting to consider a dependent intersection sort $\bigwedge x{::}S \sqsubset A.\, T$ generalizing our $S \wedge T$, but it turns out not to fit in the refinement framework: the sorts S and T must

[7] Kopylov wrote $x{:}A \cap B$ and $A \cap B$.

both refine the same type A, but this precludes T from depending on x; in other words, a dependent intersection would always be degenerate.

There is also a great deal of work that is only tangentially related, which we describe below in order to place the current work in its historical context.

Regular tree types and order-sorted logics for logic programming Types naturally arise in logic programming and automated theorem proving as a way to curtail meaningless search. For example, given a clause $\forall n.\, nat(n) \to \cdots \to prime(n)$, one may end up searching for a proof of $nat(\text{Peter}) \to \cdots \to prime(\text{Peter})$ after instantiating n; even though this search will never succeed, since it is not the case that $nat(\text{Peter})$, it would be better to avoid such meaningless search in the first place. This observation leads to the introduction of *order-sorted logics*, from which we borrow the term "sort". The clause above might be rewritten as $\forall n : nat.\ \cdots \to prime(n)$, capturing the appropriate constraint statically.

One class of types that has proved particularly profitable is the *regular tree types* [DZ92] (see [YFS92] for example), so-called due to their connection with regular tree grammars. A key property that makes the regular tree types useful is the existence of computable intersections, and this property eventually led to the introduction of intersections in the context of refinement type systems for functional languages, described below.

Among order-sorted logics are systems with "term declarations" [SS89]. Term declarations have the form $\forall x_1{::}S_1 \ldots \forall x_n{::}S_n.\, M : S$, meaning that in any context extending $x_1{::}S_1, \ldots, x_n{::}S_n$ the term M can be judged to have sort S. For instance, one might declare

$$\forall x{::}even.\, succ\ x :: odd.$$
$$\forall x{::}odd.\, succ\ x :: even.$$

to achieve roughly the same effect as our $succ :: even \to odd \wedge odd \to even$. One problem with such systems is that they fail to give first-class status to the notion that a term can have multiple sorts, like our intersection sorts do. Furthermore, the typechecking problem for systems with term declarations is tricky at best, since it requires the use of higher-order matching, a problem whose status was until recently open and for which no practical implementation currently exists.

Intersection types Intersection types were originally introduced by Coppo et al. [CDCV81] to describe a type theory in which types are preserved not only by reduction but also by convertibility, i.e. in which subject *expansion* holds in addition to subject *reduction*. Since β-conversion preserves types, they were able to precisely characterize the class of normalizing terms as those that have a non-trivial (essentially, non-$\top$) type.

Later, Reynolds used intersection types to simplify the design of the programming language Forsythe [Rey96], e.g. by representing n-ary records as

n-ary intersections of single-field records. Although Reynolds's setting of imperative programming was vastly different from our world of logical frameworks, many of his core motivations were similar to ours: namely, intersections can be used to capture multiple properties of individual terms. Furthermore, despite his working under assumptions and constraints quite different from ours, one can see shades of many of his ideas reflected in our development.

Refinement types for functional languages The utility of regular tree types in logic programming led Freeman to investigate them for functional languages [Fre94]. Freeman studied a system of refinement types for a fragment of ML based on ideas relating to regular trees, but intersection types were also a crucial addition for many of his examples. His focus was on maintaining decidable inference with minimal declarations, but ultimately the theory fell prey to algorithmic efficiency issues.

Later, Davies sought to tame the complexities and make refinement types practical for Standard ML [Dav05]. Davies's key decision was to abandon pure inference of unannotated programs in favor of a bidirectional type system and minimally annotated programs. The focus then was on minimizing the annotation burden, a task somewhat alleviated by the apparent positive benefits of annotations during program construction and their interpretation as machine-checked documentation. Along the way, Davies discovered a curious interaction of intersection types with effects quite similar to the interaction of polymorphism with effects; the solution was to impose a *value restriction* similar to the one familiar from ML-style let-polymorphism [DP00]. Additionally, the system had to be weakened by the removal of the $\wedge/\rightarrow$ distributivity subtyping rule, which could allow users to circumvent the value restriction.

Dunfield unified Davies work with a form of dependent typing inspired by Xi's Dependent ML [XP99], and extended the resulting system with the "indefinite" union and existential types [Dun07]. Dunfield abandoned the refinement restriction, studying a type system with arbitrary intersections and unions directly at the type level, but he maintained a bidirectional typing discipline. To properly account for the indefinite property types, he extended bidirectional checking with an evaluation order-directed "tridirectional rule" [DP04].

All of this work was in the context of functional programming, and thus quite different from our work in logical frameworks. Obviously, since LF has no side effects—indeed no reduction at all!—we have no need of Davies's value restriction, and since we treat only negative types, we have no need of Dunfield's tridirectional rule. But there are still some similarities that help guide the present work, such as Freeman's and Davies's refinement restriction and Davies's and Dunfield's bidirectional typing.

Subtler differences arise from different assumptions about the world at typechecking time. The work on functional languages is all concerned with typing closed terms, and datatypes embody a closed-world assumption, both of which can be leveraged to reason about things like the emptiness of the intersection $\mathit{even} \wedge \mathit{odd}$. In our setting, though, we have an inherently open-

ended signature and we work under non-empty contexts: it is impossible to show that $even \wedge odd$ is empty because one might always have an assumption of type $even \wedge odd$.[8] However, it is exactly this open-endedness, coupled with LF's very weak, purely representational function space, that allows us to show such strong theorems as the soundness and completeness of subtyping via η-expansion, a theorem one would not expect to hold generally in the presence of a large, computational function space.

6 Summary

In summary, we have exhibited a variant of the logical framework LF with a notion of subtyping based on refinement types. We have demonstrated the expressive power of this extension through a number of realistic examples, and we have shown several metatheoretic properties critical to its utility as a logical framework, including decidability of typechecking.

Our development was drastically simplified by the decision to admit only canonical forms. One effect of this choice was that subsorting was only required to be judgementally defined at base sorts; higher-sort subsorting was derived through an η-expansion-based definition with respect to which we showed the usual structural subsorting rules both sound and complete.

There are a number of avenues of future exploration. For one, it is unclear how subsorting and intersection sorts will interact with the typical features of a metalogical framework, including type reconstruction, unification, and proof search, to name a few; these questions will have to be answered before refinement types can be integrated into a practical implementation. It is also worthwhile to consider adapting the refinement system to more expressive frameworks, like the Linear Logical Framework or the Concurrent Logical Framework.

References

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

[Asp00] David Aspinall. Subtyping with power types. In Peter Clote and Helmut Schwichtenberg, editors, *CSL*, volume 1862 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2000.

[CDCV81] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 27(2-6):45–58, 1981.

[8] One might imagine extending LFR with declarations of the form $even \wedge odd \leq empty$ to allow the user to capture this property explicitly. As currently specified, LFR does not give the user the ability to define any arbitrary subtyping lattice since it excludes such declarations.

[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.

[DP00] Rowan Davies and Frank Pfenning. Intersection types and computational effects. In P. Wadler, editor, *Proceedings of the Fifth International Conference on Functional Programming (ICFP'00)*, pages 198–208, Montreal, Canada, September 2000. ACM Press.

[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 Pfenning [Pfe92], pages 157–187.

[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.

[Kop03] Alexei Kopylov. Dependent intersection: A new way of defining records in type theory. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 86–95. IEEE Computer Society, 2003.

[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.

[LP08] William Lovas and Frank Pfenning. A bidirectional refinement type system for LF. *Electronic Notes in Theoretical Computer Science*, 196:113–128, January 2008.

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

[Pfe92] Frank Pfenning, editor. *Types in Logic Programming*. MIT Press, Cambridge, Massachusetts, 1992.

[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.

[Pfe01] 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.

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

[SS89] Manfred Schmidt-Schauß. *Computational Aspects of an Order-sorted Logic with Term Declarations*. Number 395 in Lecture Notes in Computer Science. Springer, 1989.

[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.

[XP99] Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In A. Aiken, editor, *Conference Record of the 26th Symposium on Principles of Programming Languages (POPL'99)*, pages 214–227. ACM Press, January 1999.

[YFS92] Eyal Yardeni, Thom Frühwirth, and Ehud Shapiro. Polymorphically typed logic programs. In Pfenning [Pfe92], pages 63–90.

A Complete Rules

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

A.1 Grammar

Kind level

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

Type level

$P ::= a \mid P\ N$	atomic type families
$A ::= P \mid \Pi x{:}A_1.\, A_2$	canonical type families
$Q ::= s \mid Q\ N$	atomic sort families
$S ::= Q \mid \Pi x{::}S_1 \sqsubset A_1.\, S_2 \mid \top \mid S_1 \wedge S_2$	canonical sort families

Term level

$R ::= c \mid x \mid R\ N$	atomic terms
$N ::= R \mid \lambda x.\, N$	canonical terms

Signatures and contexts

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

A.2 Expansion and Substitution

All bound variables are tacitly assumed to be sufficiently fresh.

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

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

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

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

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

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

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

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

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

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

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

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

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

A.3 Kinding

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash Q \sqsubset P' \Rightarrow L \qquad P' = P \qquad L = \mathsf{sort}}{\Gamma \vdash Q \sqsubset P} \qquad \frac{\Gamma \vdash S \sqsubset A \qquad \Gamma, x{::}S{\sqsubset}A \vdash S' \sqsubset A'}{\Gamma \vdash \Pi x{::}S{\sqsubset}A.\, S' \sqsubset \Pi x{:}A.\, A'}$$

$$\frac{}{\Gamma \vdash \top \sqsubset A} \qquad \frac{\Gamma \vdash S_1 \sqsubset A \qquad \Gamma \vdash S_2 \sqsubset A}{\Gamma \vdash S_1 \wedge S_2 \sqsubset A}$$

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

A.4 Typing

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

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

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

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

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

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

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

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

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

A.5 Signatures and Contexts

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

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

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

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

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

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

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

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

B Full Proofs

B.1 Lemma 3.13 (Composition of Substitutions)

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

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

and similarly for other syntactic categories.

Proof. By lexicographic induction on the unordered pair of α_0 and α_2, and on the first substitution derivation in each clause.

Not all clauses' proofs need be mutually inductive—the four given cases can be proven independently of the ones elided. We give only the proof for the four given cases; the rest are straightforward.

1. Suppose $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N_2 = N_2'$
   and $\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} N = N'$
   and $\mathcal{E} :: [N_2/x_2]^{\mathrm{n}}_{\alpha_2} N = N''$.
   (We want to show: $[N_2'/x_2]^{\mathrm{n}}_{\alpha_2} N' = N'''$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'' = N'''$.)

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', a)}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0} R = N'}$

   $\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2} R = R''}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2} R = R''}$ By inversion, using Lemma 3.1.

   $[N_2'/x_2]^{\mathrm{n}}_{\alpha_2} N' = N'''$ and $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R'' = (N''', a)$ By i.h. (4) on $\mathcal{D}_1$.

   $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} R'' = N'''$ By rule.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{rr}}_{\alpha_0} R = R'}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0} R = R'}$

   $\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rn}}_{\alpha_2} R = (N'', a)}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2} R = N''}$ or $\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2} R = R''}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2} R = R''}$ By inversion.

Subcase: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, R = (N', a)}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, R = N'}$

$[N_2`/x_2]^{\mathrm{rn}}_{\alpha_2}\, R` = (N`', a)$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N' = N`'$ By i.h. (3) on $\mathcal{D}_1$.

$[N_2`/x_2]^{\mathrm{n}}_{\alpha_2}\, R` = N`'$ By rule.

Subcase: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R = R'}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, R = R'}$

$[N_2`/x_2]^{\mathrm{rr}}_{\alpha_2}\, R` = R`'$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R' = R`'$ By i.h. (2) on $\mathcal{D}_1$.

$[N_2`/x_2]^{\mathrm{n}}_{\alpha_2}\, R` = R`'$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, R' = R`'$ By rule.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N = N`}{[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, \lambda x.\, N = \lambda x.\, N`}$

$\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, N = N'}{[N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, \lambda x.\, N = \lambda x.\, N'}$ By inversion.

$[N_2`/x_2]^{\mathrm{n}}_{\alpha_2}\, N` = N`'$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N' = N`'$ By i.h. (1) on $\mathcal{D}_1$.

$[N_2`/x_2]^{\mathrm{n}}_{\alpha_2}\, \lambda x.\, N` = \lambda x.\, N`'$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, \lambda x.\, N' = \lambda x.\, N`'$ By rule.

2. Suppose $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2`$ and $\mathcal{D} :: [N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R`$ and $\mathcal{E} :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R = R'$. (We want to show: $[N_2`/x_2]^{\mathrm{rr}}_{\alpha_2}\, R` = R`'$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R' = R`'$.)

Case: $\mathcal{D} = \dfrac{x \neq x_0}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, x = x}$

$\mathcal{E} = \dfrac{x \neq x_2}{[N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, x = x}$ By inversion.

$[N_2`/x_2]^{\mathrm{rr}}_{\alpha_2}\, x = x$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, x = x$ By rule.

Case: $\mathcal{D} = \dfrac{}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, c = c}$

$\mathcal{E} = \dfrac{}{[N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, c = c}$ By inversion.

$[N_2`/x_2]^{\mathrm{rr}}_{\alpha_2}\, c = c$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, c = c$ By rule.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3 = R_3' \qquad \mathcal{D}_2 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_4 = N_4'}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3\ N_4 = R_3'\ N_4'}$

$$\mathcal{E} = \frac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R_3 = R_3' \qquad \mathcal{E}_2 :: [N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, N_4 = N_4'}{[N_2/x_2]^{\mathrm{rr}}_{\alpha_2}\, R_3\ N_4 = R_3'\ N_4'}$$

By inversion.

$[N_2'/x_2]^{\mathrm{rr}}_{\alpha_2}\, R_3' = R_3''$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3' = R_3''$ By i.h. (2) on $\mathcal{D}_1$.
$[N_2'/x_2]^{\mathrm{n}}_{\alpha_2}\, N_4' = N_4''$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_4' = N_4''$ By i.h. (1) on $\mathcal{D}_2$.
$[N_2'/x_2]^{\mathrm{rr}}_{\alpha_2}\, R_3'\ N_4' = R_3''\ N_4''$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3'\ N_4' = R_3''\ N_4''$ By rule.

3. Suppose $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2'$ and $\mathcal{D} :: [N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R = R'$ and $\mathcal{E} :: [N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, R = (N', \beta)$.
(We want to show: $[N_2'/x_2]^{\mathrm{rn}}_{\alpha_2}\, R' = (N'', \beta)$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N' = N''$.)

Case: $\mathcal{D} = \dfrac{x \neq x_0}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, x = x}$, where $R' = x$

$$\mathcal{E} = \frac{}{[N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, x_2 = (N_2, \alpha_2)}\text{, where } N' = N_2 \text{ and } x = x_2$$

By inversion.

$[N_2'/x_2]^{\mathrm{rn}}_{\alpha_2}\, x_2 = (N_2', \alpha_2)$ By rule.
$[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_2 = N_2'$ By assumption.

Case: $\mathcal{D} = \dfrac{}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, c = c}$, where $R = c$

Impossible: no rule can conclude $\mathcal{E} :: [N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, c = (N', \beta)$.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3 = R_3' \qquad \mathcal{D}_2 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, N_4 = N_4'}{[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\, R_3\ N_4 = R_3'\ N_4'}$

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, R_3 = (\lambda x.\, N_3', \alpha_4 \to \alpha_3) \\ \mathcal{E}_2 :: [N_2/x_2]^{\mathrm{n}}_{\alpha_2}\, N_4 = N_4' \qquad \mathcal{E}_3 :: [N_4'/x]^{\mathrm{n}}_{\alpha_4}\, N_3' = \hat{N}_3'\end{array}}{[N_2/x_2]^{\mathrm{rn}}_{\alpha_2}\, R_3\ N_4 = (N_3', \alpha_3)}$$

By inversion.

We need to show: $[N_2'/x_2]^{\mathrm{rn}}_{\alpha_2}\, R_3'\ N_4' = (\hat{N}_3'', \alpha_3)$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, \hat{N}_3' = \hat{N}_3''$.

$[N_2'/x_2]^{\mathrm{rn}}_{\alpha_2}\, R_3' = (\lambda x.\, N_3'', \alpha_4 \to \alpha_3)$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0}\, \lambda x.\, N_3' = \lambda x.\, N_3''$
By i.h. (3) on $\mathcal{D}_1$.

$[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'_3 = N^{\backprime}_3$ By inversion.
$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_4^{\backprime} = N_4^{\backprime\backprime}$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'_4 = N^{\backprime\backprime}_4$ By i.h. (1) on $\mathcal{D}_2$.

$[N_4^{\backprime\backprime}/x]^{\mathrm{n}}_{\alpha_4} N_3^{\backprime\backprime} = \hat{N}_3^{\backprime\backprime}$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} \hat{N}'_3 = \hat{N}^{\backprime\backprime}_3$
By i.h. (1) on (α_0, α_4), using
$[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'_4 = N^{\backprime\backprime}_4$,
$[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'_3 = N^{\backprime\backprime}_3$,
and $[N'_4/x]^{\mathrm{n}}_{\alpha_4} N'_3 = \hat{N}'_3$

$[N_2^{\backprime}/x_2]^{\mathrm{rn}}_{\alpha_2} R_3^{\backprime} N_4^{\backprime} = (\hat{N}_3^{\backprime\backprime}, \alpha_3)$ By rule, using
$[N_2^{\backprime}/x_2]^{\mathrm{rn}}_{\alpha_2} R'_3 = (\lambda x.\, N_3^{\backprime\backprime}, \alpha_4 \to \alpha_3)$,
$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_4^{\backprime} = N_4^{\backprime\backprime}$,
and $[N_4^{\backprime\backprime}/x]^{\mathrm{n}}_{\alpha_4} N_3^{\backprime\backprime} = \hat{N}_3^{\backprime\backprime}$.

4. Suppose $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N_2 = N_2^{\backprime}$ (We want to show:
and $\mathcal{D} :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N^{\backprime}, \beta)$ $[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N^{\backprime} = N^{\backprime\backprime}$
and $\mathcal{E} :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2} R = R'$. and $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R' = (N^{\backprime\backprime}, \beta)$.)

Case: $\mathcal{D} = \dfrac{}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} x_0 = (N_0, \alpha_0)}$, where $N^{\backprime} = N_0$

$\mathcal{E} = \dfrac{x_0 \neq x_2}{[N_2/x_2]^{\mathrm{rr}}_{\alpha_2} x_0 = x_0}$, where $R' = x_0$ By inversion.

$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_0 = N_0$ By trivial substitution, since $x_2 \notin \mathrm{FV}(N_0)$.
$[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} x_0 = (N_0, \alpha_0)$ By rule.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R_3 = (\lambda x.\, N_3, \alpha_4 \to \alpha_3) \quad \mathcal{D}_2 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} N_4 = N_4^{\backprime} \quad \mathcal{D}_3 :: [N_4^{\backprime}/x]^{\mathrm{n}}_{\alpha_4} N_3 = N_3^{\backprime}}{[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R_3\, N_4 = (N_3^{\backprime}, \alpha_3)}$

$\mathcal{E} = \dfrac{\mathcal{E}_1 :: [N_2/x_2]^{\mathrm{rr}}_{\alpha_2} R_3 = R'_3 \quad \mathcal{E}_2 :: [N_2/x_2]^{\mathrm{n}}_{\alpha_2} N_4 = N'_4}{[N_2/x_2]^{\mathrm{rr}}_{\alpha_2} R_3\, N_4 = R'_3\, N'_4}$
By inversion.

We need to show: $[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_3^{\backprime} = N_3^{\backprime\backprime}$ and $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R'_3\, N'_4 = (N_3^{\backprime\backprime}, \alpha_3)$

$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} \lambda x.\, N_3 = \lambda x.\, N'_3$ and $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R'_3 = (\lambda x.\, N'_3, \alpha_4 \to \alpha_3)$
By i.h. (4) on $\mathcal{D}_1$.
$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_3 = N'_3$ By inversion.
$[N_2^{\backprime}/x_2]^{\mathrm{n}}_{\alpha_2} N_4^{\backprime} = N_4^{\backprime\backprime}$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N'_4 = N^{\backprime\backprime}_4$
By i.h. (1) on $\mathcal{D}_2$.

$[N_4\text{`}/x]^{\mathrm{n}}_{\alpha_4} N_3' = N_3\text{`}$ and $[N_2\text{`}/x_2]^{\mathrm{n}}_{\alpha_2} N_3\text{`} = N_3\text{`}$
By i.h. (1) on (α_2, α_4), using
$[N_2\text{`}/x_2]^{\mathrm{n}}_{\alpha_2} N_4\text{`} = N_4\text{`}$,
$[N_2\text{`}/x_2]^{\mathrm{n}}_{\alpha_2} N_3 = N_3'$,
and $\mathcal{D}_3 :: [N_4\text{`}/x]^{\mathrm{n}}_{\alpha_4} N_3 = N_3\text{`}$.

$[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R_3' \; N_4' = (N_3\text{`}, \alpha_3)$
By rule, using
$[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R_3' = (\lambda x.\, N_3', \alpha_4 \to \alpha_3)$
$[N_0/x_0]^{\mathrm{n}}_{\alpha_0} N_4' = N_4\text{`}$,
and $[N_4\text{`}/x]^{\mathrm{n}}_{\alpha_4} N_3' = N_3\text{`}$.
□

B.2 Theorem 3.15 (Proto-Substitution, terms)

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*
   - $[N_0/x_0]^{\gamma}_{A_0} \Gamma_{\mathrm{R}} = \Gamma_{\mathrm{R}}\text{`}$, *and*
   - $[N_0/x_0]^{\mathrm{s}}_{A_0} S = S\text{`}$ *and* $[N_0/x_0]^{\mathrm{a}}_{A_0} A = A\text{`}$,

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

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

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

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

1. Suppose $\Gamma_L \vdash N_0 \Leftarrow S_0$
   and $\mathcal{D} :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S$
   and $[N_0/x_0]^{\gamma}_{A_0}\ \Gamma_R = \Gamma_R`$
   and $[N_0/x_0]^{s}_{A_0}\ S = S`$.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow Q_1 \qquad \mathcal{D}_2 :: Q_1 \leq Q}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Leftarrow Q}$

   $[N_0/x_0]^{s}_{A_0}\ Q_1 = Q_1`$ and either
   $([N_0/x_0]^{rr}_{A_0}\ R = R`$ and $\Gamma_L, \Gamma_R` \vdash R` \Rightarrow Q_1`)$, or
   $([N_0/x_0]^{rn}_{A_0}\ R = (N`, (P_1`)^-)$ and $\Gamma_L, \Gamma_R` \vdash N` \Leftarrow Q_1`)$
   By i.h. (2) on $\mathcal{D}_1$.

   $S` = Q`$ and $[N_0/x_0]^{q}_{A_0}\ Q = Q`$ By inversion.
   $Q_1` \leq Q`$ By Lemma 3.14 (Substitution into Subsorting).

   Subcase: $[N_0/x_0]^{rr}_{A_0}\ R = R`$ and $\Gamma_L, \Gamma_R` \vdash R` \Rightarrow Q_1`$
   $[N_0/x_0]^{n}_{A_0}\ R = R`$ By rule **subst-n-atom**.
   $\Gamma_L, \Gamma_R` \vdash R` \Leftarrow Q`$ By rule **switch**.

   Subcase: $[N_0/x_0]^{rn}_{A_0}\ R = (N`, (P_1`)^-)$ and $\Gamma_L, \Gamma_R` \vdash N` \Leftarrow Q_1`$
   $N` = R`$ and $\Gamma_L, \Gamma_R` \vdash R` \Rightarrow Q_2`$ and $Q_2` \leq Q_1`$ By inversion.
   $(P_1`)^- = a$, for some a By definition.
   $[N_0/x_0]^{n}_{A_0}\ R = R`$ By rule **subst-n-atom-norm**.
   $Q_2` \leq Q`$ By rule.
   $\Gamma_L, \Gamma_R` \vdash R` \Leftarrow Q`$ By rule **switch**.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R, x{::}S_1 \sqsubset A_1 \vdash N \Leftarrow S_2}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash \lambda x.\, N \Leftarrow \Pi x{::}S_1 \sqsubset A_1.\, S_2}$

   $S` = \Pi x{::}S_1` \sqsubset A_1`.\, S_2`$ and $[N_0/x_0]^{s}_{A_0}\ S_1 = S_1`$ and
   $[N_0/x_0]^{a}_{A_0}\ A_1 = A_1`$ and $[N_0/x_0]^{s}_{A_0}\ S_2 = S_2`$ By inversion.
   $[N_0/x_0]^{\gamma}_{A_0}\ \Gamma_R, x{::}S_1 \sqsubset A_1 = \Gamma_R`, x{::}S_1` \sqsubset A_1`$ By rule.
   $[N_0/x_0]^{n}_{A_0}\ N = N`$ and $\Gamma_L, \Gamma_R`, x{::}S_1` \sqsubset A_1` \vdash N` \Leftarrow S_2`$
   By i.h. (1) on $\mathcal{D}_1$.
   $[N_0/x_0]^{n}_{A_0}\ \lambda x.\, N = \lambda x.\, N`$ By rule.
   $\Gamma_L, \Gamma_R` \vdash \lambda x.\, N` \Leftarrow \Pi x{::}S_1` \sqsubset A_1`.\, S_2`$ By rule.

   Case: $\mathcal{D} = \dfrac{\begin{array}{l}\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S_1 \\ \mathcal{D}_2 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S_2\end{array}}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N \Leftarrow S_1 \wedge S_2}$

$S` = S`_1 \wedge S`_2$ and $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S_1 = S`_1$ and $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S_2 = S`_2$
By inversion.

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, N = N`_1$ and $\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash N`_1 \Leftarrow S`_1$ By i.h. (1) on $\mathcal{D}_1$.

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, N = N`_2$ and $\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash N`_2 \Leftarrow S`_2$ By i.h. (1) on $\mathcal{D}_2$.

$N`_1 = N`_2$ By Theorem 3.2 (Functionality of Substitution).

Let $N` = N`_1 = N`_2$: $\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash N` \Leftarrow S_1 \wedge S_2$ By rule.

Case: $\mathcal{D} = \dfrac{}{\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash N \Leftarrow \top}$

$[N_0/x_0]^{\mathrm{n}}_{A_0}\, N = N`$ By core LF Proto-Substitution Theorem.

$\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash N` \Leftarrow \top$ By rule.

Note: This case is where we make use of the three grey assumptions to clause 1. The remainder of the grey assumptions and conclusions are only required to ensure that these three key assumptions are satisfied on every inductive appeal. (The interested reader may gain great insight into the essential difficulty of the proof by tracing these dependencies; all of the action is in the "application" case of clause 2.)

2. Suppose $\Gamma_{\mathrm{L}} \vdash N_0 \Leftarrow S_0$ and $\mathcal{D} :: \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash R \Rightarrow S$ and $[N_0/x_0]^{\gamma}_{A_0}\, \Gamma_{\mathrm{R}} = \Gamma`_{\mathrm{R}}$.

Case: $\mathcal{D} = \dfrac{c{:}S \in \Sigma}{\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash c \Rightarrow S}$

$\mathrm{FV}(S) = \emptyset$ By signature well-formedness.

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S$ By trivial substitution.

$[N_0/x_0]^{\mathrm{rr}}_{A_0}\, c = c$ By rule.

$\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash c \Rightarrow S$ By rule.

Case: $\mathcal{D} = \dfrac{x{::}S \sqsubset A \in \Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}}}{\Gamma_{\mathrm{L}}, x_0{::}S_0 \sqsubset A_0, \Gamma_{\mathrm{R}} \vdash x \Rightarrow S}$

Subcase: $x{::}S \sqsubset A \in \Gamma_{\mathrm{L}}$

$x_0 \notin \mathrm{FV}(S)$ and $x_0 \neq x$ By α-conversion convention.

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, S = S$ By trivial substitution.

$[N_0/x_0]^{\mathrm{rr}}_{A_0}\, x = x$ By rule.

$\Gamma_{\mathrm{L}}, \Gamma`_{\mathrm{R}} \vdash x \Rightarrow S$ By rule.

Subcase: $x{::}S \sqsubset A = x_0{::}S_0 \sqsubset A_0$

$x_0 \notin \mathrm{FV}(S_0)$ By α-conversion convention.

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, S_0 = S_0$ By trivial substitution.

$[N_0/x_0]^{\mathrm{rn}}_{A_0}\, x_0 = (N_0, (A_0)^-)$ By rule.

$\Gamma_L \vdash N_0 \Leftarrow S_0$ By assumption.
$\Gamma_L, \Gamma_R` \vdash N_0 \Leftarrow S_0$ By weakening.

Subcase: $x{::}S \sqsubset A \in \Gamma_R$
$[N_0/x_0]^{\gamma}_{A_0}\ \Gamma_R = \Gamma_R`$ By assumption.
$x{::}S` \sqsubset A` \in \Gamma_R`$ and $[N_0/x_0]^{s}_{A_0}\ S = S`$ By inversion.
$x_0 \neq x$ By α-conversion convention.
$[N_0/x_0]^{rr}_{A_0}\ x = x$ By rule.
$\Gamma_L, \Gamma_R` \vdash x \Rightarrow S`$ By rule.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R_1 \Rightarrow \Pi x{::}S_2 \sqsubset A_2.\ S_1 \quad \mathcal{D}_2 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash N_2 \Leftarrow S_2 \quad \mathcal{D}_3 :: [N_2/x]^{s}_{A_2}\ S_1 = S_1'}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R_1\ N_2 \Rightarrow S_1'}$

$[N_0/x_0]^{s}_{A_0}\ \Pi x{::}S_2 \sqsubset A_2.\ S_1 = S`$ and either
$([N_0/x_0]^{rr}_{A_0}\ R_1 = R_1`$ and $\Gamma_L, \Gamma_R` \vdash R_1` \Rightarrow S`)$, or
$([N_0/x_0]^{rn}_{A_0}\ R_1 = (N_1`, (\Pi x{:}A_2`.\ A_1`)^-)$ and $\Gamma_L, \Gamma_R` \vdash N_1` \Leftarrow S`)$
By i.h. (2) on $\mathcal{D}_1$.
$S` = \Pi x{::}S_2` \sqsubset A_2`.\ S_1`$
and $[N_0/x_0]^{s}_{A_0}\ S_2 = S_2`$ and $[N_0/x_0]^{s}_{A_0}\ S_1 = S_1`$ By inversion.

$[N_0/x_0]^{n}_{A_0}\ N_2 = N_2`$ and $\Gamma_L, \Gamma_R` \vdash N_2` \Leftarrow S_2`$ By i.h. (1) on $\mathcal{D}_2$.
$[N_2`/x]^{s}_{A_2}\ S_1` = S_1''$ and $[N_0/x_0]^{s}_{A_0}\ S_1' = S_1''$
By Lemma 3.13 (Composition).

Subcase: $[N_0/x_0]^{rr}_{A_0}\ R_1 = R_1`$ and $\Gamma_L, \Gamma_R` \vdash R_1` \Rightarrow \Pi x{::}S_2` \sqsubset A_2`.\ A_1`$
$[N_0/x_0]^{rr}_{A_0}\ R_1\ N_2 = R_1`\ N_2`$ By rule.
$\Gamma_L, \Gamma_R` \vdash R_1`\ N_2` \Rightarrow S_1''$ By rule.

Subcase: $[N_0/x_0]^{rn}_{A_0}\ R_1 = (N_1`, (\Pi x{:}A_2`.\ A_1`)^-)$ and $\Gamma_L, \Gamma_R` \vdash N_1` \Leftarrow \Pi x{::}S_2` \sqsubset A_2`.\ S_1`$
$N_1` = \lambda x.\ N`$ and $\Gamma_L, \Gamma_R`, x{::}S_2` \sqsubset A_2` \vdash N` \Leftarrow S_1`$ By inversion.
$[N_2`/x]^{n}_{A_2}\ N` = N''$ and $\Gamma_L, \Gamma_R` \vdash N'' \Leftarrow S_1''$
By i.h. (1) at $(A_2)^-$, using
$\Gamma_L, \Gamma_R` \vdash N_2` \Leftarrow S_2`$,
$\Gamma_L, \Gamma_R`, x{::}S_2` \sqsubset A_2` \vdash N` \Leftarrow S_1`$,
$[N_2`/x]^{\gamma}_{A_2`}\ \cdot = \cdot$,
and $[N_2`/x]^{s}_{A_2`}\ S_1` = S_1''$.
$[N_0/x_0]^{rn}_{A_0}\ R_1\ N_2 = N''$ By rule, using
$[N_0/x_0]^{rn}_{A_0}\ R_1 = (\lambda x.\ N`, (\Pi x{:}A_2`.\ A_1`)^-)$,
$[N_0/x_0]^{n}_{A_0}\ N_2 = N_2`$,
and $[N_2`/x]^{n}_{A_2}\ N` = N''$.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S_1}$

$[N_0/x_0]^{\mathrm{s}}_{A_0}\, S_1 \wedge S_2 = S\text{`}$ and either
$([N_0/x_0]^{\mathrm{rr}}_{A_0}\, R = R\text{`}$ and $\Gamma_L, \Gamma_R\text{`} \vdash R\text{`} \Rightarrow S\text{`})$, or
$([N_0/x_0]^{\mathrm{rn}}_{A_0}\, R = (N\text{`}, (A\text{`})^-)$ and $\Gamma_L, \Gamma_R\text{`} \vdash N\text{`} \Leftarrow S\text{`})$
By i.h. (2) on $\mathcal{D}_1$.

$S\text{`} = S_1\text{`} \wedge S_2\text{`}$ and $[N_0/x_0]^{\mathrm{s}}_{A_0}\, S_1 = S_1\text{`}$ By inversion.

Subcase: $[N_0/x_0]^{\mathrm{rr}}_{A_0}\, R = R\text{`}$ and $\Gamma_L, \Gamma_R\text{`} \vdash R\text{`} \Rightarrow S_1\text{`} \wedge S_2\text{`}$

$\Gamma_L, \Gamma_R\text{`} \vdash R\text{`} \Rightarrow S_1\text{`}$ By rule.

Subcase: $[N_0/x_0]^{\mathrm{rn}}_{A_0}\, R = (N\text{`}, (A\text{`})^-)$ and $\Gamma_L, \Gamma_R\text{`} \vdash N\text{`} \Leftarrow S_1\text{`} \wedge S_2\text{`}$

$\Gamma_L, \Gamma_R\text{`} \vdash N\text{`} \Leftarrow S_1\text{`}$ By inversion.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S_1 \wedge S_2}{\Gamma_L, x_0{::}S_0 \sqsubset A_0, \Gamma_R \vdash R \Rightarrow S_2}$

Similar. □

B.3 Lemma 3.21 (Commutativity of Substitution and η-expansion)

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. By lexicographic induction on α and the given substitution derivation.

Not all clauses' proofs need be mutually inductive—the two given cases can be proven independently of the ones elided. We give only the proof for the two given cases; the rest are straightforward.

1. (a) Suppose $\mathcal{D} :: [\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha\, N = N'$.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha\, R = R'}{[\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha\, R = R'}$

$R = R'$ By i.h. (1b) on $\alpha, \mathcal{D}_1$.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [\eta_\alpha(x)/x]^{\mathrm{rn}}_\alpha R = (R', a')}{[\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha R = R'}$

$\eta_{a'}(R) = R'$ By i.h. (1c) on $\alpha, \mathcal{D}_1$.
$\eta_{a'}(R) = R$ By definition.
$R = R'$ By transitivity of equality.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha N = N'}{[\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha \lambda y.\, N = \lambda y.\, N'}$

$N = N'$ By i.h. (1a) on $\alpha, \mathcal{D}_1$.
$\lambda y.\, N = \lambda y.\, N'$ By compatibility of equality.

(b) Suppose $\mathcal{D} :: [\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha R = R'$.

Case: $\mathcal{D} = \dfrac{y \neq x}{[\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha y = y}$

$y = y$ By reflexivity of equality.

Case: $\mathcal{D} = \dfrac{}{[\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha c = c}$

$c = c$ By reflexivity of equality.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: [\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha R_1 = R'_1 \qquad \mathcal{D}_2 :: [\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha N_2 = N'_2}{[\eta_\alpha(x)/x]^{\mathrm{rr}}_\alpha R_1\ N_2 = R'_1\ R'_2}$

$R_1 = R'_1$ By i.h. (1b) on $\alpha, \mathcal{D}_1$.
$N_2 = N'_2$ By i.h. (1a) on $\alpha, \mathcal{D}_2$.
$R_1\ N_2 = R'_1\ N'_2$ By compatibility of equality.

(c) Suppose $\mathcal{D} :: [\eta_\alpha(x)/x]^{\mathrm{rn}}_\alpha R = (N, \beta)$

Case: $\mathcal{D} = \dfrac{}{[\eta_\alpha(x)/x]^{\mathrm{rn}}_\alpha x = (\eta_\alpha(x), \alpha)}$

$\eta_\alpha(x) = \eta_\alpha(x)$ By reflexivity of equality.

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 :: [\eta_\alpha(x)/x]^{\mathrm{rn}}_\alpha R_1 = (\lambda y.\, N_1, \alpha_2 \to \alpha_1) \\ \mathcal{D}_2 :: [\eta_\alpha(x)/x]^{\mathrm{n}}_\alpha N_2 = N'_2 \qquad \mathcal{D}_3 :: [N'_2/y]^{\mathrm{n}}_{\alpha_2} N_1 = N'_1\end{array}}{[\eta_\alpha(x)/x]^{\mathrm{rn}}_\alpha R_1\ N_2 = (N'_1, \alpha_1)}$

We need to show: $\eta_{\alpha_1}(R_1\ N_2) = N_1'$.

$\eta_{\alpha_2 \to \alpha_1}(R_1) = \lambda y.\, N_1$	By i.h. (1c) on $\alpha, \mathcal{D}_1$.
$\eta_{\alpha_2 \to \alpha_1}(R_1) = \lambda y.\, \eta_{\alpha_1}(R_1\ \eta_{\alpha_2}(y))$	By definition.
$\eta_{\alpha_1}(R_1\ \eta_{\alpha_2}(y)) = N_1$	By compatibility of equality.

$N_2 = N_2'$	By i.h. (1a) on $\alpha, \mathcal{D}_2$.

$\mathcal{D}_3 :: [N_2/y]^{\mathrm{n}}_{\alpha_2}\ \eta_{\alpha_1}(R_1\ \eta_{\alpha_2}(y)) = N_1'$	By replacing equals for equals.

$y \notin \mathrm{FV}(R_1)$ and $\mathrm{head}(R_1) \neq y$	By α-conversion convention.
$[N_2/y]^{\mathrm{rr}}_{\alpha_2}\ R_1\ \eta_{\alpha_2}(y) = R'$ and $\eta_{\alpha_1}(R') = N_1'$	By i.h. (2a) on $\alpha_1, \mathcal{D}_3$.
$R' = R_1'\ N_2''$ and	
$[N_2/y]^{\mathrm{rr}}_{\alpha_2}\ R_1 = R_1'$ and $\mathcal{D}_4 :: [N_2/y]^{\mathrm{n}}_{\alpha_2}\ \eta_{\alpha_2}(y) = N_2''$	By inversion.

$[N_2/y]^{\mathrm{rr}}_{\alpha_2}\ R_1 = R_1$	By trivial substitution.
$R_1 = R_1'$	By functionality of substitution.

$\mathrm{head}(y) = y$	By definition.
$y{:}\alpha_2 \vdash y : \alpha_2$	By rule.
$[N_2/y]^{\mathrm{rn}}_{\alpha_2}\ y = (N_2'', \alpha_2)$	By i.h. (2b) on $\alpha_2, \mathcal{D}_4$.
$N_2'' = N_2$	By inversion.

$R' = R_1\ N_2$	By equality reasoning.
$\eta_{\alpha_1}(R') = N_1'$	From above.
$\eta_{\alpha_1}(R_1\ N_2) = N_1'$	By replacing equals for equals.

2. Suppose $\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\ \eta_\alpha(R) = N'$.

   (a) Suppose $\mathrm{head}(R) \neq x_0$. We need to show: $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\ R = R'$ and $\eta_\alpha(R') = N'$.

Case: $\alpha = a$.

$\eta_a(R) = R$	By definition.
$\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\ R = N'$	By equality.
$N' = R'$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\ R = R'$.	By inversion, using Lemma 3.1.
$\eta_a(R') = R'$	By definition.

Case: $\alpha = \alpha_2 \to \alpha_1$.

$\eta_{\alpha_2 \to \alpha_1}(R) = \lambda y.\, \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y))$	By definition.
$\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\ \lambda y.\, \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y)) = N'$	By equality.
$N' = \lambda y.\, N''$ and $\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0}\ \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y)) = N''$	By inversion.
$[N_0/x_0]^{\mathrm{rr}}_{\alpha_0}\ R\ \eta_{\alpha_2}(y) = R''$ and $\eta_{\alpha_1}(R'') = N''$	By i.h. (2a) on $\alpha_1, \mathcal{D}_1$.

$R'' = R'\ N$ and $[N_0/x_0]^{\mathrm{rr}}_{\alpha_0} R = R'$ and $[N_0/x_0]^{\mathrm{n}}_{\alpha_0} \eta_{\alpha_2}(y) = N$
By inversion.

$N = \eta_{\alpha_2}(y)$ By trivial substitution and functionality.

$\eta_{\alpha_1}(R'') = \eta_{\alpha_1}(R' \eta_{\alpha_2}(y)) = N''$ By equality.

$\eta_{\alpha_2 \to \alpha_1}(R') = \lambda y.\, \eta_{\alpha_1}(R'\ \eta_{\alpha_2}(y))$ By definition.

$= \lambda y.\, N'' = N'$ By equality.

(b) Suppose the head$(R) = x_0$ and $x_0{:}\alpha_0 \vdash R : \alpha$. We need to show: $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha)$.

Case: $\alpha = a$.

$\eta_a(R) = R$ By definition.

$\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} R = N'$. By equality.

$[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', a')$ By inversion, using Lemma 3.1.

$a' = a$ By Lemma 3.20.

Case: $\alpha = \alpha_2 \to \alpha_1$.

$\eta_{\alpha_2 \to \alpha_1}(R) = \lambda y.\, \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y))$ By definition.

$\mathcal{D} :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} \lambda y.\, \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y)) = N'$ By equality.

$N' = \lambda y.\, N''$ and $\mathcal{D}_1 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} \eta_{\alpha_1}(R\ \eta_{\alpha_2}(y)) = N''$ By inversion.

$x_0{:}\alpha_0 \vdash R\ \eta_{\alpha_2}(y) : \alpha_1$ By rule.

$\mathcal{E} :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R\ \eta_{\alpha_2}(y) = (N'', \alpha_1)$ By i.h. (2b) on $\alpha_1, \mathcal{D}_1$.

$\mathcal{E}_1 :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (\lambda y.\, N''', \alpha_2' \to \alpha_1)$ and
$\mathcal{E}_2 :: [N_0/x_0]^{\mathrm{n}}_{\alpha_0} \eta_{\alpha_2}(y) = N$ and
$\mathcal{E}_3 :: [N/y]^{\mathrm{n}}_{\alpha_2'} N''' = N''$ By inversion.

$\alpha_2' \to \alpha_1 = \alpha_2 \to \alpha_1$ By Lemma 3.20.

$N = \eta_{\alpha_2}(y)$ By trivial substitution and functionality.

$\mathcal{E}_3 :: [\eta_{\alpha_2}(y)/y]^{\mathrm{n}}_{\alpha_2} N''' = N''$ By equality.

$N''' = N''$ By i.h. (1a) on $\alpha_2, \mathcal{E}_3$.

$\mathcal{E}_1 :: [N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (\lambda y.\, N'', \alpha_2 \to \alpha_1)$,
i.e. $[N_0/x_0]^{\mathrm{rn}}_{\alpha_0} R = (N', \alpha)$ By equality.

□

B.4 Theorem 3.22 (Expansion)

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. By induction on S.

Case: $S = \top$

$\Gamma \vdash \eta_A(R) \Leftarrow \top$ By rule.

Case: $S = S_1 \wedge S_2$

$\Gamma \vdash S \sqsubset A_1$ and $\Gamma \vdash S \sqsubset A_2$	By inversion.
$\Gamma \vdash R \Rightarrow S_1$ and $\Gamma \vdash R \Rightarrow S_2$	By rules $\wedge$-$\mathbf{E}_1$ and $\wedge$-$\mathbf{E}_2$.
$\Gamma \vdash \eta_A(R) \Leftarrow S_1$ and $\Gamma \vdash \eta_A(R) \Leftarrow S_2$	By i.h. on S_1 and S_2.
$\Gamma \vdash \eta_A(R) \Leftarrow S_1 \wedge S_2$	By rule $\wedge$-**I**.

Case: $S = Q$

$A = P$	By inversion.
$\eta_A(R) = \eta_P(R) = R$	By definition.
$Q \leq Q$	By rule.
$\Gamma \vdash R \Leftarrow Q$	By rule **switch**.

Case: $S = \Pi x{::}S_1 \sqsubset A_1.\, S_2$

$A = \Pi x{:}A_1.\, A_2$ and $\Gamma \vdash S_1 \sqsubset A_1$ and $\Gamma, x{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$	By inversion.
$\eta_A(R) = \eta_{\Pi x:A_1.\, A_2}(R) = \lambda x.\, \eta_{A_2}(R\ \eta_{A_1}(x))$	By definition.
$\Gamma, y{::}S_1 \sqsubset A_1 \vdash S_1 \sqsubset A_1$	By weakening.
$\Gamma, y{::}S_1 \sqsubset A_1 \vdash y \Rightarrow S_1$	By rule.
$\Gamma, y{::}S_1 \sqsubset A_1 \vdash \eta_{A_1}(y) \Leftarrow S_1$	By i.h. on S_1.
$\Gamma, y{::}S_1 \sqsubset A_1, x{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$	By weakening.
$[\eta_{A_1}(y)/x]^{\mathrm{s}}_{A_1}\, S_2 = S_2'$	By Theorem 3.19 (Substitution).
$S_2' = [y/x]\, S_2$	By Lemma 3.21 (Commutativity).
$\Gamma, y{::}S_1 \sqsubset A_1 \vdash R \Rightarrow \Pi x{::}S_1 \sqsubset A_1.\, S_2$	By weakening.
$\Gamma, y{::}S_1 \sqsubset A_1 \vdash R\ \eta_{A_1}(y) \Rightarrow [y/x]\, S_2$	By rule Π-**E**.
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash R\ \eta_{A_1}(x) \Rightarrow S_2$	By α-conversion.
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash \eta_{A_2}(R\ \eta_{A_1}(x)) \Leftarrow S_2$	By i.h. on S_2.
$\Gamma \vdash \lambda x.\, \eta_{A_2}(R\ \eta_{A_1}(x)) \Leftarrow \Pi x{::}S_1 \sqsubset A_1.\, S_2$	By rule Π-**I**.

□

B.5 Theorem 4.6 (Generalized Algorithmic ⇒ Declarative)

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. By induction on $\mathcal{D}$. To reduce clutter, we omit the refined type A_1 from bound variables, since it does not affect declarative subsorting in any significant way.

1. Suppose $\mathcal{D} :: \Delta \leqq T$.

   Case: $\mathcal{D} = \dfrac{}{\Delta \leqq \top}$

   $\bigwedge(\Delta) \leq \top$ By rule $\top$**-R**.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \leqq S_1 \qquad \mathcal{D}_2 :: \Delta \leqq S_2}{\Delta \leqq S_1 \wedge S_2}$

   $\bigwedge(\Delta) \leq S_1$ By i.h. (1) on $\mathcal{D}_1$.
   $\bigwedge(\Delta) \leq S_2$ By i.h. (1) on $\mathcal{D}_2$.
   $\bigwedge(\Delta) \leq S_1 \wedge S_2$ By rule $\wedge$**-R**.

   Case: $\mathcal{D} = \dfrac{Q' \in \Delta \qquad Q' \leq Q}{\Delta \leqq Q}$

   $\bigwedge(\Delta) \leq Q$ By Lemma 4.5.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \mathbin{@} x{::}\mathrm{split}(S_1) = \Delta_2 \qquad \mathcal{D}_2 :: \Delta_2 \leqq S_2}{\Delta \leqq \Pi x{::}S_1.\, S_2}$

   $\bigwedge(\Delta) \leq \Pi x{::} \bigwedge(\mathrm{split}(S_1)).\ \bigwedge(\Delta_2)$ By i.h. (2) on $\mathcal{D}_1$.

   $S_1 \leq \bigwedge(\mathrm{split}(S_1))$ By Lemma 4.4.
   $\bigwedge(\Delta_2) \leq S_2$ By i.h. (1) on $\mathcal{D}_2$.
   $\Pi x{::} \bigwedge(\mathrm{split}(S_1)).\ \bigwedge(\Delta)_2 \leq \Pi x{::}S_1.\, S_2$ By rule **S-**Π.

   $\bigwedge(\Delta) \leq \Pi x{::}S_1.\, S_2$ By rule **trans**.

2. Suppose $\mathcal{D} :: \Delta \mathbin{@} x{::}\mathrm{split}(S_1) = \Delta_2$.

   Case: $\mathcal{D} = \dfrac{}{\cdot \mathbin{@} x{::}\Delta_1 = \cdot}$

   $\bigwedge(\cdot) = \top$ By definition.
   We need to show $\top \leq \Pi x{::} \bigwedge(\Delta_1).\ \top$.
   $\top \leq \Pi x{::} \bigwedge(\Delta_1).\ \top$ By rule $\top/\Pi$**-dist**.

   Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \mathbin{@} x{::}\Delta_1 = \Delta_2 \qquad \mathcal{D}_2 :: \Delta_1 \leqq S_1 \qquad \mathcal{D}_3 :: [\eta_A(x)/y]^{\mathrm{s}}_A\, S_2 = S_2'}{(\Delta, \Pi y{::}S_1 \sqsubset A.\, S_2) \mathbin{@} x{::}\Delta_1 = \Delta_2, \mathrm{split}(S_2')}$

   $\bigwedge(\Delta, \Pi y{::}S_1 \sqsubset A.\, S_2) = \bigwedge(\Delta) \wedge \Pi y{::}S_1 \sqsubset A.\, S_2$ By definition.
   Want to show: $\bigwedge(\Delta) \wedge \Pi y{::}S_1 \sqsubset A.\, S_2 \leq \Pi x{::} \bigwedge(\Delta_1).\ \bigwedge(\Delta_2, \mathrm{split}(S_2'))$.
   $S_2' = [x/y]\, S_2$ By Lemma 3.21 (Commutativity).
   So (α-varied): $\bigwedge(\Delta) \wedge \Pi x{::}S_1 \sqsubset A.\, S_2' \leq \Pi x{::} \bigwedge(\Delta_1).\ \bigwedge(\Delta_2, \mathrm{split}(S_2'))$.

   Note: in the following, we omit some uses of reflexivity (rule **refl**).

$\bigwedge(\Delta) \leq \Pi x{::}\bigwedge(\Delta_1).\ \bigwedge(\Delta_2)$ By i.h. (2) on $\mathcal{D}_1$.
$\bigwedge(\Delta_1) \leq S_1$ By i.h. (1) on $\mathcal{D}_2$.
$\bigwedge(\Delta) \wedge \Pi x{::}S_1.\ S_2' \leq \Pi x{::}\bigwedge(\Delta_1).\ \left(\bigwedge(\Delta_2) \wedge S_2'\right)$ By rule $\wedge/\Pi$-**dist'**.

$S_2' \leq \bigwedge(\text{split}(S_2'))$ By Lemma 4.4.
$\bigwedge(\Delta_2) \wedge S_2' \leq \bigwedge(\Delta_2) \wedge \bigwedge(\text{split}(S_2'))$ By rule **S-**$\wedge$.
$\bigwedge(\Delta_2) \wedge \bigwedge(\text{split}(S_2')) \leq \bigwedge(\Delta_2, \text{split}(S_2'))$ By Lemma 4.3.
$\bigwedge(\Delta_2) \wedge S_2' \leq \bigwedge(\Delta_2, \text{split}(S_2'))$ By rule **trans**.
$\Pi x{::}\bigwedge(\Delta_1).\ \left(\bigwedge(\Delta_2) \wedge S_2'\right) \leq \Pi x{::}\bigwedge(\Delta_1).\ \bigwedge(\Delta_2, \text{split}(S_2'))$ By rule **S-**Π.

$\bigwedge(\Delta) \wedge \Pi x{::}S_1.\ S_2' \leq \Pi x{::}\bigwedge(\Delta_1).\ \bigwedge(\Delta_2, \text{split}(S_2'))$ By rule **trans**.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \mathbin{@} x{::}\Delta_1 = \Delta_2 \qquad \Delta_1 \not\leqq S_1}{\Delta, \Pi y{::}S_1 \sqsubset A.\ S_2 \mathbin{@} x :: \Delta_1 = \Delta_2}$

$\bigwedge(\Delta, \Pi y{::}S_1 \sqsubset A.\ S_2) = \bigwedge(\Delta) \wedge \Pi y{::}S_1 \sqsubset A.\ S_2$ By definition.
$\bigwedge(\Delta) \leq \Pi x{::}\bigwedge(\Delta_1).\ \bigwedge(\Delta_2)$ By i.h. (2) on $\mathcal{D}_1$.
$\bigwedge(\Delta) \wedge \Pi y{::}S_1 \sqsubset A.\ S_2 \leq \Pi x{::}\bigwedge(\Delta_1).\ \bigwedge(\Delta_2)$ By rule $\wedge$-$\mathbf{L}_1$.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \mathbin{@} x{::}\Delta_1 = \Delta_2 \qquad \nexists S_2'.\ [\eta_A(x)/y]^{\text{s}}_A\ S_2 = S_2'}{\Delta, \Pi y{::}S_1 \sqsubset A.\ S_2 \mathbin{@} x :: \Delta_1 = \Delta_2}$

Similar.

Case: $\mathcal{D} = \dfrac{\mathcal{D}_1 :: \Delta \mathbin{@} x{::}\Delta_1 = \Delta_2}{\Delta, Q \mathbin{@} x :: \Delta_1 = \Delta_2}$

Similar. □

B.6 Lemma 4.10

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. By induction on $\mathcal{E}$.

Case: $\mathcal{E} = \dfrac{}{\Gamma \vdash \cdot \mathbin{@} N = \cdot}$

$\Gamma \vdash \cdot \sqsubset A_2'$ By rule.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2 \qquad \mathcal{E}_2 :: \Gamma \vdash N \Leftarrow S_1 \qquad \mathcal{E}_3 :: [N/x]^{\text{s}}_{A_1}\ S_2 = S_2'}{\Gamma \vdash (\Delta, \Pi x{::}S_1 \sqsubset A_1.\ S_2) \mathbin{@} N = \Delta_2, \text{split}(S_2')}$

$\mathcal{D}_1 :: \Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ and
$\mathcal{D}_2 :: \Gamma \vdash \Pi x{::}S_1 \sqsubset A_1.\, S_2 \sqsubset \Pi x{:}A_1.\, A_2$ By inversion on $\mathcal{D}$.
$\Gamma \vdash \Delta_2 \sqsubset A_2'$ By i.h. on $\mathcal{E}_1$.

$\Gamma \vdash S_1 \sqsubset A_1$ and $\Gamma, x{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$ By inversion on $\mathcal{D}_2$.
$\Gamma \vdash N \Leftarrow S_1$ By Theorem 3.9 (Soundness of Alg. Typing).
$\Gamma \vdash S_2' \sqsubset A_2'$ By Theorem 3.19 (Substitution).
$\Gamma \vdash \mathrm{split}(S_2') \sqsubset A_2'$ By Lemma 4.9.

$\Gamma \vdash (\Delta_2, \mathrm{split}(S_2')) \sqsubset A_2'$ By Lemma 4.8.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2 \qquad \Gamma \nvdash N \Leftarrow S_1}{\Gamma \vdash (\Delta, \Pi x{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} N = \Delta_2}$

$\Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ By inversion on $\mathcal{D}$.
$\Gamma \vdash \Delta_2 \sqsubset A_2'$ By i.h. on $\mathcal{E}_1$.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2 \qquad \nexists S_2'.\, [N/y]^{\mathrm{s}}_{A_1} S_2 = S_2'}{\Gamma \vdash (\Delta, \Pi x{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} N = \Delta_2}$

Similar.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2}{\Gamma \vdash (\Delta, Q) \mathbin{@} N = \Delta_2}$

Impossible:
$\Gamma \vdash Q \sqsubset \Pi x{:}A_1.\, A_2$ By inversion on $\mathcal{D}$.
But there is no rule that can conclude this. □

B.7 Theorem 4.14 (Generalized Intrinsic ⇒ Algorithmic)

Theorem 4.14 (Generalized Intrinsic ⇒ 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. By induction on A, S, and $\mathcal{E}$. We omit the refined type A_1 from the Δ_1 argument of the application judgement when it is clear from context.

1. Suppose $\mathcal{D} :: \Gamma \vdash R \Rrightarrow \Delta$, $\mathcal{E} :: \Gamma \vdash \eta_A(R) \Leftarrow S$, $\mathcal{F} :: \Gamma \vdash \Delta \sqsubset A$, and $\mathcal{G} :: \Gamma \vdash S \sqsubset A$.

Case: $S = Q$

$A = P$ By inversion on $\mathcal{G}$.
$\eta_P(R) = R$ By definition.

$$\mathcal{E} = \frac{\Gamma \vdash R \Rrightarrow \Delta \qquad Q' \in \Delta \qquad Q' \leq Q}{\Gamma \vdash R \Lleftarrow Q}$$ By inversion.

$\Delta \leqq Q$ By rule.

Case: $S = \top$

$\Delta \leqq \top$ By rule.

Case: $S = S_1 \wedge S_2$

$$\mathcal{E} = \frac{\mathcal{E}_1 :: \Gamma \vdash \eta_A(R) \Lleftarrow S_1 \qquad \mathcal{E}_2 :: \Gamma \vdash \eta_A(R) \Lleftarrow S_2}{\Gamma \vdash \eta_A(R) \Lleftarrow S_1 \wedge S_2}$$ By inversion.

$\Delta \leqq S_1$ By i.h. (1) on S_1 and $\mathcal{E}_1$
$\Delta \leqq S_2$ By i.h. (1) on S_2 and $\mathcal{E}_2$
$\Delta \leqq S_1 \wedge S_2$ By rule.

Case: $S = \Pi x{::}S_1 \sqsubset A_1.\, S_2$

$A = \Pi x{:}A_1.\, A_2$ and
$\Gamma \vdash S_1 \sqsubset A_1$ and $\Gamma, x{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$ By inversion on $\mathcal{G}$.
$\eta_{\Pi x:A_1.\,A_2}(R) = \lambda x.\, \eta_{A_2}(R\ \eta_{A_1}(x))$ By definition.

$$\mathcal{E} = \frac{\mathcal{E}_1 :: \Gamma, x{::}S_1 \sqsubset A_1 \vdash \eta_{A_2}(R\ \eta_{A_1}(x)) \Lleftarrow S_2}{\Gamma \vdash \lambda x.\, \eta_{A_2}(R\ \eta_{A_1}(x)) \Lleftarrow \Pi x{::}S_1 \sqsubset A_1.\, S_2}$$ By inversion.

$\Gamma, x{::}S_1 \sqsubset A_1 \vdash R \Rrightarrow \Delta$ By weakening.
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2$ By Theorem 3.7, clause (3).
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash R\ \eta_{A_1}(x) \Rrightarrow \Delta_2$ By rule.
$[\eta_{A_1}(x)/A_1]^{\mathrm{a}}_x A_2 = A_2$ By validity of $\Pi x{:}A_1.\, A_2$ and Lemma 3.21.
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash \Delta_2 \sqsubset A_2$ By Lemma 4.10.
$\Delta_2 \leqq S_2$ By i.h. (1) on A_2, S_2, and $\mathcal{E}_1$.

$\Gamma, x{::}S_1 \sqsubset A_1 \vdash x \Rrightarrow \mathrm{split}(S_1)$ By rule.
$\Gamma, x{::}S_1 \sqsubset A_1 \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2$ From above.
$\Gamma \vdash \mathrm{split}(S_1) \sqsubset A_1$ By Lemma 4.9.
$\Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ By assumption.
$\Delta \mathbin{@} x{::}\mathrm{split}(S_1) = \Delta_2$ By i.h. (2) on A_1.

$\Delta \leqq \Pi x{::}S_1 \sqsubset A_1.\, S_2$ By rule.

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

Case: $\mathcal{E} = \dfrac{}{\Gamma \vdash \cdot \mathbin{@} \eta_{A_1}(x) = \cdot}$

$\cdot \mathbin{@} x{::}\Delta_1 = \cdot$ By rule.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2 \qquad \mathcal{E}_2 :: \Gamma \vdash \eta_{A_1}(x) \Leftarrow S_1 \qquad \mathcal{E}_3 :: [\eta_{A_1}(x)/y]^{\mathrm{s}}_{A_1} S_2 = S_2'}{\Gamma \vdash (\Delta, \Pi y{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} \eta_{A_1}(x) = \Delta_2, \mathrm{split}(S_2')}$

$\Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ and
$\Gamma \vdash S_1 \sqsubset A_1$ and $\Gamma, y{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$ By inversion on $\mathcal{G}$.
$\Delta \mathbin{@} x{::}\Delta_1 = \Delta_2$ By i.h. (2) on $\mathcal{E}_1$.
$\Delta_1 \leqq S_1$ By i.h. (1) on $\mathcal{E}_2$.
$[\eta_{A_1}(x)/y]^{\mathrm{s}}_{A_1} S_2 = S_2'$ By subderivation $\mathcal{E}_3$.
$(\Delta, \Pi y{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} x{::}\Delta_1 = (\Delta_2, \mathrm{split}(S_2'))$ By rule.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2 \qquad \Gamma \not\vdash \eta_{A_1}(x) \Leftarrow S_1}{\Gamma \vdash (\Delta, \Pi x{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} \eta_{A_1}(x) = \Delta_2}$

$\Gamma \vdash \Delta \sqsubset \Pi x{:}A_1.\, A_2$ and
$\Gamma \vdash S_1 \sqsubset A_1$ and $\Gamma, y{::}S_1 \sqsubset A_1 \vdash S_2 \sqsubset A_2$ By inversion on $\mathcal{G}$.
$\Delta \mathbin{@} x{::}\Delta_1 = \Delta_2$ By i.h. (2) on $\mathcal{E}_1$.
$\Delta_1 \not\leqq S_1$ By Theorem 4.13 (Alg. Subsumption), contrapositive.
$(\Delta, \Pi y{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} x{::}\Delta_1 = \Delta_2$ By rule.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} \eta_{A_1}(x) = \Delta_2 \qquad \nexists S_2'.\, [\eta_{A_1}(x)/y]^{\mathrm{s}}_{A_1} S_2 = S_2'}{\Gamma \vdash (\Delta, \Pi y{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} \eta_{A_1}(x) = \Delta_2}$

$\Delta \mathbin{@} x{::}\Delta_1 = \Delta_2$ By i.h. (2) on $\mathcal{E}_1$.
$\nexists S_2'.\, [\eta_A(x)/y]^{\mathrm{s}}_{A_1} S_2 = S_2'$ By side condition.
$(\Delta, \Pi y{::}S_1 \sqsubset A_1.\, S_2) \mathbin{@} x{::}\Delta_1 = \Delta_2$ By rule.

Case: $\mathcal{E} = \dfrac{\mathcal{E}_1 :: \Gamma \vdash \Delta \mathbin{@} N = \Delta_2}{\Gamma \vdash (\Delta, Q) \mathbin{@} N = \Delta_2}$

Impossible:
$\Gamma \vdash Q \sqsubset \Pi x{:}A_1.\, A_2$ By inversion on $\mathcal{G}$.
But there is no rule that can conclude this. □