Linear Higher-Order Pre-Unification

Iliano Cervesato and Frank Pfenning[1]

July 20, 1997

CMU-CS-97-160

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We develop a pre-unification algorithm in the style of Huet for the linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ which includes intuitionistic functions ($\rightarrow$), linear functions ($\multimap$), additive pairing ($\&$), and additive unit ($\top$). This procedure conveniently operates on an efficient representation of $\lambda^{\rightarrow\multimap\&\top}$, the spine calculus $S^{\rightarrow\multimap\&\top}$ for which we define the concept of weak head-normal form. We prove the soundness and completeness of our algorithm with respect to the proper notion of definitional equality for $S^{\rightarrow\multimap\&\top}$, and illustrate the distinctive aspects of linear higher-order unification by means of examples. We also show that, surprisingly, a similar pre-unification algorithm does not exist for certain sublanguages. Applications lie in proof search, logic programming, and logical frameworks based on linear type theories.

[1] The authors can be reached at *iliano@cs.cmu.edu* and *fp@cs.cmu.edu*.

This work was sponsored NSF Grant CCR-9303383. The second author was supported by the *Alexander-von-Humboldt-Stiftung* when working on this paper, during a visit to the Department of Mathematics of the Technical University Darmstadt.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. Government.

Keywords: Linear Lambda Calculus, Linear Higher-Order Unification.

Contents

1 Introduction **1**

2 A Linear Simply-Typed λ-Calculus **2**

2.1 Basic Formulation . . . 2
2.2 The Spine Calculus . . . 4
2.3 Head-Normal Forms in the Spine Calculus . . . 8
2.4 Equality in the Spine Calculus . . . 12
2.5 Eta-Expansion in the Spine Calculus . . . 15

3 Linear Higher-Order Unification **24**

3.1 The Unification Problem . . . 24
3.2 Examples . . . 25
3.3 A Pre-Unification Algorithm . . . 26
3.4 Soundness and Completeness . . . 34
3.4.1 Soundness . . . 34
3.4.2 Preliminary Definitions for the Completeness Theorem . . . 35
3.4.3 Non-Deterministic Completeness . . . 39
3.5 Non-Determinism . . . 50

4 Discussion **51**

4.1 Sublanguages . . . 51
4.2 Towards a Practical Implementation . . . 52
4.3 Related Work . . . 53

5 Conclusion and Future Work **54**

References **54**

Notation **57**

List of Statements **60**

Index **61**

List of Figures

1 Typing in $\lambda^{\rightarrow\multimap\&\top}$ . . . 3
2 Typing for η-Long $S^{\rightarrow\multimap\&\top}$ Terms . . . 4
3 Reduction Semantics for $S^{\rightarrow\multimap\&\top}$ . . . 6
4 (Weak) Head-Reduction for $S^{\rightarrow\multimap\&\top}$ . . . 9
5 Equality in $S^{\rightarrow\multimap\&\top}$ . . . 13
6 Typing for Pseudo-Spines and Pseudo-Roots . . . 15
7 Variable η-Expansion in $S^{\rightarrow\multimap\&\top}$ . . . 18
8 Pre-Unification in $S^{\rightarrow\multimap\&\top}$, Equation Manipulation . . . 27
9 Pre-Unification in $S^{\rightarrow\multimap\&\top}$, Generation of Substitutions . . . 28
10 Pre-Unification in $S^{\rightarrow\multimap\&\top}$, Raising Variables . . . 29
11 Sublanguages of $\lambda^{\rightarrow\multimap\&\top}$ . . . 51

1 Introduction

Linear logic [Gir87] enriches more traditional logical formalisms with a notion of consumable resource, which provides direct means for expressing and reasoning about mutable state. Attempts at mechanizing this additional expressive power led to the design of several logic programming languages based on various fragments of linear logic. The only new aspect in the operational semantics of most proposals, such as *Lolli* [HM94], *Lygon* [HP94] and *Forum* [Mil96], concerns the management of linear context formulas [CHP96]. In particular, the instantiation of logical variables relies on the traditional unification algorithms, in their first- or higher-order variants, depending on the language. More recent proposals, such as the language of the linear logical framework *LLF* [Cer96, CP96] and the system *RLF* [IP96], introduce linearity not only at the level of formulas, but also within terms. Consequently, implementations of these languages must solve higher-order equations on linear terms in order to instantiate existential variables. In this paper we present a complete algorithm for pre-unification in a linear λ-calculus which conservatively extends the ordinary simply-typed λ-calculus and could be used directly for the above languages.

An example will shed some light on the novel issues brought in by linearity. A rewrite rule $r : t_1 \Longrightarrow t_2$ is applicable to a term t if there is an instance of t_1 in t; then, applying r has the effect of replacing it with t_2 (assume t_1 and t_2 ground, for simplicity). This is often formalized by writing $t = \tilde{t}[t_1]$, where the *rewriting context* $\tilde{t}$ is a term containing a unique occurrence of a *hole* ($[\_]$) so that replacing the hole with t_1 yields t. We can then express r as the parametric transition $T[t_1] \Longrightarrow T[t_2]$, where T is a variable standing for a rewriting context. The applicability of r to a term t reduces to the problem of whether t and the higher-order expression $(T\,t_1)$ are unifiable, where T is viewed as a functional variable. Traditional higher-order unification does not take into consideration the *linearity constraint* that exactly one occurrence of t_1 must be abstracted away from t. Indeed, matching $(T\,t_1)$ with $(c\,t_1\,t_1)$ has four solutions:

$$\boxed{\begin{array}{l} T \longleftarrow \lambda x.\, c\, t_1\, t_1 \\ T \longleftarrow \lambda x.\, c\, x\, x \end{array}} \qquad\qquad \begin{array}{l} T \longleftarrow \lambda x.\, c\, x\, t_1 \\ T \longleftarrow \lambda x.\, c\, t_1\, x \end{array}$$

But the first match in the box does not have any hole (the variable x) in it while the second contains two. Linear unification, on the other hand, returns correctly only the two unboxed solutions. This means also that a natural encoding of a rewrite system based on rewriting contexts in the logical framework *LF* would implement a post-processing phase that filters out non-linear solutions, while this step would be unnecessary in *LLF*. The problem representation would therefore be more direct and compact in this language.

The introduction of linear term languages in *LLF* and *RLF* has been motivated by a number of applications. Linear terms provide a statically checkable notation for natural deductions [IP96] or sequent derivations [CP96] in substructural logics. In the realm of programming languages, linear terms naturally model *computations* in imperative languages [CP96] or sequences of moves in games [Cer96]. When we want to specify, manipulate, or reason about such objects (which is common in logic and the theory of programming languages), then internal linearity constraints are critical in practice (see, for example, the first formalizations of cut-elimination in linear logic and type preservation for *Mini-ML* with references [CP96]).

Differently from the first-order case, higher-order unification in Church's simply typed λ-calculus $\lambda^{\rightarrow}$ is undecidable and does not admit most general unifiers [Gol81]. Nevertheless sound and complete (although possibly non-terminating) procedures have been proposed in order to enumerate solutions [JP76]. In particular, Huet's pre-unification algorithm [Hue75] computes unifiers in a non-redundant manner as constraints and has therefore been adopted in the implementation of higher-order logic programming languages [NM88]. Fragments of $\lambda^{\rightarrow}$ of practical relevance for which unification is decidable and yields most general unifiers have also been discovered. An example are Miller's higher-order patterns [Mil91], that have been implemented in the higher-order constraint logic programming language *Elf* [Pfe91a]. Unification in the context of linear λ-calculi has received limited attention in the literature and, to our knowledge, only a restricted fragment of a multiplicative language has been treated [Lev96]. Unification in $\lambda^{\rightarrow}$ with linear restrictions on existential variables has been studied in [Pre95].

In this extended abstract, we investigate the unification problem in the linear simply-typed λ-calculus $\lambda^{\rightarrow\multimap\&\top}$. We give a pre-unification procedure in the style of Huet and discuss the new sources of non-determinism due to linearity. Moreover, we show that no such algorithm can be devised for linear sublanguages deprived of $\top$ and of the corresponding constructor. $\lambda^{\rightarrow\multimap\&\top}$ corresponds, via a natural extension of the Curry-Howard isomorphism, to the fragment of intuitionistic linear logic freely generated from the connectives $\rightarrow$, $\multimap$, $\&$ and $\top$, which constitutes the propositional core of *Lolli* [HM94] and *LLF* [CP96]. $\lambda^{\rightarrow\multimap\&\top}$ is also the simply-typed variant of the term language of *LLF* and shares similarities with the calculus proposed in [Bar96]. Its theoretical relevance derives from the fact that it is the largest linear λ-calculus that admits unique long $\beta\eta$-normal forms.

The principal contributions of this work are: (1) a first solution to the problem of linear higher-order unification, currently a major obstacle to the implementation of logical frameworks and logic programming languages relying on a linear higher-order term language; (2) the elegant and precise presentation of an extension of Huet's pre-unification procedure as a system of inference rules.

Our presentation is organized as follows. In Section 2, we define $\lambda^{\rightarrow\multimap\&\top}$ and introduce the spine calculus $S^{\rightarrow\multimap\&\top}$ as an equivalent formulation better suited for our purposes. The pre-unification algorithm is the subject of Section 3, where we define the problem, present our solution and prove its soundness and completeness with respect to the proper notion of equality for $S^{\rightarrow\multimap\&\top}$. We study the unification problem in sublanguages of $\lambda^{\rightarrow\multimap\&\top}$ and hint at the possibility of a practical implementation in Section 4. In order to facilitate our description in the available space, we must assume the reader familiar with traditional higher-order unification [Hue75] and linear logic [Gir87].

2 A Linear Simply-Typed λ-Calculus

This section defines the simply-typed linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ (Section 2.1) and presents an equivalent formulation, $S^{\rightarrow\multimap\&\top}$ (Section 2.2), which is more convenient for describing and implementing unification. Moreover, we define the notion of (weak) head-normal form for $S^{\rightarrow\multimap\&\top}$ (Section 2.3), and discuss equality in this calculus (Section 2.4). We conclude with a technical note about η-expansion in $S^{\rightarrow\multimap\&\top}$ (Section 2.5).

2.1 Basic Formulation

The linear simply-typed λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ extends Church's $\lambda^{\rightarrow}$ with the three type constructors $\multimap$ (*multiplicative arrow*), $\&$ (*additive product*) and $\top$ (*additive unit*), derived from the identically denoted connectives of linear logic. The language of terms is augmented accordingly with constructors and destructors, devised from the natural deduction style inference rules for these connectives. Although not strictly necessary at this level of the description, the inclusion of intuitionistic constants will be convenient in the development of the discussion. We present the resulting grammar in a tabular format that relates each type constructor (left) to the corresponding term operators (center), with constructors preceding

$$
\frac{}{\Gamma; \cdot \vdash_{\Sigma, c:A} c : A}\,\lambda\_\mathrm{con}
\qquad
\frac{}{\Gamma; x:A \vdash_{\Sigma} x : A}\,\lambda\_\mathrm{lvar}
\qquad
\frac{}{\Gamma, x:A; \cdot \vdash_{\Sigma} x : A}\,\lambda\_\mathrm{ivar}
$$

$$
\frac{}{\Gamma; \Delta \vdash_{\Sigma} \langle\rangle : \top}\,\lambda\_\mathrm{unit}
\qquad
\text{(No elimination rule for } \top\text{)}
$$

$$
\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \quad \Gamma; \Delta \vdash_{\Sigma} N : B}{\Gamma; \Delta \vdash_{\Sigma} \langle M, N\rangle : A \mathbin{\&} B}\,\lambda\_\mathrm{pair}
\qquad
\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \mathbin{\&} B}{\Gamma; \Delta \vdash_{\Sigma} \textsc{fst}\, M : A}\,\lambda\_\mathrm{fst}
\qquad
\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \mathbin{\&} B}{\Gamma; \Delta \vdash_{\Sigma} \textsc{snd}\, M : B}\,\lambda\_\mathrm{snd}
$$

$$
\frac{\Gamma; \Delta, x:A \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \hat{\lambda} x:A.\, M : A \multimap B}\,\lambda\_\mathrm{llam}
\qquad
\frac{\Gamma; \Delta' \vdash_{\Sigma} M : A \multimap B \quad \Gamma; \Delta'' \vdash_{\Sigma} N : A}{\Gamma; \Delta', \Delta'' \vdash_{\Sigma} M\,\hat{}\,N : B}\,\lambda\_\mathrm{lapp}
$$

$$
\frac{\Gamma, x:A; \Delta \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \lambda x:A.\, M : A \to B}\,\lambda\_\mathrm{ilam}
\qquad
\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \to B \quad \Gamma; \cdot \vdash_{\Sigma} N : A}{\Gamma; \Delta \vdash_{\Sigma} M\, N : B}\,\lambda\_\mathrm{iapp}
$$

Figure 1: Typing in $\lambda^{\to\multimap\&\top}$

destructors. Clearly constants and variables can have any type.

$$
\begin{array}{llllll}
\textit{Types:}\ A ::= & a & \textit{Terms:}\ M ::= & c \mid x & & \\
 & \mid A_1 \to A_2 & & \mid \lambda x:A.\, M & \mid M_1\, M_2 & (\textit{intuitionistic functions}) \\
 & \mid A_1 \multimap A_2 & & \mid \hat{\lambda} x:A.\, M & \mid M_1\,\hat{}\,M_2 & (\textit{linear functions}) \\
 & \mid A_1 \mathbin{\&} A_2 & & \mid \langle M_1, M_2\rangle & \mid \textsc{fst}\, M \mid \textsc{snd}\, M & (\textit{additive pairs}) \\
 & \mid \top & & \mid \langle\rangle & & (\textit{additive unit})
\end{array}
$$

As usual, we rely on signatures and contexts to assign types to constants and free variables, respectively.

$$
\textit{Signatures:}\ \Sigma ::= \cdot \mid \Sigma, c : A \qquad\qquad \textit{Contexts:}\ \Gamma ::= \cdot \mid \Gamma, x : A
$$

Here x, c and a range over variables, constants and base types, respectively. In addition to the names displayed above, we will often use N, B and Δ for objects, types and contexts, respectively.

The notions of free and bound variables are adapted from $\lambda^{\to}$. As usual, we identify terms that differ only by the name of their bound variables and write $[M/x]N$ for the capture-avoiding substitution of M for x in the term N. We require variables and constants to be declared at most once in a context and in a signature, respectively. Since the order in which these declarations occur will be irrelevant in our presentation, we will treat contexts and signatures as multisets (with every element occurring exactly once). We promote "," to denote their union and omit writing "$\cdot$" when unnecessary; when using this notation in Δ, Δ' for example, we shall always assume that the participating multisets Δ and Δ' are disjoint.

The typing judgment for $\lambda^{\to\multimap\&\top}$ has the form

$$
\Gamma; \Delta \vdash_{\Sigma} M : A
$$

where Γ and Δ are called the *intuitionistic* and the *linear* context, respectively. The inference rules for this judgment are displayed in Figure 1. Deleting the terms that appear in them results in the usual rules for the $(\to\multimap\&\top)$ fragment of intuitionistic linear logic, $ILL^{\to\multimap\&\top}$ [HM94], in a natural deduction formulation. $\lambda^{\to\multimap\&\top}$ and $ILL^{\to\multimap\&\top}$ are related by a form of the Curry-Howard isomorphism.

The reduction semantics of $\lambda^{\to\multimap\&\top}$ is given by the transitive and reflexive closure of the congruence relation built on the following β-reduction rules:

$$
\begin{array}{lcl}
\textsc{fst}\,\langle M, N\rangle & \longrightarrow & M \\
\textsc{snd}\,\langle M, N\rangle & \longrightarrow & N
\end{array}
\qquad\qquad
\begin{array}{lcl}
(\hat{\lambda} x:A.\, M)\,\hat{}\,N & \longrightarrow & [N/x]M \\
(\lambda x:A.\, M)\, N & \longrightarrow & [N/x]M
\end{array}
$$

Similarly to $\lambda^{\to}$, $\lambda^{\to\multimap\&\top}$ enjoys a number of highly desirable properties [Cer96]. In particular, since the usual presentation of the elimination rules for the remaining operators (for example for $\otimes$)

Terms

$$\frac{\Gamma;\Delta' \vdash_\Sigma U : A \quad \Gamma;\Delta'' \vdash_\Sigma S : A > a}{\Gamma;\Delta',\Delta'' \vdash_\Sigma U \cdot S : a}\,\textsf{lS\_redex}$$

$$\frac{\Gamma;\Delta \vdash_{\Sigma,c:A} S : A > a}{\Gamma;\Delta \vdash_{\Sigma,c:A} c \cdot S : a}\,\textsf{lS\_con} \qquad \frac{\Gamma;\Delta \vdash_\Sigma S : A > a}{\Gamma;\Delta, x\!:\!A \vdash_\Sigma x \cdot S : a}\,\textsf{lS\_lvar} \qquad \frac{\Gamma, x\!:\!A;\Delta \vdash_\Sigma S : A > a}{\Gamma, x\!:\!A;\Delta \vdash_\Sigma x \cdot S : a}\,\textsf{lS\_ivar}$$

$$\frac{}{\Gamma;\Delta \vdash_\Sigma \langle\rangle : \top}\,\textsf{lS\_unit} \qquad \frac{\Gamma;\Delta \vdash_\Sigma U_1 : A_1 \quad \Gamma;\Delta \vdash_\Sigma U_2 : A_2}{\Gamma;\Delta \vdash_\Sigma \langle U_1, U_2\rangle : A_1 \mathbin{\&} A_2}\,\textsf{lS\_pair}$$

$$\frac{\Gamma;\Delta, x\!:\!A \vdash_\Sigma U : B}{\Gamma;\Delta \vdash_\Sigma \hat{\lambda} x\!:\!A.\, U : A \multimap B}\,\textsf{lS\_llam} \qquad \frac{\Gamma, x\!:\!A;\Delta \vdash_\Sigma U : B}{\Gamma;\Delta \vdash_\Sigma \lambda x\!:\!A.\, U : A \to B}\,\textsf{lS\_ilam}$$

Spines

$$\frac{}{\Gamma;\cdot \vdash_\Sigma \textsc{nil} : a > a}\,\textsf{lS\_nil}$$

(No spine rule for $\top$)

$$\frac{\Gamma;\Delta \vdash_\Sigma S : A_1 > a}{\Gamma;\Delta \vdash_\Sigma \pi_1\, S : A_1 \mathbin{\&} A_2 > a}\,\textsf{lS\_fst} \qquad \frac{\Gamma;\Delta \vdash_\Sigma S : A_2 > a}{\Gamma;\Delta \vdash_\Sigma \pi_2\, S : A_1 \mathbin{\&} A_2 > a}\,\textsf{lS\_snd}$$

$$\frac{\Gamma;\Delta' \vdash_\Sigma U : A \quad \Gamma;\Delta'' \vdash_\Sigma S : B > a}{\Gamma;\Delta',\Delta'' \vdash_\Sigma U \,\hat{;}\, S : A \multimap B > a}\,\textsf{lS\_lapp} \qquad \frac{\Gamma;\cdot \vdash_\Sigma U : A \quad \Gamma;\Delta \vdash_\Sigma S : B > a}{\Gamma;\Delta \vdash_\Sigma U ; S : A \to B > a}\,\textsf{lS\_iapp}$$

Figure 2: Typing for η-Long $S^{\to\multimap\&\top}$ Terms

introduces commutative conversions, it is the largest linear λ-calculus for which strong normalization holds and yields unique normal forms. However, non-standard presentations bypass commutative conversions and therefore extend the class of strongly normalizing languages (for example allowing $\otimes$ as a type constructor), although at the cost of added complexity [Min98]. We will not pursue this thread.

A term M of type A is in *η-long form* if it is structured as a sequence consisting solely of constructors (abstractions, pairing and unit) that matches the structure of the type A, applied to *atomic* terms in those positions where objects of base type are required. An atomic term consists of a sequences of destructors (applications and projections) that ends with a constant, a variable or an η-long β-redex, where the argument part of each application is required to be itself an η-long term. This definition extends the usual notion of η-long term of $\lambda^{\to}$ to the linear type operators $\multimap$, $\&$ and $\top$ of $\lambda^{\to\multimap\&\top}$. For example, in a context consisting solely of the assumption $x\!:\!A$, for $A = a \mathbin{\&} (a \multimap a)$,

$$M = \langle \textsc{fst}\ x, \hat{\lambda} y\!:\!a.\, (\textsc{snd}\ x)\,\hat{}\,y\rangle$$

is an η-long term of type A. Indeed M starts with a paring construct that matches the conjunction in A, its left component, which has base type a, is atomic, and its right component is itself an η-long term of type $a \multimap a$. Instead x by itself is not an η-long term of type A. The unit type $\top$ manifests an interesting behavior since there is a unique η-long term of that type, namely $\langle\rangle$. As in $\lambda^{\to}$, every well-typed term in our language has a corresponding η-long form, called its *η-expansion*. The η-long form of x above is the term M, while every term of type $\top$ is expanded to $\langle\rangle$.

We write $\mathrm{Can}(M)$ for the *canonical form* of the $\lambda^{\to\multimap\&\top}$ term M, defined as the η-expansion of its β-normal form. Notice that $\mathrm{Can}(x)$ corresponds to the η-long form of the variable x. In the following, we will insist in dealing always with fully η-expanded terms.

2.2 The Spine Calculus

Unification algorithms base a number of choices on the nature of the heads of the terms to be unified. The head is immediately available in the first-order case, and still discernible in $\lambda^{\to}$ since every η-long normal term has the form

$$\lambda x_1 : A_1. \ldots \lambda x_n : A_n. h\, M_1 \ldots M_m$$

where the head h is a constant or a variable and $(h\, M_1 \ldots M_m)$ has base type. The usual parentheses saving conventions hide the fact that h is indeed deeply buried in the sequence of application and therefore not immediately accessible. A similar notational trick fails in $\lambda^{\rightarrow -\circ \& \top}$ since on the one hand a term of compound type can have several heads (e.g. c_1 and c_2 in $\langle c_1, c_2 \rangle$), possibly none (e.g. $\langle\rangle$), and on the other hand destructors can be interleaved arbitrarily in a term of base type (e.g. $\textsc{fst}\ ((\textsc{snd}\ c)\,\hat{}\,x\ y))$

The *spine calculus* $S^{\rightarrow -\circ \& \top}$ [CP97] permits recovering both efficient head accesses and notational convenience. Every $\lambda^{\rightarrow -\circ \& \top}$ term M of base type is written in this presentation as a *root* $H \cdot S$, where H corresponds to the head of M and the *spine* S collects the sequence of destructors applied to it. For example, $M = (h\, M_1 \ldots M_m)$ is written $U = h \cdot (U_1; \ldots U_m; \textsc{nil})$ in this language, where ";" represents application, $\textsc{nil}$ identifies the end of the spine, and U_i is the translation of M_i. Application and ";" have opposite associativity so that M_1 is the innermost subterm of M while U_1 is outermost in the spine of U. This approach was suggested by an empirical study of higher-order logic programs based on $\lambda^{\rightarrow}$ terms [MP92] and is reminiscent of the notion of abstract Böhm trees [Her95a, Her95b]; its practical merits in our setting are currently assessed in an experimental implementation. The following grammar describes the syntax of $S^{\rightarrow -\circ \& \top}$: we write constructors as in $\lambda^{\rightarrow -\circ \& \top}$, but use new symbols to distinguish a spine operator from the corresponding term destructor.

$$\begin{array}{llll}
\textit{Terms:}\ U ::= & H \cdot S & \textit{Spines:}\ S ::= & \textsc{nil} \qquad\qquad \textit{Heads:}\ H ::= c \mid x \mid U \\
 & \mid \lambda x : A. U & & \mid U ; S \\
 & \mid \hat{\lambda} x : A. U & & \mid U \,\hat{;}\, S \\
 & \mid \langle U_1, U_2 \rangle & & \mid \pi_1\, S \mid \pi_2\, S \\
 & \mid \langle\rangle & &
\end{array}$$

We adopt the same syntactic conventions as in $\lambda^{\rightarrow -\circ \& \top}$ and often write V for terms in $S^{\rightarrow -\circ \& \top}$. Terms are allowed as heads in order to construct β-redices. Indeed, a normal term has either a constant or a variable as its heads.

The typing judgments for terms and spines are denoted as follows:

$\Gamma; \Delta \vdash_\Sigma U : A$	*U is a term of type A in $\Gamma; \Delta$ and Σ*
$\Gamma; \Delta \vdash_\Sigma S : A > a$	*S is a spine from heads of type A to terms of type a in $\Gamma; \Delta$ and Σ*

The latter expresses the fact that given a head H of type A, the root $H \cdot S$ has type a. Notice that the target type of a well-typed spine is a base type. This has the desirable effect of permitting only η-long terms to be derivable in this calculus [CP97]: allowing arbitrary types on the right-hand side of the spine typing judgment corresponds to dropping this property, as we will see in Section 2.5. Abstract Böhm trees [Bar80, Her95a] are obtained in this manner.

The mutual definition of the two typing judgments of $S^{\rightarrow -\circ \& \top}$ is given in Figure 2. The opposite associativity that characterizes the spine calculus with respect to the more traditional formulation is reflected in the manner types are managed in the lower part of this figure.

There exists a structural translation of terms in $\lambda^{\rightarrow -\circ \& \top}$ to terms in $S^{\rightarrow -\circ \& \top}$, and vice versa. This mapping and the proofs of soundness and completeness for the respective typing derivations can be found in [CP97].

In the sequel, we will need the following simple property of typing derivations, which states that the intuitionistic context of any valid derivation can be arbitrarily weakened.

Lemma 2.1 (*Intuitionistic weakening*)

i. If $\Gamma; \Delta \vdash_\Sigma U : A$, *then for any context* Γ', *there is a derivation of* $\Gamma, \Gamma'; \Delta \vdash_\Sigma U : A$.

ii. If $\Gamma; \Delta \vdash_\Sigma S : A > a$, *then for any context* Γ', *there is a derivation of* $\Gamma, \Gamma'; \Delta \vdash_\Sigma S : A > a$. □

On the basis of this result, it is a simple matter to prove the following lemma, that we will need in the sequel. It states that linear hypotheses can be viewed as intuitionistic assumptions with additional properties. An analogous result is proved in [Cer96]. Clearly, the reverse property does not hold.

Reductions

$$\frac{}{(H \cdot S) \cdot \mathrm{NIL} \longrightarrow H \cdot S}\ \textbf{Sr\_nil}$$

$$\frac{}{\langle U, V\rangle \cdot (\pi_1\, S) \longrightarrow U \cdot S}\ \textbf{Sr\_beta\_fst} \qquad \frac{}{\langle U, V\rangle \cdot (\pi_2\, S) \longrightarrow V \cdot S}\ \textbf{Sr\_beta\_snd}$$

$$\frac{}{(\hat{\lambda} x\!:\!A.\,U) \cdot V \mathbin{\hat{;}} S \longrightarrow [V/x]U \cdot S}\ \textbf{Sr\_beta\_lin} \qquad \frac{}{(\lambda x\!:\!A.\,U) \cdot V ; S \longrightarrow [V/x]U \cdot S}\ \textbf{Sr\_beta\_int}$$

Congruences *terms*

$$\frac{S \longrightarrow S'}{c \cdot S \longrightarrow c \cdot S'}\ \textbf{Sr\_con} \qquad \frac{S \longrightarrow S'}{x \cdot S \longrightarrow x \cdot S'}\ \textbf{Sr\_var}$$

$$\frac{U \longrightarrow U'}{U \cdot S \longrightarrow U' \cdot S}\ \textbf{Sr\_redex1} \qquad \frac{S \longrightarrow S'}{U \cdot S \longrightarrow U \cdot S'}\ \textbf{Sr\_redex2}$$

$$\frac{U \longrightarrow U'}{\langle U, V\rangle \longrightarrow \langle U', V\rangle}\ \textbf{Sr\_pair1} \qquad \frac{V \longrightarrow V'}{\langle U, V\rangle \longrightarrow \langle U, V'\rangle}\ \textbf{Sr\_pair2}$$

$$\frac{U \longrightarrow U'}{\hat{\lambda} x\!:\!A.\,U \longrightarrow \hat{\lambda} x\!:\!A.\,U'}\ \textbf{Sr\_llam} \qquad \frac{U \longrightarrow U'}{\lambda x\!:\!A.\,U \longrightarrow \lambda x\!:\!A.\,U'}\ \textbf{Sr\_ilam}$$

spines

$$\frac{S \longrightarrow S'}{\pi_1\, S \longrightarrow \pi_1\, S'}\ \textbf{Sr\_fst} \qquad \frac{S \longrightarrow S'}{\pi_2\, S \longrightarrow \pi_2\, S'}\ \textbf{Sr\_snd}$$

$$\frac{U \longrightarrow U'}{U \mathbin{\hat{;}} S \longrightarrow U' \mathbin{\hat{;}} S}\ \textbf{Sr\_lapp1} \qquad \frac{S \longrightarrow S'}{U \mathbin{\hat{;}} S \longrightarrow U \mathbin{\hat{;}} S'}\ \textbf{Sr\_lapp2}$$

$$\frac{U \longrightarrow U'}{U ; S \longrightarrow U' ; S}\ \textbf{Sr\_iapp1} \qquad \frac{S \longrightarrow S'}{U ; S \longrightarrow U ; S'}\ \textbf{Sr\_iapp2}$$

Iteration

$$\frac{U \longrightarrow V}{U \longrightarrow^* V}\ \textbf{Sr*\_Sr} \qquad \frac{}{U \longrightarrow^* U}\ \textbf{Sr*\_refl} \qquad \frac{U \longrightarrow^* U' \quad U' \longrightarrow^* U''}{U \longrightarrow^* U''}\ \textbf{Sr*\_trans}$$

Figure 3: Reduction Semantics for $S^{\rightarrow\multimap\&\top}$

Lemma 2.2 (*Promotion*)

i. *If* $\Gamma; \Delta, x\!:\!B \vdash_\Sigma U : A$, *then there is a derivation of* $\Gamma, x\!:\!B; \Delta \vdash_\Sigma U : A$.

ii. *If* $\Gamma; \Delta, x\!:\!B \vdash_\Sigma S : A > a$, *then there is a derivation of* $\Gamma, x\!:\!B; \Delta \vdash_\Sigma S : A > a$. □

The reduction semantics of $S^{\rightarrow\multimap\&\top}$ is based on the following β-reductions, which are obtained from the analogous rules of $\lambda^{\rightarrow\multimap\&\top}$ [CP96, CP97] by means of the mentioned translation.

$$\begin{array}{rcl}
\langle U, V\rangle \cdot (\pi_1\, S) & \longrightarrow & U \cdot S \\
\langle U, V\rangle \cdot (\pi_2\, S) & \longrightarrow & V \cdot S \\
(\hat{\lambda} x\!:\!A.\,U) \cdot (V \mathbin{\hat{;}} S) & \longrightarrow & [V/x]U \cdot S \\
(\lambda x\!:\!A.\,U) \cdot (V ; S) & \longrightarrow & [V/x]U \cdot S
\end{array}$$

The trailing spine in the reductions for $S^{\rightarrow\multimap\&\top}$ is a consequence of the fact that this language reverses the nesting order of $\lambda^{\rightarrow\multimap\&\top}$ destructors. The structure of roots in the spine calculus makes one more β-reduction rule necessary, namely:

$$(H \cdot S) \cdot \mathrm{NIL} \quad \longrightarrow \quad H \cdot S$$

For future reference, we give the complete rule set for reduction in $S^{\rightarrow\multimap\&\top}$ in Figure 3. We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$. It is easy to prove that the inference rules obtained by systematically replacing $\longrightarrow$ with $\longrightarrow^*$ in this figure are admissible. In particular, we will make implicit uses of the transitivity rule to build chains of reductions.

In most of this paper, we will insist on terms being in η-long form. Enforcing this requirement and maintaining it as an invariant of the operations we consider will have the beneficial effect of simplifying considerably the discussion. Indeed, while working around extensionality leads only to minor complications for function and product types, accomodating the unit type ($\top$) requires a large amount of machinary and elaborate techniques. Furthermore, an implementation that works on η-long terms only can be essentially type-free, while a program that performs η-expansion at run-time needs typing information pervasively.

As a result of working with η-long terms only, roots have always base type and so do the target types in the spine typing judgment. The β-reduction rules above preserve not only well-typedness, but also long forms so that η-expansion steps never need to be performed. This property is formalized in the following lemma, whose proof can be found in [CP97].

Lemma 2.3 (*Subject reduction*)

i. If $\Gamma; \Delta \vdash_\Sigma U : A$ *and* $U \longrightarrow V$, *then* $\Gamma; \Delta \vdash_\Sigma V : A$.

ii. If $\Gamma; \Delta \vdash_\Sigma S : A > a$ *and* $S \longrightarrow S'$, *then* $\Gamma; \Delta \vdash_\Sigma S' : A > a$. □

The following technical result is proved as in $\lambda^{\rightarrow\multimap\&\top}$ [Cer96].

Lemma 2.4 (*Substitution*)

i. If $U \longrightarrow^* U'$ *and* $V \longrightarrow^* V'$, *then* $[V/x]U \longrightarrow^* [V'/x]U'$.

ii. If $S \longrightarrow^* S'$ *and* $V \longrightarrow^* V'$, *then* $[V/x]S \longrightarrow^* [V'/x]S'$. □

Similarly to $\lambda^{\rightarrow\multimap\&\top}$, the spine calculus is confluent, i.e. every two sequences of reductions at a term (spine) can be extended to a common reduct. This fact is formalized in the following theorem [CP97].

Theorem 2.5 (*Confluence*)

i. If $U \longrightarrow^* U_1$ *and* $U \longrightarrow^* U_2$, *then there is a term* U' *such that* $U_1 \longrightarrow^* U'$ *and* $U_2 \longrightarrow^* U'$.

ii. If $S \longrightarrow^* S_1$ *and* $S \longrightarrow^* S_2$, *then there is a spine* S' *such that* $S_1 \longrightarrow^* S'$ *and* $S_2 \longrightarrow^* S'$. □

Moreover, every reduction sequence necessarily terminates when starting from a well-typed term or spine. We have indeed the following theorem, proved in [CP97].

Theorem 2.6 (*Strong normalization*)

i. If $\Gamma; \Delta \vdash_\Sigma U : A$, *then* U *is strongly normalizing.*

ii. If $\Gamma; \Delta \vdash_\Sigma S : A > a$, *then* S *is strongly normalizing.* □

With these two theorems, we easily prove that every well-typed term in $S^{\rightarrow\multimap\&\top}$ has a unique canonical form with respect to the notion of reduction given in Figure 3. We write $\mathrm{Can}(U)$ for the canonical form of the term U with respect to these reductions, and similarly for spines.

2.3 Head-Normal Forms in the Spine Calculus

We call two $S^{\rightarrow\multimap\&\top}$ terms *equal* if it is possible to rewrite them to a common reduct by means of the rules in Figure 3. Our notion of equality is therefore syntactic equality considered modulo β-reduction (recall that we assume to start always with terms in η-long form). The problem of whether two terms are equal is undecidable in the general case, in particular in the presence of ill-typed terms. Indeed, while recognizing two equal terms as such can always be done in a finite number of steps, establishing that they differ can go beyond the power of automation if these terms admit infinite chains of reductions.

This issue does not arise if we limit our attention to well-typed terms (as we do in this paper) since, by the strong normalization theorem 2.6, every reduction sequence starting at a typable term necessarily ends with a canonical form after finitely many steps. Since canonical forms are unique, a simple way to decide whether two terms U_1 and U_2 satisfy our notion of equality is to compute their canonical form and check whether $\mathrm{Can}(U_1)$ and $\mathrm{Can}(U_2)$ are syntactically equal (modulo renaming of bound variables, as always).

If U_1 and U_2 are indeed equal, then this method is often very efficient. However, it performs poorly on average since it might do large amounts of unnecessary computation when they are not equal. Assume for example that U_1 and U_2 are the root terms $c_1 \cdot S_1$ and $c_2 \cdot S_2$, respectively, with c_1 and c_2 different constants and S_1 and S_2 some (possibly very complex) spines. Then, looking at the heads of U_1 and U_2 suffices to establish that they cannot be reduced to a common term. Computing their canonical form requires instead visiting the whole terms and possibly reducing deep redices unnecessarily. Reduction to canonical form performs poorly also when used in unification, as we will see in the next section. Intuitively, a solution is computed in stages and each stage produces a redex that needs to be normalized in order to proceed. Using reduction to canonical form for this purpose is inefficient since it would cause the same term to be traversed over and over.

We overcome these deficiencies by considering *head-normal forms.* A term is head-normal if it is canonical except for the possible presence of β-redices within a spine, i.e. in an argument position. Head-normal roots are called *weakly head-normal terms* and will be our primary focus. A (weakly) head-normal term consists therefore of a superficial layer that is redex-free and a deeper layer that is arbitrary. Canonical terms are simply hereditarily head-normal, and reduction to canonical form can be implemented by iterated reductions to head-normal form with the advantage that each stage of the process can be interleaved with other operations, such as detecting failure in an equality test, or equation simplification in a unification problem.

In this section, we will study head-normal forms and discuss an algorithm to achieve them. The results below hold in particular in the more specific case of weakly head-normal term. We will apply the latter notion to improve our naive equality test in Section 2.4. Its applications in the context of unification will appear in Section 3.

The basic reduction relation $\longrightarrow$, given in Figure 3, is built by congruence over the five β-reduction rules of $S^{\rightarrow\multimap\&\top}$ and constitutes the basis of the notion of canonical form. The reduction relation consisting solely of these β-reduction rules is called *weak head-reduction* and will be indicated as $\stackrel{whr}{\longrightarrow}$. It is only applicable to terms that are roots, and therefore of base type since we operate on η-long terms only. We formally define it in the upper part of Figure 4. Its reflexive and transitive closure, denoted $\stackrel{whr}{\longrightarrow}{}^*$, permits forming chains of basic β-reductions. It can be easily proved that the rules obtained by replacing $\stackrel{whr}{\longrightarrow}$ with $\stackrel{whr}{\longrightarrow}{}^*$ in this figure are admissible.

Head-normal terms draw their origin from the *head-reduction relation*, that we indicate as $\stackrel{hr}{\longrightarrow}$. It builds on weak head-reduction by congruence over the term constructors of $S^{\rightarrow\multimap\&\top}$, and therefore operates on terms that are not necessary of base type. In particular, root boundaries are never crossed and it is not defined on spines. This relation is formalized in Figure 4. We write $\stackrel{hr}{\longrightarrow}{}^*$ for its reflexive and transitive closure, which definition is given at the bottom of this figure. As with weak head-reduction, the rules obtained by replacing $\stackrel{hr}{\longrightarrow}$ with $\stackrel{hr}{\longrightarrow}{}^*$ in this figure are admissible.

Observe that weak head-reduction coincides with the head-reduction relation for roots. Therefore, by virtue of the subject reduction lemma below, every property of the latter relation holds (sometimes trivially) for its weak counterpart. In the sequel, we will rely exclusively on the weak head-reduction

Weak head−reductions

$$\dfrac{}{(H\cdot S)\cdot \mathrm{NIL} \xrightarrow{whr} H\cdot S}\;\textbf{whr\_nil}$$

$$\dfrac{}{\langle U,V\rangle\cdot(\pi_1\, S) \xrightarrow{whr} U\cdot S}\;\textbf{whr\_beta\_fst} \qquad \dfrac{}{\langle U,V\rangle\cdot(\pi_2\, S) \xrightarrow{whr} V\cdot S}\;\textbf{whr\_beta\_snd}$$

$$\dfrac{}{(\hat{\lambda} x\!:\!A.\,U)\cdot V \,\hat{;}\, S \xrightarrow{whr} [V/x]U\cdot S}\;\textbf{whr\_beta\_lin} \qquad \dfrac{}{(\lambda x\!:\!A.\,U)\cdot V; S \xrightarrow{whr} [V/x]U\cdot S}\;\textbf{whr\_beta\_int}$$

Congruences (*head−reduction only*)

$$\dfrac{U \xrightarrow{whr} U'}{U \xrightarrow{hr} U'}\;\textbf{hr\_whr}$$

$$\dfrac{U \xrightarrow{hr} U'}{\langle U,V\rangle \xrightarrow{hr} \langle U',V\rangle}\;\textbf{hr\_pair1} \qquad \dfrac{V \xrightarrow{hr} V'}{\langle U,V\rangle \xrightarrow{hr} \langle U,V'\rangle}\;\textbf{hr\_pair2}$$

$$\dfrac{U \xrightarrow{hr} U'}{\hat{\lambda} x\!:\!A.\,U \xrightarrow{hr} \hat{\lambda} x\!:\!A.\,U'}\;\textbf{hr\_llam} \qquad \dfrac{U \xrightarrow{hr} U'}{\lambda x\!:\!A.\,U \xrightarrow{hr} \lambda x\!:\!A.\,U'}\;\textbf{hr\_ilam}$$

Iteration (*weak head−reduction*)

$$\dfrac{U \xrightarrow{whr} V}{U \xrightarrow{whr}{}^{*}\, V}\;\textbf{whr*\_whr} \qquad \dfrac{}{U \xrightarrow{whr}{}^{*}\, U}\;\textbf{whr*\_refl} \qquad \dfrac{U \xrightarrow{whr}{}^{*}\, U' \quad U' \xrightarrow{whr}{}^{*}\, U''}{U \xrightarrow{whr}{}^{*}\, U''}\;\textbf{whr*\_trans}$$

Iteration (*head−reduction*)

$$\dfrac{U \xrightarrow{hr} V}{U \xrightarrow{hr}{}^{*}\, V}\;\textbf{hr*\_hr} \qquad \dfrac{}{U \xrightarrow{hr}{}^{*}\, U}\;\textbf{hr*\_refl} \qquad \dfrac{U \xrightarrow{hr}{}^{*}\, U' \quad U' \xrightarrow{hr}{}^{*}\, U''}{U \xrightarrow{hr}{}^{*}\, U''}\;\textbf{hr*\_trans}$$

Figure 4: (Weak) Head-Reduction for $S^{\rightarrow\multimap\&\top}$

relation, although in this section we will study the more general head-reduction relation.

Notice that, beyond the arrow decoration, the rules for $\xrightarrow{hr}$ displayed in Figure 4 are a subset of the rules given for $\longrightarrow$ in Figure 3. This implies that (weak) head-reduction is a sub-relation of the general notion of reduction for $S^{\rightarrow\multimap\&\top}$. This simple fact is formally expressed in the following lemma.

Lemma 2.7 (*Reduction subsumes head-reduction*)

If $U \xrightarrow{hr} U'$*, then* $U \longrightarrow U'$.

Proof.

The formal proof proceeds by induction on the structure of a derivation $\mathcal{W}$ of $U \xrightarrow{hr} U'$. ☑

Head-reduction and its weak variant enjoy many of the properties that hold for $\longrightarrow$, and similarly for their reflexive and transitive closures. The above lemma permits significant simplifications of their otherwise rather involved proofs. The first of these results is an adaptation of the strong normalization theorem. Notice that this result is stated for terms only, and not for spines.

Theorem 2.8 (*Strong normalization for head-reduction*)

If $\Gamma;\Delta \vdash_\Sigma U : A$*, then* U *is strongly normalizing with respect to* $\xrightarrow{hr}$.

Proof.

Assume we have a (possibly infinite) sequence of terms $U_0, U_1, U_2, \ldots$ such that $U = U_0$ and there are derivations for each of the following reductions:

$$\sigma = U_0 \xrightarrow{hr} U_1 \xrightarrow{hr} U_2 \xrightarrow{hr} \ldots$$

Since, by Lemma 2.7, every head-reduction derivation corresponds trivially to a valid reduction derivation, the following sequence of reductions is derivable:

$$\sigma' = U_0 \longrightarrow U_1 \longrightarrow U_2 \longrightarrow \ldots$$

By the strong normalization property for $\longrightarrow$, σ' must be finite. Therefore, also σ must be finite. ☑

Next, we prove that $\xrightarrow{hr}$ is confluent, i.e. that if a head-reduction is applicable in two positions in a term, then the resulting terms can be reduced to a common reduct by a further application (unless they are already identical). Here and in the sequel, we abbreviate the phrases "the judgment J has derivation $\mathcal{J}$" and "there is a derivation $\mathcal{J}$ for the judgment J" as $\mathcal{J} :: J$.

Lemma 2.9 (*Local confluence for head-reduction*)

If $\mathcal{W}' :: U \xrightarrow{hr} U'$ *and* $\mathcal{W}'' :: U \xrightarrow{hr} U''$, *then either* $U' = U''$, *or there is a term* V *such that* $U' \xrightarrow{hr} V$ *and* $U'' \xrightarrow{hr} V$.

Proof.

$\mathcal{W}'$ and $\mathcal{W}''$ can differ only if U contains a subterm $\hat{U}$ of the form $\langle \hat{U}_1, \hat{U}_2 \rangle$ and the two derivations proceed by head-reducing different components of this pair. Assume for instance that $\hat{U}_1$ is reduced to $\hat{U}_1'$ in $\mathcal{W}'$, and $\hat{U}_2$ is reduced to $\hat{U}_2'$ in $\mathcal{W}''$. Then U' will contain $\hat{U}' = \langle \hat{U}_1', \hat{U}_2 \rangle$ and U'' will contain $\hat{U}'' = \langle \hat{U}_1, \hat{U}_2' \rangle$. We now obtain V by reducing both $\hat{U}'$ and $\hat{U}''$ to $\hat{V} = \langle \hat{U}_1, \hat{U}_2' \rangle$.

Formally, the proof proceeds by simultaneous induction on the structure of $\mathcal{W}'$ and $\mathcal{W}''$. ☑

When restricting our attention to weak head-reduction in the above lemma, the existence of $\mathcal{W}'$ and $\mathcal{W}''$ implies that $U' = U''$ since every term of base type can start at most one head-reduction sequence.

Well-known results in term-rewriting theory [DJ90] permit lifting this property, in the presence of termination (Theorem 2.8 here), to the reflexive and transitive closure of the above relation.

Lemma 2.10 (*Confluence of head-reduction*)

If $\mathcal{W}' :: U \xrightarrow{hr}{}^* U'$ *and* $\mathcal{W}'' :: U \xrightarrow{hr}{}^* U''$, *then there is a term* V *such that* $U' \xrightarrow{hr}{}^* V$ *and* $U'' \xrightarrow{hr}{}^* V$. □

We are now in a position to prove the uniqueness of head-normal forms: by strong normalization every well-typed term admits only finitely many head-reductions, however the term that is eventually produced is the same no matter which redex we start with.

Theorem 2.11 (*Uniqueness of head-normal forms*)

If $\Gamma; \Delta \vdash_\Sigma U : A$, *then there is a unique head-normal term* V *such that* $U \xrightarrow{hr}{}^* V$.

Proof.

By the strong normalization theorem 2.8, we know that every sequence of reductions starting at U leads to a term in head-normal form. Let us consider two reduction sequences validating $U \xrightarrow{hr}{}^* V'$ and $U \xrightarrow{hr}{}^* V''$, for terms V' and V'' in head-normal form. By confluence, there is a term V to which both head-reduce. However, since there is no head-reduction derivation starting at either V' or V'', the only way to close the diamond is to have that $V' = V'' = V$, and use rule **hr*_refl**. ☑

This theorem entitles us to speak about *the* head-normal form of a well-typed term U. We will indicate this object as $\mathrm{HNF}(U)$ for the moment.

We would like now to characterize the structure of the head-normal forms $\mathrm{HNF}(U)$ computable with the rules in Figure 4. In particular, we want to verify that it corresponds to the informal definition given at the beginning of this section. Prior to doing so, we need to show that the head-reduction relation respects typing and extensionality. We have the following subject reduction lemma.

Lemma 2.12 (*Subject reduction for head-reduction*)

If $\Gamma; \Delta \vdash_\Sigma U : A$ *and* $U \xrightarrow{hr} U'$, *then* $\Gamma; \Delta \vdash_\Sigma U' : A$.

Proof.

By the subsumption lemma 2.7, there is a derivation of $U \longrightarrow U'$. Then, by the subject reduction theorem 2.3, $\Gamma; \Delta \vdash_\Sigma U' : A$. ☑

This result extends to the reflexive and transitive closure of $\xrightarrow{hr}$.

The following lemma entails that, in a head-normal $S^{\rightarrow\multimap\&\top}$ term, redices are confined within spines. Indeed, the only atomic (weakly) head-normal terms are roots with a constant or a variable as their head: redices are excluded.

Lemma 2.13 (*Characterization of head-normal forms*)

If $\Gamma; \Delta \vdash_\Sigma U : A$ *and* $V = \mathrm{HNF}(U)$, *then*

- *if* $A = a$, *then either* $V = c \cdot S$ *or* $V = x \cdot S$.
- *if* $A = \top$, *then* $V = \langle\rangle$;
- *if* $A = A_1 \& A_2$, *then* $V = \langle V_1, V_2 \rangle$ *and* V_1 *and* V_2 *are in head-normal form;*
- *if* $A = A \multimap B$, *then* $V = \hat{\lambda} x : A. V'$ *and* V' *is in head-normal form;*
- *if* $A = A \rightarrow B$, *then* $V = \lambda x : A. V'$ *and* V' *is in head-normal form.*

Proof.

By iterated applications of the subject reduction lemma 2.12, we know that there is a derivation $\mathcal{U}$ of $\Gamma; \Delta \vdash_\Sigma V : A$. We proceed then by inversion on the structure of $\mathcal{U}$. In particular, if A is a base type, it must be the case that $V = c \cdot S$ or $V = x \cdot S$, otherwise, V would not be in head-normal form. ☑

The above results imply that head-normalization is a total function from typable $S^{\rightarrow\multimap\&\top}$ terms U to objects in head-normal form $\mathrm{HNF}(U)$. We want now to give an explicit functional definition for this operation. To this end, we propose the function $(\overline{\ldots})$ defined as follows.

$$\begin{array}{rcl} \overline{\langle\rangle} & = & \langle\rangle \\ \overline{\langle U, V\rangle} & = & \langle \overline{U}, \overline{V}\rangle \\ \overline{\hat{\lambda} x : A. U} & = & \hat{\lambda} x : A. \overline{U} \\ \overline{\lambda x : A. U} & = & \lambda x : A. \overline{U} \\ \overline{c \cdot S} & = & c \cdot S \\ \overline{x \cdot S} & = & x \cdot S \end{array} \qquad \begin{array}{rcl} \overline{(H \cdot S) \cdot \mathrm{NIL}} & = & \overline{H \cdot S} \\ \overline{\langle U, V\rangle \cdot (\pi_1 S)} & = & \overline{U \cdot S} \\ \overline{\langle U, V\rangle \cdot (\pi_2 S)} & = & \overline{V \cdot S} \\ \overline{(\hat{\lambda} x : A. U) \cdot (V \mathbin{\hat{;}} S)} & = & \overline{[V/x]U \cdot S} \\ \overline{(\lambda x : A. U) \cdot (V ; S)} & = & \overline{[V/x]U \cdot S} \end{array}$$

We need to show that $(\overline{\ldots})$ actually computes the head-normal form of any well-typed $S^{\rightarrow\multimap\&\top}$ term. We have the following soundness result: if $V = \overline{U})$, then U head-reduces to V.

Lemma 2.14 (*Soundness of* $(\overline{\ldots})$)

If there is a term V *such that* $\overline{U} = V$, *then* $U \xrightarrow{hr}{}^{*} V$.

Proof.

By induction on the computation of $\overline{U}$. ☑

The completeness property below states that $(\overline{\ldots})$ computes precisely head-normal forms.

Lemma 2.15 (*Completeness of* $(\overline{\ldots})$)

If $U \xrightarrow{hr}{}^{*} V$ *and* V *is in head-normal form, then* $\overline{U}$ *is defined and* $\overline{U} = V$.

Proof.

This proof proceeds in two steps.

1. Every derivation of $U \xrightarrow{hr}{}^{*} V$ can be transformed into a derivation of the same judgment such that:

   - Reflexivity (rule **hr*_refl**) is only applied to terms of the form $\langle\rangle$, $c \cdot S$ or $x \cdot S$;
   - Transitivity (rule **hr*_trans**) is only applied either to terms of base type (roots) or to pairs, in which case its right premiss ends in rule **hr_fst** and its left premiss ends in rule **hr_snd**.

   We omit the proof of this simple property.

2. Then, we proceed by induction on the structure of a derivation $\mathcal{W}$ of $U \xrightarrow{hr}{}^{*} V$ with the above characteristics. ☑

Notice that neither the soundness nor the completeness lemma above mention typing information. Their generality specializes to the well-typed terms we are interested in as a special case. Observe however that $\overline{(\ldots)}$ can diverge when applied to certain ill-typed terms.

Thanks to the subject reduction lemma 2.12, the above properties imply that, whenever applied to a (well-typed) term of base type, $\overline{(\ldots)}$ computes its weak head-normal form. Therefore, whenever U is some term of base type, $\overline{U}$ will denote its weak head-normal form.

We conclude this section by proving a technical lemma that establishes the connection between head-normalization and canonical forms. A head-normal form can be seen as an intermediate stage towards reaching a canonical form. By virtue of the strong normalization theorem above, this lemma justifies iterated head-normalization as a specific reduction strategy to canonical form.

Lemma 2.16 (*Connection between head-normal forms and canonical forms*)

If $\Gamma; \Delta \vdash_\Sigma U : A$ *and* $\overline{U} = V$, *then* $\mathrm{Can}(U) = \mathrm{Can}(V)$.

Proof.

By the soundness of $\overline{(\ldots)}$, since $\overline{U} = V$, we have that $U \xrightarrow{hr}{}^{*} V$ and consequently $U \longrightarrow^{*} V$ by subsumption. By subject reduction, we deduce that $\Gamma; \Delta \vdash_\Sigma V : A$ and therefore, by the strong normalization theorem 2.6, both $\mathrm{Can}(U)$ and $\mathrm{Can}(V)$ exist and $U \longrightarrow^{*} \mathrm{Can}(U)$ and $V \longrightarrow^{*} \mathrm{Can}(V)$. Now, since canonical forms are unique, we derive that $\mathrm{Can}(U) = \mathrm{Can}(V)$. ☑

2.4 Equality in the Spine Calculus

In the previous section, we defined two $S^{\rightarrow\multimap\&\top}$ terms U_1 and U_2 to be *equal* if they can be β-reduced to a common term V. We observed that, by strong normalization and the Church-Rosser theorem [CP97], it suffices to compute $\mathrm{Can}(U_1)$ and $\mathrm{Can}(U_2)$ and check whether they are syntactically equal (modulo renaming of bound variables). We noticed however that this method for testing equality involves a high overhead in case of failure, and that reduction to canonical form is inefficient when dealing with unification, a problem closely related to equality checking (see Section 3).

In this section, we propose an alternative algorithm for verifying that two $S^{\rightarrow\multimap\&\top}$ terms are equal. This efficient method is based on weak head-normalization and parallels the use of this form of reduction in the pre-unification algorithm discussed in Section 3. We will prove that it is indeed equivalent to the naive procedure based on comparing canonical forms.

This test, that we will sometimes identify as *staged equality*, is based on the following equality judgments for terms and spines, respectively:

$\Gamma; \Delta \vdash_\Sigma U_1 = U_2 : A$	U_1 *and* U_2 *are equal terms of type* A *in* $\Gamma; \Delta$ *and* Σ
$\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$	S_1 *and* S_2 *are equal spines from heads of type* A *to terms of type* a *in* $\Gamma; \Delta$ *and* Σ

Terms

$$\frac{\Gamma;\Delta \vdash_\Sigma \overline{U \cdot S_1} = H \cdot S_2 : a}{\Gamma;\Delta \vdash_\Sigma U \cdot S_1 = H \cdot S_2 : a}\ \textbf{Seq\_redex\_l} \qquad \frac{\Gamma;\Delta \vdash_\Sigma H \cdot S_1 = \overline{U \cdot S_2} : a}{\Gamma;\Delta \vdash_\Sigma H \cdot S_1 = U \cdot S_2 : a}\ \textbf{Seq\_redex\_r}$$

$$\frac{\Gamma;\Delta \vdash_{\Sigma,c:A} S_1 = S_2 : A > a}{\Gamma;\Delta \vdash_{\Sigma,c:A} c \cdot S_1 = c \cdot S_2 : a}\ \textbf{Seq\_con}$$

$$\frac{\Gamma;\Delta \vdash_\Sigma S_1 = S_2 : A > a}{\Gamma;\Delta, x\!:\!A \vdash_\Sigma x \cdot S_1 = x \cdot S_2 : a}\ \textbf{Seq\_lvar} \qquad \frac{\Gamma, x\!:\!A;\Delta \vdash_\Sigma S_1 = S_2 : A > a}{\Gamma, x\!:\!A;\Delta \vdash_\Sigma x \cdot S_1 = x \cdot S_2 : a}\ \textbf{Seq\_ivar}$$

$$\frac{}{\Gamma;\Delta \vdash_\Sigma \langle\rangle = \langle\rangle : \top}\ \textbf{Seq\_unit} \qquad \frac{\Gamma;\Delta \vdash_\Sigma U_1 = V_1 : A_1 \quad \Gamma;\Delta \vdash_\Sigma U_2 = V_2 : A_2}{\Gamma;\Delta \vdash_\Sigma \langle U_1, U_2\rangle = \langle V_1, V_2\rangle : A_1 \,\&\, A_2}\ \textbf{Seq\_pair}$$

$$\frac{\Gamma;\Delta, x\!:\!A \vdash_\Sigma U = V : B}{\Gamma;\Delta \vdash_\Sigma \hat{\lambda} x\!:\!A.\, U = \hat{\lambda} x\!:\!A.\, V : A \multimap B}\ \textbf{Seq\_llam} \qquad \frac{\Gamma, x\!:\!A;\Delta \vdash_\Sigma U = V : B}{\Gamma;\Delta \vdash_\Sigma \lambda x\!:\!A.\, U = \hat{\lambda} x\!:\!A.\, V : A \to B}\ \textbf{Seq\_ilam}$$

Spines

$$\frac{}{\Gamma;\cdot \vdash_\Sigma \mathrm{NIL} = \mathrm{NIL} : a > a}\ \textbf{Seq\_nil} \qquad \text{(No spine rule for } \top\text{)}$$

$$\frac{\Gamma;\Delta \vdash_\Sigma S_1 = S_2 : A_1 > a}{\Gamma;\Delta \vdash_\Sigma \pi_1\, S_1 = \pi_1\, S_2 : A_1 \,\&\, A_2 > a}\ \textbf{Seq\_fst} \qquad \frac{\Gamma;\Delta \vdash_\Sigma S_1 = S_2 : A_2 > a}{\Gamma;\Delta \vdash_\Sigma \pi_2\, S_1 = \pi_2\, S_2 : A_1 \,\&\, A_2 > a}\ \textbf{Seq\_snd}$$

$$\frac{\Gamma;\Delta' \vdash_\Sigma U_1 = U_2 : A \quad \Gamma;\Delta'' \vdash_\Sigma S_1 = S_2 : B > a}{\Gamma;\Delta', \Delta'' \vdash_\Sigma U_1 \mathbin{\hat{;}} S_1 = U_2 \mathbin{\hat{;}} S_2 : A \multimap B > a}\ \textbf{Seq\_lapp}$$

$$\frac{\Gamma;\cdot \vdash_\Sigma U_1 = U_2 : B \quad \Gamma;\Delta \vdash_\Sigma S_1 = S_2 : B > a}{\Gamma;\Delta \vdash_\Sigma U_1; S_1 = U_2; S_2 : A \to B > a}\ \textbf{Seq\_iapp}$$

Figure 5: Equality in $S^{\to\multimap\&\top}$

The inference rules defining them are given in Figure 5. These rules are type-directed and their correctness and termination rely heavily on the assumption that the involved terms are η-long and have a canonical form. This requirement entails the fact that two terms of compound type cannot be equal unless their top-level constructors are the same and their subterms are recursively equal; rules **Seq_unit** to **Seq_ilam** in the top part of this figure take advantage of this fact. A similar property applies to spines and is realized by the rules in the bottom part of Figure 5. This characterization is complete in the case of roots (the only terms of base type) only if both heads are a constant or a variable. If the head of either root is a generic term, we first need to reduce the resulting redex. In this situation, we avoid the drawbacks of reduction to canonical forms by using weak head-normalization in rules **Seq_redex_l** and **Seq_redex_r** (recall that the function $\overline{(\ldots)}$ computes weak head-normal forms when applied to terms of base type). This will have the effect of exposing a constant or a variable as the head of our terms. We will be able to compare these heads directly before verifying the equality of the associated spines (rules **Seq_con**, **Seq_lvar** and **Seq_ivar**). Redices possibly appearing in the latter will be handled similarly. This way of proceeding corresponds to imposing a reduction strategy guided by weak head-normalization in order to handle the redices occurring in terms.

The typing information in the equality judgments is convenient when proving properties, especially those concerning unification in the next section. It is however redundant as long as we assume that the terms we start with are η-long and have a canonical form. Therefore, it can safely be omitted altogether when implementing this procedure.

We call a derivation $\mathcal{E}$ for the equality judgment $\Gamma;\Delta \vdash_\Sigma U_1 = U_2 : A$ *well-typed* if there exist typing derivations $\mathcal{U}_1$ and $\mathcal{U}_2$ of $\Gamma;\Delta \vdash_\Sigma U_1 : A$ and $\Gamma;\Delta \vdash_\Sigma U_2 : A$, respectively. Notice that not every equality derivation is well-typed since the appeals to weak head-normalization in rules **Seq_redex_l** and **Seq_redex_r** might eliminate ill-typed subterms. This property holds however if U_1 and U_2 are in canonical form. Similar considerations apply to the spine equality judgment.

We will now prove that the deductive system given in Figure 5 does implement an equality test as defined at the beginning of the previous section. We first prove the soundness of this procedure, i.e. that every time it claims that two terms are equal, they actually are. The involved terms are not required to be well-typed.

Theorem 2.17 (*Soundness of staged equality*)

i. If $\mathcal{E} :: \Gamma; \Delta \vdash_\Sigma U_1 = U_2 : A$*, then* $\mathrm{Can}(U_1) = \mathrm{Can}(U_2)$*;*

ii. If $\mathcal{E} :: \Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$*, then* $\mathrm{Can}(S_1) = \mathrm{Can}(S_2)$*;*

Proof.
The proof proceeds by induction on the structure of the derivation $\mathcal{E}$. All cases match trivially the rules in Figure 3, except for derivations that end in **Seq_redex_l** or **Seq_redex_r**. In these cases, we take advantage of the connection lemma 2.16 and of transitivity. ☑

Next, we need to show that whenever two terms (or spines) are equal according to our definition, then there is a derivation for the corresponding staged equality judgment. We equip the statement of this theorem with typing assumptions to ensure the existence of the claimed canonical forms. This also establishes the origin of the type, contexts and signature appearing in the equality judgments. However, a more Spartan version of this theorem, devoid of any typing assumption, also holds: only the existence of a canonical form for the terms involved is required.

Theorem 2.18 (*Completeness of staged equality*)

i. Let $\mathcal{E}_1 :: \Gamma; \Delta \vdash_\Sigma U_1 : A$ *and* $\mathcal{E}_2 :: \Gamma; \Delta \vdash_\Sigma U_2 : A$*.*
If $\mathrm{Can}(U_1) = \mathrm{Can}(U_2)$*, then* $\Gamma; \Delta \vdash_\Sigma U_1 = U_2 : A$*.*

ii. Let $\mathcal{E}_1 :: \Gamma; \Delta \vdash_\Sigma S_1 : A > a$ *and* $\mathcal{E}_2 :: \Gamma; \Delta \vdash_\Sigma S_2 : A > a$*.*
If $\mathrm{Can}(S_1) = \mathrm{Can}(S_2)$*, then* $\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$*.*

Proof.
The proof proceeds by nested induction over computation of $\mathrm{Can}(U_1)$ and $\mathrm{Can}(U_2)$, measured as the sequence of β-reductions, from U_1 and U_2 (S_1 and S_2) to $\mathrm{Can}(U_1)$ and $\mathrm{Can}(U_2)$ respectively ($\mathrm{Can}(S_1)$ and $\mathrm{Can}(S_2)$ respectively), and the structures of $\mathcal{E}_1$ and $\mathcal{E}_2$. We distinguish cases depending on the last rule applied in $\mathcal{E}_1$ and $\mathcal{E}_2$, or equivalently on the structure of U_1 and U_2 (or S_1 and S_2).

Unless either derivation ends in rule **lS_redex**, the cases are handled trivially since each of these typing rules corresponds to a uniquely determined equality rule. The induction hypothesis can be applied to the premisses of these rules since the sequence of reductions does not change, but the involved derivations are simpler.

We map occurrences of rule **lS_redex** in $\mathcal{E}_1$ to applications of rule **Seq_redex_l**, and its occurrences in $\mathcal{E}_2$ to uses of **Seq_redex_r**. Rule **lS_redex** witnesses the presence of an exposed redex in U_1 (U_2) so that we can apply weak head-normalization to this term. By the subject reduction lemma 2.12, $\overline{U_1}$ ($\overline{U_2}$) has a typing derivation $\mathcal{E}'_1$ ($\mathcal{E}'_2$). We can therefore apply the induction hypothesis since the sequence of reductions is shorter, although the structure of $\mathcal{E}'_1$ ($\mathcal{E}'_2$) might be very different from that of $\mathcal{E}_1$ ($\mathcal{E}_2$). ☑

We conclude this section with a collection of properties of the equality judgments. More precisely, we establish that it is a congruence relation relative to the two terms it equates.

Lemma 2.19 (*Equality induces a congruence*)

- *Reflexivity:* *If* $\Gamma; \Delta \vdash_\Sigma U : A$*, then* $\Gamma; \Delta \vdash_\Sigma U = U : A$*. Similarly for spines.*
- *Symmetry:* *If* $\Gamma; \Delta \vdash_\Sigma U_1 = U_2 : A$*, then* $\Gamma; \Delta \vdash_\Sigma U_2 = U_1 : A$*. Similarly for spines.*

Partial roots

$$\frac{\Gamma;\Delta' \vdash_\Sigma U : A \quad \Gamma;\Delta'' \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A > B}{\Gamma;\Delta',\Delta'' \vdash_\Sigma U \cdot \widetilde{S} \mathrel{\tilde{:}} B}\;\textbf{pS\_redex}$$

$$\frac{\Gamma;\Delta \vdash_{\Sigma,c:A} \widetilde{S} \mathrel{\tilde{:}} A > B}{\Gamma;\Delta \vdash_{\Sigma,c:A} c \cdot \widetilde{S} \mathrel{\tilde{:}} B}\;\textbf{pS\_con} \qquad \frac{\Gamma;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A > B}{\Gamma;\Delta, x\!:\!A \vdash_\Sigma x \cdot \widetilde{S} \mathrel{\tilde{:}} B}\;\textbf{pS\_lvar} \qquad \frac{\Gamma, x\!:\!A;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A > B}{\Gamma, x\!:\!A;\Delta \vdash_\Sigma x \cdot \widetilde{S} \mathrel{\tilde{:}} B}\;\textbf{pS\_ivar}$$

Partial spines

$$\frac{}{\Gamma;\cdot \vdash_\Sigma \textsc{nil} \mathrel{\tilde{:}} A > A}\;\textbf{pS\_nil}$$

(No pseudo-spine rule for $\top$)
$$\frac{\Gamma;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A_1 > B}{\Gamma;\Delta \vdash_\Sigma \pi_1\,\widetilde{S} \mathrel{\tilde{:}} A_1 \,\&\, A_2 > B}\;\textbf{pS\_fst} \qquad \frac{\Gamma;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A_2 > B}{\Gamma;\Delta \vdash_\Sigma \pi_2\,\widetilde{S} \mathrel{\tilde{:}} A_1 \,\&\, A_2 > B}\;\textbf{pS\_snd}$$

$$\frac{\Gamma;\Delta' \vdash_\Sigma U : A_1 \quad \Gamma;\Delta'' \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A_2 > B}{\Gamma;\Delta',\Delta'' \vdash_\Sigma U \mathbin{\hat{;}} \widetilde{S} \mathrel{\tilde{:}} A_1 \multimap A_2 > B}\;\textbf{pS\_lapp} \qquad \frac{\Gamma;\cdot \vdash_\Sigma U : A_1 \quad \Gamma;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A_2 > a}{\Gamma;\Delta \vdash_\Sigma U ; \widetilde{S} \mathrel{\tilde{:}} A_1 \rightarrow A_2 > B}\;\textbf{pS\_iapp}$$

Figure 6: Typing for Pseudo-Spines and Pseudo-Roots

- *Transitivity:* *If* $\Gamma;\Delta \vdash_\Sigma U_1 = U_2 : A$ *and* $\Gamma;\Delta \vdash_\Sigma U_2 = U_3 : A$, *then* $\Gamma;\Delta \vdash_\Sigma U_1 = U_3 : A$. *Similarly for spines*
- *Congruence:*
  - *If* $\Gamma;\Delta_1, x\!:\!A \vdash_\Sigma U : B$ *and* $\Gamma;\Delta_2 \vdash_\Sigma V_1 = V_2 : A$, *then* $\Gamma;\Delta_1,\Delta_2 \vdash_\Sigma [V_1/x]U = [V_2/x]U : B$.
  - *If* $\Gamma, x\!:\!A;\Delta \vdash_\Sigma U : B$ *and* $\Gamma;\cdot \vdash_\Sigma V_1 = V_2 : A$, *then* $\Gamma;\Delta \vdash_\Sigma [V_1/x]U = [V_2/x]U : B$.

  Similarly if the first assumption is a spine typing judgment.

Proof.

Reflexivity is a direct consequence of strong normalization, the uniqueness of canonical forms and the above completeness theorem.

The remaining properties are proved my means of simple inductive arguments. However, had we assumed that the terms they mention are well-typed, their validity would be a direct consequence of the soundness and completeness of staged equality. ☑

In the remainder of this paper, we will always assume that our equality derivations are well-typed and therefore omit explicit typing judgments for their sides.

2.5 Eta-Expansion in the Spine Calculus

In Section 2.1, we observed that, given a declaration $x : A$ the η-long form of a variable x corresponds to Can(x) in $\lambda^{\rightarrow\multimap\&\top}$. A similar notational trick is not viable in the spine calculus since a variable is a head while the reduction semantics of $S^{\rightarrow\multimap\&\top}$ is defined only for terms and spines. In this section, we will present a method for computing the η-long form of a variable at a given type, and prove typing and reduction properties about these objects. This method can easily be generalized to generate the η-long form of arbitrary $S^{\rightarrow\multimap\&\top}$ terms.

The procedure we will develop relies on the notion of *partial spine*, a technical device required to cope with the fact that, during η-expansion, spines are built from the outside in. Partial spines are syntactically undistinguishable from spines (see the definition in Section 2.2), but they obey a different typing semantics: they lift the requirement that the target type of a well-typed $S^{\rightarrow\multimap\&\top}$ spine be a base type. We rely on the symbol $\widetilde{S}$, possibly subscripted, as a syntactic variable for partial spines.

We will also make use of objects that differ from roots for the fact that they pair up an $S^{\rightarrow\multimap\&\top}$ head H, as defined in Section 2.2, and a partial spine $\widetilde{S}$. Such entities, called *partial roots*, are denoted $H \cdot \widetilde{S}$.

The distinguishing characteristic of the typing policy of the entities we just introduced with respect to the related $S^{\rightarrow\multimap\&\top}$ concepts is that partial roots are not required to be of base type. Consequently, we relax the constraints on the target type of a partial spine by admitting compound types in addition to base types. The typing semantics of partial spines and partial roots is formalized by means of the judgments

$$\Gamma;\Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} B > A \qquad\qquad \text{and} \qquad\qquad \Gamma;\Delta \vdash_\Sigma H \cdot \widetilde{S} \mathrel{\tilde{:}} A,$$

respectively. Notice again that the type A is arbitrary while it is bound to be a base type in the corresponding $S^{\rightarrow\multimap\&\top}$ relations. The definition of these judgments is displayed in Figure 6. It parallels the rules for spines and roots given in Figure 2. The base case in rule **pS_nil** handles the end of spine marker. It differs from the treatment of NIL in rule **lS_nil** by lifting the commitment to base types.

Observe that the definition of typing for partial spines and partial roots accesses the term typing judgment of $S^{\rightarrow\multimap\&\top}$ in rules **pS_redex**, **pS_lapp** and **pS_iapp**. This means in particular that roots possibly occurring in the arguments of a partial root or spine must have base type. Therefore, the deviation to the typing policy of $S^{\rightarrow\multimap\&\top}$ permitted by partiality is confined to the most shallow layer of terms.

We rely on partial spines and derived notions as a means to denote and manipulate $S^{\rightarrow\multimap\&\top}$ terms that are not in η-long form. Notice indeed that there is a derivation of the judgment

$$x : a \,\&\, (a \multimap a); \cdot \vdash_\Sigma x \cdot \text{NIL} \mathrel{\tilde{:}} a \,\&\, (a \multimap a)$$

since NIL is a valid partial spine of type $a \,\&\, (a \multimap a)$. Instead, the corresponding $S^{\rightarrow\multimap\&\top}$ judgment is not derivable because $x \cdot \text{NIL}$ is not in η-long form (i.e. it is not of base type). Instead, replacing this term with its η-expansion, $\langle x \cdot \pi_1 \,\text{NIL}, \hat{\lambda} y{:}a.\, x \cdot (y \cdot \text{NIL}) \,\hat{;}\, \pi_2 \,\text{NIL}\rangle$, (written $\langle \text{FST}\, x, \hat{\lambda} y{:}a.\, (\text{SND}\, x)\,\hat{}\,y\rangle$ in $\lambda^{\rightarrow\multimap\&\top}$) yields a derivable $S^{\rightarrow\multimap\&\top}$ judgment. Not every typable term that fails to be η-long is expressible in our extended language, but sufficiently many are in order to achieve the η-expansion of variables (in particular, our definition requires partial spine arguments to be η-long).

Partial root and spine typing is a conservative extension of the typing semantics of $S^{\rightarrow\multimap\&\top}$ roots and spines. Indeed, any well-typed root (spine) in $S^{\rightarrow\multimap\&\top}$ admits an isomorphic derivation according to the rules in Figure 6, and conversely every partial root of base type (partial spine of base target type) is typable according to the typing semantics presented in Section 2.2. This intuition is formally captured by the following lemma.

Lemma 2.20 (*Partial typing conservatively extends typing*)

i. $\Gamma;\Delta \vdash_\Sigma H \cdot S \mathrel{\tilde{:}} a$ *if and only if* $\Gamma;\Delta \vdash_\Sigma H \cdot S : a >;$

ii. $\Gamma;\Delta \vdash_\Sigma S \mathrel{\tilde{:}} B > a$ *if and only if* $\Gamma;\Delta \vdash_\Sigma S : B > a.$

Proof.

Each direction of the proof proceeds by mutual induction on the given derivations. ☑

This lemma implies that $S^{\rightarrow\multimap\&\top}$ roots and spines are semantically (and of course syntactically) special cases of the partial roots and spines we just defined. Therefore, in most results below, a spine (root) can be supplied whenever a partial spine (root) is expected. We will take advantage of this possibility in the sequel.

Given a partial spine $\widetilde{S}$, the *concatenation* of $\widetilde{S}$ with another partial spine $\widetilde{S}'$, denoted $\widetilde{S} \,@\, \widetilde{S}'$, constructs the partial spine $\widetilde{S}''$ obtained by replacing the trailing NIL of $\widetilde{S}$ with $\widetilde{S}'$. A formal definition is given as follows.

$$\begin{array}{rcl}
\text{NIL} \,@\, \widetilde{S}' & = & \widetilde{S}' \\
(\pi_1 \,\widetilde{S}) \,@\, \widetilde{S}' & = & \pi_1 \,(\widetilde{S} \,@\, \widetilde{S}') \\
(\pi_2 \,\widetilde{S}) \,@\, \widetilde{S}' & = & \pi_2 \,(\widetilde{S} \,@\, \widetilde{S}') \\
(V \,\hat{;}\, \widetilde{S}) \,@\, \widetilde{S}' & = & V \,\hat{;}\, (\widetilde{S} \,@\, \widetilde{S}') \\
(V ; \widetilde{S}) \,@\, \widetilde{S}' & = & V ; (\widetilde{S} \,@\, \widetilde{S}')
\end{array}$$

It is easy to ascertain that @ is a total function of its two arguments. Concatenation is associative, as expressed in the following lemma.

Lemma 2.21 (*Associativity of partial spine concatenation*)

$(\widetilde{S} \mathbin{@} \widetilde{S}') \mathbin{@} \widetilde{S}'' = \widetilde{S} \mathbin{@} (\widetilde{S}' \mathbin{@} \widetilde{S}'')$

Proof.

This statement is proved by induction on the structure of the partial spine $\widetilde{S}$. ☑

Many properties of the typing judgments of $S^{\rightarrow\multimap\&\top}$ extend to the current setting. In particular, weakening and promotion apply to partial spines and partial roots. We do not show their updated statement for the sake of economy, but we will rely on them in the sequel. In addition to these properties, the following lemma gives the typing properties of the concatenation operation. It will play a key role in the proofs below.

Lemma 2.22 (*Transitivity of partial spine typing*)

If $\widetilde{\mathcal{S}} :: \Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} A > A'$ *and* $\Gamma; \Delta' \vdash_\Sigma \widetilde{S}' \mathrel{\tilde{:}} A' > B$, *then* $\Gamma; \Delta, \Delta' \vdash_\Sigma \widetilde{S} \mathbin{@} \widetilde{S}' \mathrel{\tilde{:}} A > B$.

Proof.

This proof proceeds by a simple induction on the structure of $\widetilde{\mathcal{S}}$. ☑

By inspection of the typing rules of $S^{\rightarrow\multimap\&\top}$, it is easy to observe that no valid spine can have a source type of the form $\top$, or $a \multimap \top$, or more in general any type which result type is $\top$ (i.e. $\top$ or a type containing a positive occurrence of $\top$ as the right hand-side of linear or intuitionistic implication). No such restriction applies to partial spines because of the generality of rule **pS_nil**; for example, this rule alone constitutes a derivation of the judgment $\Gamma; \cdot \vdash_\Sigma \text{NIL} \mathrel{\tilde{:}} \top > \top$. This indicates that partial spines are a more general approximation of the notion of spines than we actually need. However their application in η-expansion does not make use of their full generality when dealing with types having $\top$ as their result type.

The reduction semantics of $S^{\rightarrow\multimap\&\top}$ extends without changes to partial roots and partial spines. In particular, we will make heavy use of weak head-reduction on partial roots. We adopt the notation already defined for $S^{\rightarrow\multimap\&\top}$. Many reduction properties of $S^{\rightarrow\multimap\&\top}$ apply naturally to our extended setting. The most important for our purposes is subject reduction and the substitution lemma. We will also take repeated advantage of the statement below, that describes the interaction between concatenation and the reduction of partial roots.

Lemma 2.23 (*Concatenation*)

Let $\mathcal{R}$ *be a derivation of* $H_1 \cdot \widetilde{S}_1 \xrightarrow{hr}{}^* H_2 \cdot \widetilde{S}_2$. *Then,* $H_1 \cdot (\widetilde{S}_1 \mathbin{@} \widetilde{S}) \xrightarrow{hr}{}^* H_2 \cdot (\widetilde{S}_2 \mathbin{@} \widetilde{S})$ *is derivable.*

Proof.

By induction on the structure of $\mathcal{R}$. ☑

Our η-expansion procedure is formalized by means of the judgment

$$x \xrightarrow{A} \widetilde{S} \rhd U$$

which is defined in Figure 7 by induction on the type A. In this judgment, x is the variable to be η-expanded, A is initially set to its type and then to subexpressions of this type, and the partial spine $\widetilde{S}$ serves as an accumulator for the spine S to which x should be applied. The term U corresponds to intermediate stages of the construction of the η-expansion of x. We will see that, given a variable x and a type A, there is always a term U such that the judgment

$$x \xrightarrow{A} \text{NIL} \rhd U$$

$$\frac{}{x \xrightarrow{a} \widetilde{S} \rhd x \cdot (\widetilde{S} \mathbin{@} \text{NIL})}\ \textbf{Sexp\_root}$$

$$\frac{}{x \xrightarrow{\top} \widetilde{S} \rhd \langle\rangle}\ \textbf{Sexp\_unit} \qquad \frac{x \xrightarrow{A_1} \widetilde{S} \mathbin{@} (\pi_1\, \text{NIL}) \rhd U_1 \quad x \xrightarrow{A_2} \widetilde{S} \mathbin{@} (\pi_2\, \text{NIL}) \rhd U_2}{x \xrightarrow{A_1 \,\&\, A_2} \widetilde{S} \rhd \langle U_1, U_2\rangle}\ \textbf{Sexp\_pair}$$

$$\frac{y \xrightarrow{A} \text{NIL} \rhd V \quad x \xrightarrow{B} \widetilde{S} \mathbin{@} (V \,\hat{;}\, \text{NIL}) \rhd U}{x \xrightarrow{A \multimap B} \widetilde{S} \rhd \hat{\lambda} y : A. U}\ \textbf{Sexp\_llam} \qquad \frac{y \xrightarrow{A} \text{NIL} \rhd V \quad x \xrightarrow{B} \widetilde{S} \mathbin{@} (V ; \text{NIL}) \rhd U}{x \xrightarrow{A \to B} \widetilde{S} \rhd \lambda y : A. U}\ \textbf{Sexp\_llam}$$

Figure 7: Variable η-Expansion in $S^{\to\multimap\&\top}$

is derivable, and this term is precisely the η-expansion of x at type A.

We start by proving that the judgment $x \xrightarrow{A} \widetilde{S} \rhd U$ as defined in Figure 7 is a total function of the variable x, the type A and the partial spine $\widetilde{S}$.

Lemma 2.24 (*Functionality of η-expansion*)

For every variable x, type A and partial spine $\widetilde{S}$, there is a unique term U such that the judgment

$$x \xrightarrow{A} \widetilde{S} \rhd U$$

is derivable.

Proof.

The proof proceeds by an easy induction on the structure of A. ☑

We would like now to show that what this procedure computes, when given a variable x, a type A and the end of spine NIL, is the η-expansion of x at type A. For our purposes, it will be sufficient to show that the object U it outputs has type A in a context consisting solely of $x : A$. In order to prove this property, we need to generalize it to consider intermediate stages of the construction of U. We have the following lemma.

Lemma 2.25 (*Well-typedness of η-expansion*)

Assume that there is a derivation $\mathcal{H}$ of the judgment $x \xrightarrow{A} \widetilde{S} \rhd U$. Then for all contexts Γ and Δ and type B such that the judgment $\Gamma; \Delta \vdash_\Sigma \widetilde{S} \,\hat{:}\, B > A$ is derivable, there is a derivation of $\Gamma; \Delta, x : B \vdash_\Sigma U : A$.

Proof.

This proof proceeds by induction on the structure of $\mathcal{H}$ or equivalently on the type A. We give the details of the most significant cases.

***A* = *a*:** Then

$$\mathcal{H} = \frac{}{x \xrightarrow{a} \widetilde{S} \rhd x \cdot (\widetilde{S} \mathbin{@} \text{NIL})}\ \textbf{Sexp\_root}$$

with $U = x \cdot (\widetilde{S} \mathbin{@} \text{NIL})$.

Assume there is a derivation of $\Gamma; \Delta \vdash_\Sigma \widetilde{S} \,\hat{:}\, B > a$. By rule **lS_nil**, the judgment $\Gamma; \cdot \vdash_\Sigma \text{NIL} : a > a$ is derivable. Therefore, by the transitivity lemma 2.22 there exists a derivation of

$$\Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathbin{@} \text{NIL} : B > a.$$

Then, it suffices to apply rule **lS_lvar** to obtain the desired derivation of

$$\Gamma; \Delta, x : B \vdash_\Sigma x \cdot (\widetilde{S} \mathbin{@} \text{NIL}) : B.$$

$\boldsymbol{A = \top}$**:** Then,

$$\mathcal{H} = \frac{}{x \xrightarrow{\top} \widetilde{S} \rhd \langle\rangle} \textbf{Sexp\_unit}$$

with $U = \langle\rangle$.

Rule **lS_unit** constitutes a derivation of $\Gamma; \Delta \vdash_\Sigma \langle\rangle : \top$ for any contexts Γ and Δ. In particular, this result holds for contexts Γ and $\Delta = \Delta', x : B$ such that $\Gamma; \Delta' \vdash_\Sigma \widetilde{S} \mathrel{\bar{:}} B > \top$.

$\boldsymbol{A = A_1 \,\&\, A_2}$**:** Then

$$\mathcal{H} = \frac{\begin{matrix}\mathcal{H}_1 & & \mathcal{H}_2 \\ x \xrightarrow{A_1} \widetilde{S} \mathbin{@} (\pi_1 \,\mathrm{NIL}) \rhd U_1 & & x \xrightarrow{A_2} \widetilde{S} \mathbin{@} (\pi_2 \,\mathrm{NIL}) \rhd U_2\end{matrix}}{x \xrightarrow{A_1 \,\&\, A_2} \widetilde{S} \rhd \langle U_1, U_2 \rangle} \textbf{Sexp\_pair}$$

with $U = \langle U_1, U_2 \rangle$.

Assume that $\widetilde{\mathcal{S}} :: \Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathrel{\bar{:}} B > A_1 \,\&\, A_2$. By chaining rule **pS_nil** with **pS_fst** and **pS_snd**, we can achieve derivations $\widetilde{\mathcal{S}}_i$ of $\Gamma; \cdot \vdash_\Sigma \pi_i \,\mathrm{NIL} \mathrel{\bar{:}} A_1 \,\&\, A_2 > A_i$, for $i = 1, 2$. By the transitivity lemma on $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{S}}_i$, we obtain derivations $\widetilde{\mathcal{S}}'_i$ of

$$\Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathbin{@} (\pi_i \,\mathrm{NIL}) \mathrel{\bar{:}} B > A_i.$$

By two applications of the induction hypothesis to $\mathcal{H}_i$ and $\widetilde{\mathcal{S}}'_i$, there are derivations of $\Gamma; \Delta, x : B \vdash_\Sigma U_i : A_i$. Using rule **lS_pair** yields the desired derivation of

$$\Gamma; \Delta, x : B \vdash_\Sigma \langle U_1, U_2 \rangle : A_1 \,\&\, A_2.$$

$\boldsymbol{A = A_1 \multimap A_2}$**:** Then,

$$\mathcal{H} = \frac{\begin{matrix}\mathcal{H}_1 & & \mathcal{H}_2 \\ y \xrightarrow{A_1} \mathrm{NIL} \rhd V' & & x \xrightarrow{A_2} \widetilde{S} \mathbin{@} (V' \mathbin{\hat{;}} \mathrm{NIL}) \rhd V\end{matrix}}{x \xrightarrow{A_1 \multimap A_2} \widetilde{S} \rhd \hat{\lambda} y : A_1 .\, V} \textbf{Sexp\_llam}$$

with $U = \hat{\lambda} y : A_1 .\, V$.

By induction hypothesis on $\mathcal{H}_1$, for every Γ, Δ and B such that $\widetilde{\mathcal{S}}_{\widetilde{\mathrm{NIL}}} :: \Gamma; \Delta \vdash_\Sigma \mathrm{NIL} \mathrel{\bar{:}} B > A_1$, there is a derivation of $\Gamma; \Delta, y : B \vdash_\Sigma V' : A_1$. Notice however that $\widetilde{\mathcal{S}}_{\widetilde{\mathrm{NIL}}}$ can only result from the application of rule **pS_nil**, forcing $\Delta = \cdot$ and $B = A_1$. Therefore, we have that for every context Γ, there is a derivation $\mathcal{U}_y$ of $\Gamma; y : A_1 \vdash_\Sigma V' : A_1$.

By concatenating rules **pS_nil** and **pS_lapp** relative to $\mathcal{U}_y$, we produce a derivation of the judgment $\Gamma; y : A_1 \vdash_\Sigma V' \mathbin{\hat{;}} \mathrm{NIL} \mathrel{\bar{:}} A_1 \multimap A_2 > A_2$. Assume we are given a derivation $\widetilde{\mathcal{S}}$ of $\Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathrel{\bar{:}} B > A_1 \multimap A_2$. Then, an application of the transitivity lemma yields a derivation $\widetilde{\mathcal{S}}'$ of

$$\Gamma; \Delta, y : A_1 \vdash_\Sigma \widetilde{S} \mathbin{@} (V' \mathbin{\hat{;}} \mathrm{NIL}) \mathrel{\bar{:}} B > A_2.$$

Therefore, by induction hypothesis on $\mathcal{H}_2$ and $\widetilde{\mathcal{S}}'$, the judgment

$$\Gamma; \Delta, y : A_1, x : B \vdash_\Sigma V : A_2$$

is derivable. Application of rule **lS_llam** yields the desired derivation of

$$\Gamma; \Delta, x : B \vdash_\Sigma \hat{\lambda} y : A_1 .\, V : A_1 \multimap A_2.$$

$\boldsymbol{A = A_1 \to A_2}$**:** The proof proceeds similarly to the previous case, except for the need to use the promotion lemma 2.2. ☑

A stronger version of this property holds when the result type of A is $\top$, as can be observed from the way we handled the case where $A = \top$. Indeed, given an η-expansion derivation $\mathcal{H} :: x \xrightarrow{A} \widetilde{S} \rhd U$, it is easy to show that, in this specific situation, for every contexts Γ and Δ, there is a derivation of $\Gamma; \Delta \vdash_\Sigma U : A$ (the assumption $x : B$ is not needed). We will not need to take advantage of this specialized property.

The above lemma specializes to the following corollary when we are in the initial configuration.

Corollary 2.26 (*Well-typedness of η-expansion*)

If $\mathcal{H} :: x \xrightarrow{A} \text{NIL} \rhd U$, *then* $\cdot; x\!:\!A \vdash_\Sigma U : A$ *and* $x\!:\!A; \cdot \vdash_\Sigma U : A$.

Proof.

By the above lemma, for all Γ, Δ and B such that $\widetilde{\mathcal{S}} :: \Gamma; \Delta \vdash_\Sigma \text{NIL} \mathrel{\tilde{:}} B > A$, there is a derivation of $\Gamma; \Delta, x\!:\!B \vdash_\Sigma U : A$. Notice however that $\widetilde{\mathcal{S}}$ can only result from the application of rule **pS_nil**, forcing $\Delta = \cdot$ and $B = A$. By further choosing $\Gamma = \cdot$, we obtain the desired derivation of $\cdot; x\!:\!A \vdash_\Sigma U : A$.

A derivation of $x\!:\!A; \cdot \vdash_\Sigma U : A$ is then obtained by appealing to the promotion lemma 2.2. ☑

We will call the unique object U such that $x \xrightarrow{A} \text{NIL} \rhd U$ is derivable the *η-expansion* of variable x at type A and denote it as x_η^A.

We conclude this section with a technical property concerning the reduction of η-expanded variables. As for the previous results, we will need only a very specific instance of a more general lemma. However, we shall state it in its full generality in order to be able to prove it. In this statement, we make use of our extension of the notion of reduction to partial roots.

Lemma 2.27 (*Reduction of η-expanded variables*)

i. If $\mathcal{H} :: x \xrightarrow{A} \widetilde{S} \rhd U$ *and* $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$, *then* $U \cdot S \longrightarrow^* x \cdot (\widetilde{S} \mathbin{@} S)$.

ii. If $\mathcal{H} :: x \xrightarrow{A} \widetilde{S} \rhd U$, $\widetilde{\mathcal{S}} :: \Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} B > A$, *$x$ does not occur free in* $\widetilde{S}$, $\mathcal{U} :: \Gamma; \Delta' \vdash_\Sigma V : B$ *and* $V \cdot \widetilde{S} \xrightarrow{hr}{}^* U^* \cdot \text{NIL}$, *then* $[V/x]U \longrightarrow^* U^*$.

iii. If $\mathcal{U} :: \Gamma, z\!:\!B; \Delta \vdash_\Sigma U : A$, *then* $[x_\eta^B/z]U \longrightarrow^* [x/z]U$.

iv. If $\mathcal{S} :: \Gamma, z\!:\!B; \Delta \vdash_\Sigma S : A > a$, *then* $[x_\eta^B/z]S \longrightarrow^* [x/z]S$.

Proof.

This rather involved proof proceeds by simultaneous induction over the structure of S in (i), of U^* in (ii), of U in (iii), and of S in (iv). More precisely, we admit appealing to the induction hypothesis in the following circumstances:

- Given a spine S in (i), we will induce on (i) for spines S' smaller than S, and on (ii) for terms U^* contained in S.
- Given a term U^* in (ii), we will apply the induction hypothesis (ii) to term U^{**} that differ from a subterm U'' of U^* only by the renaming of a free variable (if $U^{**} = [x/z]U''$ for example). We will also appeal to (iii) on terms U smaller than U^*.
- Given a term U in (iii), we will induce on (iii) for subterms of U, and on (i) and (iv) for spines S embedded in U.
- Finally, given a spine S in (iv), we will access either (iv) on spines S' smaller than S, or (iii) on subterms U of S.

We will now outline the development of a number of significant cases. We distinguish cases on the basis of the type A appearing in the various parts of this lemma.

(i) $A = a$: By inversion on $\mathcal{S}$ and $\mathcal{H}$, it must be the case that $S = \textsc{nil}$ and $U = x \cdot (\widetilde{S} \mathbin{@} \textsc{nil})$. Then, by rule **Sr_nil**,

$$U \cdot S = (x \cdot (\widetilde{S} \mathbin{@} \textsc{nil})) \cdot \textsc{nil} \longrightarrow x \cdot (\widetilde{S} \mathbin{@} \textsc{nil}) = x \cdot (\widetilde{S} \mathbin{@} S)$$

(i) $A = \top$: By inversion on $\mathcal{S}$, this case cannot arise.

(i) $A = A_1 \mathbin{\&} A_2$: By inversion on $\mathcal{H}$, we deduce that $U = \langle U_1, U_2 \rangle$ and that there are derivations of $x \xrightarrow{A_i} \widetilde{S} \mathbin{@} (\pi_i\, \textsc{nil}) \rhd U_i$ for $i = 1, 2$. Furthermore, inversion on $\mathcal{S}$ opens two alternative courses:

- $S = \pi_1\, S_1$ and $\mathcal{S}_1 :: \Gamma; \Delta \vdash_\Sigma S_1 : A_1 > a$. By induction hypothesis, the associativity of concatenation and the definition of this operation,

  $$U_1 \cdot S_1 \longrightarrow^* x \cdot ((\widetilde{S} \mathbin{@} \pi_1\, \textsc{nil}) \mathbin{@} S_1) = x \cdot (\widetilde{S} \mathbin{@} (\pi_1\, \textsc{nil} \mathbin{@} S_1)) = x \cdot (\widetilde{S} \mathbin{@} \pi_1\, S_1)$$

  Now, by rule **Sr_beta_fst**,

  $$\langle U_1, U_2 \rangle \cdot \pi_1\, S_1 \longrightarrow U_1 \cdot S_1 \longrightarrow^* x \cdot (\widetilde{S} \mathbin{@} \pi_1\, S_1).$$

- $S = \pi_1\, S_2$ and $\mathcal{S}_2 :: \Gamma; \Delta \vdash_\Sigma S_2 : A_2 > a$. We proceed symmetrically to the previous subcase.

(i) $A = A_1 \multimap A_2$: By inversion on $\mathcal{H}$, $U = \hat{\lambda} y{:}A_1.\, U'$ and there are derivations of

$$x \xrightarrow{A_1} \textsc{nil} \rhd y_\eta^{A_1} \qquad \text{and} \qquad x \xrightarrow{A_2} \widetilde{S} \mathbin{@} (y_\eta^{A_1} \mathbin{\hat{;}} \textsc{nil}) \rhd U'.$$

Since y is bound in U, we can assume it occurs neither in $\widetilde{S}$ nor in S. By further inversion on $\mathcal{S}$, we obtain that $S = V \mathbin{\hat{;}} S'$, $\Delta = \Delta', \Delta''$ and there exist derivations of

$$\Gamma; \Delta'' \vdash_\Sigma V : A_1 \qquad \text{and} \qquad \Gamma; \Delta' \vdash_\Sigma S' : A_2 > a.$$

The induction hypothesis and the associativity of @ permits concluding

$$U' \cdot S' \longrightarrow^* x \cdot ((\widetilde{S} \mathbin{@} y_\eta^{A_1} \mathbin{\hat{;}} \textsc{nil}) \mathbin{@} S') = x \cdot (\widetilde{S} \mathbin{@} y_\eta^{A_1} \mathbin{\hat{;}} S').$$

We then conclude this case of the proof as follows:

$$\begin{array}{rcll}
(\hat{\lambda} y{:}A_1.\, U') \cdot (V \mathbin{\hat{;}} S') & \longrightarrow & [V/y]U' \cdot S' & \text{by rule } \mathbf{Sr_beta_lin}, \\
& = & [V/y](U' \cdot S') & \text{since } y \text{ does not occur free in } S', \\
& \longrightarrow^* & [V/y](x \cdot (\widetilde{S} \mathbin{@} y_\eta^{A_1} \mathbin{\hat{;}} S')) & \text{by the substitution lemma 2.4 and} \\
& & & \text{the above induction hypothesis,} \\
& = & x \cdot (\widetilde{S} \mathbin{@} ([V/y]y_\eta^{A_1} \mathbin{\hat{;}} S')) & \text{since } y \text{ is not free in } x,\ S' \text{ and } \widetilde{S}, \\
& = & [[V/y]y_\eta^{A_1}/z](x \cdot (\widetilde{S} \mathbin{@} z \mathbin{\hat{;}} S')) & \text{for some new variable } z, \\
& \longrightarrow^* & [V/z](x \cdot (\widetilde{S} \mathbin{@} z \mathbin{\hat{;}} S')) & \text{by induction hypothesis } (ii) \text{ and} \\
& & & \text{the substitution lemma,} \\
& = & x \cdot (\widetilde{S} \mathbin{@} V \mathbin{\hat{;}} S') & \text{by definition of substitution.}
\end{array}$$

(i) $A = A_1 \to A_2$: We proceed similarly.

(ii) $A = a$: By inversion on $\mathcal{H}$, we have that $U = x \cdot (\widetilde{S} \mathbin{@} \textsc{nil})$. By applying the transitivity lemma on rule **lS_nil** and $\widetilde{\mathcal{S}}$, we obtain a derivation $\mathcal{S}$ of $\Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathbin{@} \textsc{nil} : B > a$.

Since $V \cdot \widetilde{S} \xrightarrow{hr}{}^* U^* \cdot \textsc{nil}$ holds by assumption, the concatenation lemma allows us to conclude that

$$V \cdot (\widetilde{S} \mathbin{@} \textsc{nil}) \longrightarrow^* U^* \cdot (\textsc{nil} \mathbin{@} \textsc{nil}) = U^* \cdot \textsc{nil}.$$

Thus, since x does not occur free in $\widetilde{S}$, we have that

$$[V/x](x \cdot (\widetilde{S} \mathbin{@} \textsc{nil})) = V \cdot (\widetilde{S} \mathbin{@} \textsc{nil}) \longrightarrow^* U^* \cdot \textsc{nil}.$$

Finally, by rule **lS_redex** on $\mathcal{U}$ and $\mathcal{S}$, there is a derivation of $\Gamma;\Delta,\Delta' \vdash_\Sigma V \cdot (\widetilde{S} \,@\, \mathrm{NIL}) : a$, so that, by subject reduction, also $\Gamma;\Delta,\Delta' \vdash_\Sigma U^* \cdot \mathrm{NIL} : a$ is derivable, and therefore by inversion on rule **lS_redex**, there is derivation of $\Gamma;\Delta,\Delta' \vdash_\Sigma U^* : a$ as well. Again by inversion, U^* must be a root. Therefore we can apply rule **Sr_nil**, obtaining that $U^* \cdot \mathrm{NIL} \longrightarrow U^*$. By chaining this reduction with the previous ones, we get the requested derivation of the judgment

$$[V/x](x \cdot (\widetilde{S} \,@\, \mathrm{NIL})) \longrightarrow^* U^*.$$

(ii) $\boldsymbol{A = \top}$**:** By inversion on $\mathcal{H}$, we deduce that $U = \langle\rangle$. By rule **pS_redex**, $\Gamma;\Delta,\Delta' \vdash_\Sigma V \cdot \widetilde{S} \mathrel{\bar{:}} \top$ is derivable and therefore, by subject reduction, there is a derivation of $\Gamma;\Delta,\Delta' \vdash_\Sigma U^* \cdot \mathrm{NIL} \mathrel{\bar{:}} \top$, from which we deduce, by inversion, that $\Gamma;\Delta,\Delta' \vdash_\Sigma U^* : \top$, and therefore, again by inversion, that $U^* = \langle\rangle$. Then, trivially $[V/x]\langle\rangle = \langle\rangle$.

Observe that the treatment of this case relies on the existence of a derivation for $\Gamma;\cdot \vdash_\Sigma \mathrm{NIL} \mathrel{\bar{:}} \top > \top$, that is readily produced by means of rule **pS_nil**. As we said, concatenating NIL with no $S^{\rightarrow\multimap\&\top}$ object can yields a well-typed spine.

(ii) $\boldsymbol{A = A_1 \,\&\, A_2}$**:** By inversion on $\mathcal{H}$, we have that $U = \langle U_1, U_2\rangle$ and that there are derivations of $x \xrightarrow{A_i} \widetilde{S} \,@\, (\pi_i\,\mathrm{NIL}) \rhd U_i$ for $i = 1,2$. By rules **pS_fst** and **pS_snd** and the transitivity lemma on $\widetilde{\mathcal{S}}$, we can produce derivations of

$$\Gamma;\Delta \vdash_\Sigma \widetilde{S} \,@\, (\pi_i\,\mathrm{NIL}) : B > A_i.$$

By knowing that $V \cdot \widetilde{S} \xrightarrow{hr}{}^* U^* \cdot \mathrm{NIL}$, we deduce by the concatenation lemma that

$$V \cdot (\widetilde{S} \,@\, \pi_i\,\mathrm{NIL}) \longrightarrow^* U^* \cdot \pi_i\,\mathrm{NIL}.$$

Similarly to the previous case, appeals to rule **pS_redex**, to the transitivity lemma 2.22 and to inversion permit us to deduce that there is a derivation of $\Gamma;\Delta,\Delta' \vdash_\Sigma U^* : A_1 \,\&\, A_2$ and thus that $U^* = \langle U_1^*, U_2^*\rangle$ and, once more by inversion, that $\Gamma;\Delta,\Delta' \vdash_\Sigma U_i^* : A_i$. By chaining rules **Sr_beta_fst** and **Sr_beta_snd** to the reduction sequence above, we obtain that, for $i = 1,2$,

$$V \cdot (\widetilde{S} \,@\, \pi_i\,\mathrm{NIL}) \xrightarrow{hr}{}^* U_i^* \cdot \mathrm{NIL}.$$

We are now in a position of appealing to the induction hypothesis, obtaining derivations for the reduction judgments $[V/x]U_i \longrightarrow^* U_i^*$ from which we easily achieve the desired derivation of

$$[V/x]\langle U_1, U_2\rangle \longrightarrow^* \langle U_1^*, U_2^*\rangle$$

by rules **Sr_pair1**, **Sr_pair2**, the definition of substitution and transitivity at the level of reductions.

(ii) $\boldsymbol{A = A_1 \multimap A_2}$**:** By inversion on $\mathcal{H}$, we know that $U = \hat{\lambda} y : A_1.\, U'$ and that there are derivations $\mathcal{H}_1$ and $\mathcal{H}_2$ of

$$x \xrightarrow{A_1} \mathrm{NIL} \rhd y_\eta^{A_1} \qquad \text{and} \qquad x \xrightarrow{A_2} \widetilde{S} \,@\, (y_\eta^{A_1} \,\hat{;}\, \mathrm{NIL}) \rhd U',$$

respectively. By the above corollary 2.26 and weakening, there is a derivation of

$$\Gamma; y : A_1 \vdash_\Sigma y_\eta^{A_1} : A_1.$$

An application of rules **pS_nil** and **pS_lapp** yields a derivation of $\Gamma; y : A_1 \vdash_\Sigma y_\eta^{A_1} \,\hat{;}\, \mathrm{NIL} \mathrel{\bar{:}} A_1 \multimap A_2 > A_2$. This derivation and $\widetilde{\mathcal{S}}$ can be combined by means of the transitivity lemma 2.22 into a derivation of

$$\Gamma;\Delta, y : A_1 \vdash_\Sigma \widetilde{S} \,@\, (y_\eta^{A_1} \,\hat{;}\, \mathrm{NIL}) \mathrel{\bar{:}} B > A_2.$$

By rule **pS_redex**, subject reduction and inversion, we deduce that $U^* = \hat{\lambda} z : A_1.\, U^{**}$ and $\Gamma;\Delta,\Delta', z : A_1 \vdash_\Sigma U^{**} : A_2$ is derivable. By the promotion lemma, there is also a derivation of

$$\Gamma, z : A_1; \Delta,\Delta' \vdash_\Sigma U^{**} : A_2.$$

On the basis of these facts, we can now construct the following sequence of reductions:

$V \cdot (\widetilde{S} \mathbin{@} y_\eta^{A_1} \mathbin{\hat{;}} \mathrm{NIL})$	$\xrightarrow{hr}{}^*$	$(\hat{\lambda} z : A_1 .\, U^{**}) \cdot (\mathrm{NIL} \mathbin{@} y_\eta^{A_1} \mathbin{\hat{;}} \mathrm{NIL})$	by the concatenation lemma on the assumption $V \cdot \widetilde{S} \xrightarrow{hr}{}^* U^* \cdot \mathrm{NIL}$,
	$=$	$(\hat{\lambda} z : A_1 .\, U^{**}) \cdot (y_\eta^{A_1} \mathbin{\hat{;}} \mathrm{NIL})$	by definition of concatenation,
	$\xrightarrow{hr}$	$[y_\eta^{A_1}/z]U^{**} \cdot \mathrm{NIL}$	by rule **whr_beta_lapp**
	$\xrightarrow{hr}{}^*$	$[y/z]U^{**} \cdot \mathrm{NIL}$	by induction hypothesis (iii) and the substitution lemma.

We can now apply the induction hypothesis, obtaining a derivation of $[V/x]U' \longrightarrow^* [y/z]U^{**}$. Then, rule **Sr_llam** yields the desired result:

$$[V/x](\hat{\lambda} y : A_1 .\, U') \longrightarrow^* \hat{\lambda} y : A_1 .\, [y/z]U^{**} = \hat{\lambda} z : A_1 .\, U^{**},$$

where the last equality relies on our convention about implicit renaming of bound variables.

***(ii)* $A = A_1 \to A_2$:** We proceed as in the previous case, except that there is no need to appeal to the promotion lemma.

***(iii–iv)*:** Most of the cases falling into this category have a simple proof based on straightforward inversion and appeals to the induction hypothesis. We will concentrate on the case of (iii) where $A = a$, from which we deduce that $U = H \cdot S$ for some term H and spine S. We then need to proceed by considering the different alternatives for the head H. All these subcases are handled trivially except for the situation where H is the variable z.

Then, by inversion we know that there exist a derivation of $\mathcal{S} :: \Gamma, z : B; \Delta \vdash_\Sigma S : B > a$. By induction hypothesis (iv), we have therefore that

$$[x_\eta^B/z]S \longrightarrow^* [x/z]S.$$

From $\mathcal{S}$, it is a simple matter to prove that there is a derivation of $\Gamma, x : B; \Delta \vdash_\Sigma [x/z]S : B > a$. Since, by definition, $x \xrightarrow{B} \mathrm{NIL} \rhd x_\eta^B$, we can apply the induction hypothesis (i) obtaining that

$$x_\eta^B \cdot [x/z]S \longrightarrow^* x \cdot [x/z]S.$$

We can now chain these reductions as follows:

$$[x_\eta^B/z](z \cdot S) = x_\eta^B \cdot [x_\eta^B/z]S \longrightarrow^* x_\eta^B \cdot [x/z]S \longrightarrow^* x \cdot [x/z]S = [x/z](z \cdot S),$$

obtaining in this way the desired result. ☑

Below, we will only need a very special case of the above lemma, reported as the following corollary.

Corollary 2.28 (*Canonical reduction of η-expanded variables*)

If $\Gamma; \Delta \vdash_\Sigma S : A > a$, *then* $\mathrm{Can}(x_\eta^A \cdot S) = \mathrm{Can}(x \cdot S)$.

Proof.

By part (i) of the previous lemma and the definition of η-expansion x_η^A, we know that there is a derivation of $x_\eta^A \cdot S \longrightarrow^* x \cdot S$. We obtain the desired result by confluence (Lemma 2.5) and strong normalization (Theorem 2.6). ☑

Observe that this property fails as soon as we replace reduction to canonical form with weak head-normalization: the shallow reductions performed by the latter operation cope inadequately with the thorough transformation resulting from η-expansion. Indeed, it is not true in general that, if $\Gamma; \Delta \vdash_\Sigma$

$S : A > a$, then $\overline{x^A_\eta \cdot S} = \overline{x \cdot S}$. As a counterexample, assume the variable x has type $A = (a \to a) \to a$, so that

$$x^A_\eta = \lambda f : a \to a.\,(x \cdot ((\lambda y : a.\,(f \cdot (y; \text{NIL}))); \text{NIL})),$$

and S is the spine $(\lambda z : a.\,(z \cdot \text{NIL})); \text{NIL}$. Then, $\overline{x \cdot S} = x \cdot S$. Instead,

$$\overline{x^A_\eta \cdot S} = x \cdot ((\lambda y : a.\,((\lambda z : a.\,(z \cdot \text{NIL})) \cdot (y; \text{NIL}))); \text{NIL})).$$

A further step of β-reduction is needed to obtain $x \cdot S$ from this expression.

3 Linear Higher-Order Unification

In this section, we define the unification problem for $S^{\to\multimap\&\top}$ (Section 3.1), show a few examples (Section 3.2), describe a pre-unification algorithm à la Huet for it (Section 3.3), prove its soundness and completeness (Section 3.4), and discuss new sources of non-determinism introduced by linearity (Section 3.5).

3.1 The Unification Problem

Equality checking becomes a *unification problem* as soon as we admit objects containing *logical variables* (sometimes called *existential variables* or *meta-variables*), standing for unknown terms. The equalities above, called *equations* in this setting, are *unifiable* if there exists a substitution for the logical variables which makes the two sides equal, according to the definition given in the previous section. These substitutions are called *unifiers*. The task of a *unification procedure* is to determine whether equations are solvable and, if so, report their unifiers. As for $\lambda^{\to}$, it is undecidable whether two $S^{\to\multimap\&\top}$ terms can be unified, since the equational theory of $\lambda^{\to\multimap\&\top}$ is a conservative extension of the equational theory of the simply-typed λ-calculus.

An algorithm that returns a set of solvable residual equations, besides a substitution with the above properties, is called a *pre-unification procedure* [Hue75]. The idea behind this approach is to postpone some solvable equations (the so called *flex-flex equations*) as *constraints* instead of enumerating their solutions, as done by a unification algorithm. Pre-unification is undecidable in both $\lambda^{\to}$ and $\lambda^{\to\multimap\&\top}$ since it subsumes deciding whether a set of equations has a solution.

Logical variables stand for heads and cannot replace spines or generic terms. Therefore, the alterations to the definition of $S^{\to\multimap\&\top}$ required for unification are limited to enriching the syntax of heads with logical variables, that we denote F and G, possibly subscripted. We continue to write U, V and S for terms and spines in this extended language. In order to avoid confusion we will call the proper variables of $S^{\to\multimap\&\top}$ *parameters* in the remainder of the paper. The resulting extended syntax for $S^{\to\multimap\&\top}$ is formalized as follows:

$$\begin{array}{llll@{\qquad}llll@{\qquad}llll}
\textit{Terms:} & U & ::= & H \cdot S & \textit{Spines:} & S & ::= & \text{NIL} & \textit{Heads:} & H & ::= & c \quad (\textit{constants}) \\
 & & \mid & \lambda x : A.\,U & & & \mid & U; S & & & \mid & x \quad (\textit{parameters}) \\
 & & \mid & \hat{\lambda} x : A.\,U & & & \mid & U \mathbin{\hat{;}} S & & & \mid & U \quad (\textit{redices}) \\
 & & \mid & \langle U_1, U_2 \rangle & & & \mid & \pi_1\, S \mid \pi_2\, S & & & \mid & F \quad (\textit{logical variables}) \\
 & & \mid & \langle\rangle & & & & & & & &
\end{array}$$

The machinery required in order to state a unification problem is summarized in the grammar below. We will in general solve *systems* Ξ of equations that share the same signature Σ and a common set of logical variables Φ. A system can contain both *term equations* $\Gamma; \Delta \vdash U_1 = U_2 : A$ and *spine equations* $\Gamma; \Delta \vdash S_1 = S_2 : A > a$. A *solution* to a unification problem, also called a *pre-unifier*, is a substitution Θ that, when applied to Ξ, yields a *system of flex-flex equations* Ξ_{ff} that is known to be solvable. A flex-flex equation relates roots with logical variables as their heads. This notion of solution, characteristic of pre-unification, subsumes unifiers as the particular case in which the residual flex-flex system is empty.

Finally, we record the types of the logical variables in use in a *pool*.

$$
\begin{array}{rlll}
\textit{Equation systems:} & \Xi & ::= & \cdot \mid \Xi, (\Gamma; \Delta \vdash U_1 = U_2 : A) \mid \Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A > a) \\
\textit{Flex-flex systems:} & \Xi_{\mathit{ff}} & ::= & \cdot \mid \Xi_{\mathit{ff}}, (\Gamma; \Delta \vdash F_1 \cdot S_1 = F_2 \cdot S_2 : a) \\
\textit{Substitutions:} & \Theta & ::= & \cdot \mid \Theta, U/F \\
\textit{Pools:} & \Phi & ::= & \cdot \mid \Phi, F : A
\end{array}
$$

We assume that variables appear at most once in a pool and in the domain of a substitution. Similarly to contexts, we treat equation systems, substitutions and pools as multisets. We write ξ for individual equations. The context $\Gamma; \Delta$ in an equation ξ enumerates the parameters that the substitutions for logical variables appearing in ξ are not allowed to mention directly. Therefore, legal substitution terms U for a variable $F : A$ must be typable in the empty context, i.e. $\cdot; \cdot \vdash_{\Sigma,\Phi} U : A$ should be derivable where Φ includes the logical variables appearing in U (notice in particular that U is purely intuitionistic). This is sometimes emphasized by denoting an equation system Ξ as $\forall\Sigma.\, \exists\Phi.\, \forall(\Xi)$, where the inner expression means that the context $\Gamma; \Delta$ of every equation ξ is universally quantified in front of it.

A term or spine equation ξ can be interpreted as an equality judgment with signature (Σ, Φ), where again Φ includes the logical variables appearing in ξ. In the following, we will occasionally view an equation system Ξ as the multiset of the equality judgments corresponding to its equations. In these cases, we write $\vec{\mathcal{E}} :: \Xi$ to indicate that each equation ξ, seen as an equality judgment, in the system Ξ has a derivation $\mathcal{E}_\xi$. We treat $\vec{\mathcal{E}}$ as a multiset with elements the derivations $\mathcal{E}_\xi$. We call an equation *well-typed* if the corresponding equality judgment is well-typed. This notion extends naturally to equation systems.

The usual definitions concerning substitutions [Bar80] are trivially extended to our language. In particular, the *domain* of a substitution Θ, denoted $\mathrm{dom}(\Theta)$, is the multiset of variables F such that U/F occurs in Θ, its *image*, $\mathrm{Im}(\Theta)$, is the multiset of the corresponding terms U, and its *range*, written $\mathrm{rg}(\Theta)$, is the multiset of logical variables appearing in $\mathrm{Im}(\Theta)$. We will always assume the range of a substitution to be disjoint from its domain. The *application* of a substitution Θ to a term U (spine S) is denoted $[\Theta]U$ ($[\Theta]S$, respectively). We extend this notion to the application of a substitution Θ to another substitution Θ', written $[\Theta]\Theta'$ and defined as the substitution obtained by applying Θ to every term in the image of Θ'. We write $\Theta \circ \Theta'$ for the *composition* of substitutions Θ and Θ'. These operations retain their usual semantics [Bar80] also in our setting. We will take particular advantage of the following properties.

Property 3.1 (*Substitutions*)

i. $[\Theta \circ \Theta']U = [\Theta]([\Theta']U)$ *and similarly for spines;*

ii. $\Theta \circ \Theta' = \Theta, [\Theta]\Theta'$*;*

iii. (Associativity) $(\Theta \circ \Theta') \circ \Theta'' = \Theta \circ (\Theta' \circ \Theta'')$. □

A consequence of (*ii*) is that $[\Theta]\Theta' = (\Theta \circ \Theta')_{|\,\mathrm{dom}(\Theta')}$. We define the *canonical form* of a substitution Θ, written $\mathrm{Can}(\Theta)$, as the substitution that differs from Θ by replacing every element U/F in it with $\mathrm{Can}(U)/F$, where logical variables are treated as if they were constants.

Given a signature Σ, a substitution Θ and a pool Φ that assigns a type at least to every logical variable in the domain and range of Θ, we say that Θ is *well-typed* with respect to Σ and Φ if, whenever U/F occurs in Θ and $F : A$ appears in Φ, there is a derivation of $\cdot; \cdot \vdash_{\Sigma,\Phi} U : A$. Notice that the logical variables in $\mathrm{rg}(\Theta)$ are again treated as constants.

The above informal definitions will be sufficient to follow the development of the discussion. It is lengthy but not difficult to make them fully formal. We refrain from doing so in order not to blur the analysis of our unification algorithm with additional complexity.

3.2 Examples

The example given in the introduction clearly shows how linearity restricts the set of solutions found by traditional higher-order unification in the absence of linear constructs. We can indeed rewrite this example in the syntax of $\lambda^{\rightarrow\multimap\&\top}$ (chosen over $S^{\rightarrow\multimap\&\top}$ for the sake of clarity) as the following equation

$$\cdot;\cdot \vdash F\hat{\ }M = c\hat{\ }M\hat{\ }M : a.$$

where we assume M to be a closed term. As we saw, only two of the four independent solutions returned by traditional higher-order unification on the corresponding $\lambda^{\rightarrow}$ problem are linearly valid.

More complex situations rule out the simple-minded strategy of keeping only the linearly valid solutions returned by a traditional unification procedure on a linear problem. Consider the following equation, written again in the syntax of $\lambda^{\rightarrow\multimap\&\top}$ for simplicity,

$$x\!:\!a, y\!:\!a; \cdot \vdash F\hat{\ }x\hat{\ }y = c\hat{\ }(G_1\ x\ y)\hat{\ }(G_2\ x\ y) : a.$$

The parameters x and y are intuitionistic, but F uses them as linear objects. We must instantiate F to a term of the form $\hat{\lambda} x'\!:\!a.\, \hat{\lambda} y'\!:\!a.\, c\hat{\ }M_1\hat{\ }M_2$ where each of the *linear* parameters x' and y' must appear either in M_1 or in M_2, but not in both. Indeed, we have the following four incomparable substitutions:

$$\begin{array}{lll}
F \longleftarrow \hat{\lambda} x'\!:\!a.\, \hat{\lambda} y'\!:\!a.\, c\hat{\ }(F_1\hat{\ }x'\hat{\ }y')\hat{\ }F_2, & G_1 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_1\hat{\ }x'\hat{\ }y', & G_2 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_2. \\
F \longleftarrow \hat{\lambda} x'\!:\!a.\, \hat{\lambda} y'\!:\!a.\, c\hat{\ }(F_1\hat{\ }x')\hat{\ }(F_2\hat{\ }y'), & G_1 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_1\hat{\ }x', & G_2 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_2\hat{\ }y'. \\
F \longleftarrow \hat{\lambda} x'\!:\!a.\, \hat{\lambda} y'\!:\!a.\, c\hat{\ }(F_1\hat{\ }y')\hat{\ }(F_2\hat{\ }x'), & G_1 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_1\hat{\ }y', & G_2 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_2\hat{\ }x'. \\
F \longleftarrow \hat{\lambda} x'\!:\!a.\, \hat{\lambda} y'\!:\!a.\, c\hat{\ }F_1\hat{\ }(F_2\hat{\ }x'\hat{\ }y'), & G_1 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_1, & G_2 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_2\hat{\ }x'\hat{\ }y'.
\end{array}$$

Traditional unification on the analogous $\lambda^{\rightarrow}$ equation is unitary and has a single solution:

$$F \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, c\,(F_1\ x'\ y')\,(F_2\ x'\ y'),\ \ G_1 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_1\ x'\ y',\ \ G_2 \longleftarrow \lambda x'\!:\!a.\, \lambda y'\!:\!a.\, F_2\ x'\ y'.$$

which is *not* linearly valid. This example also illustrates one reason why linear term languages and unification are useful. Linearity constraints rule out certain unifiers when compared to the simply-typed formulation of the same expression, which can be used to eliminate ill-formed terms early.

3.3 A Pre-Unification Algorithm

Our adaptation of Huet's pre-unification procedure to $S^{\rightarrow\multimap\&\top}$ is summarized in Figures 8–10. We adopt a structured operational semantics presentation as a system of inference rules, which isolates and makes every step of the algorithm explicit. Although more verbose than the usual formulations, it is, at least in this setting, more understandable and closer to an actual implementation. In this subsection, we describe the general structure of the algorithm. We will prove its correctness in Section 3.4 and discuss the specific aspects brought in by linearity in Section 3.5.

On the basis of the above definitions, a unification problem is expressed by the following judgment:

$$\Xi \setminus \Xi_{f\!f}, \Theta$$

where, for the sake of readability, we keep the signature Σ and the current variable pool Φ implicit. We assume Ξ consists of well-typed equations. The procedure we describe accepts Σ, Φ and Ξ as input arguments and attempts to construct a derivation $\mathcal{X}$ of $\Xi \setminus \Xi_{f\!f}, \Theta$ for some Θ and $\Xi_{f\!f}$. This could terminate successfully (in which case Θ is a unifier if $\Xi_{f\!f}$ is empty, and only a pre-unifier otherwise). It might also fail (in which case there are no unifiers) or not terminate (in which case we have no information).

Given a system of well-typed equations Ξ to be solved with respect to a signature Σ and a logical variables pool Φ, the procedure non-deterministically selects an equation ξ from Ξ and attempts to apply in a bottom up fashion one of the rules in Figure 8. If several rules are applicable, the procedure succeeds if one of them yields a solution. If none applies, we have a local failure. The procedure terminates when all equations in Ξ are flex-flex, as described below.

Well-typed equations in η-long form have a very disciplined structure. In particular, both sides must either be roots, or have the same top-most term or spine constructor. Spine equations and non-atomic term equations are therefore decomposed until problems of base type are exposed, as shown in the

Term traversal

$$\frac{\Xi, (\Gamma; \Delta \vdash \overline{U \cdot S_1} = H \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash U \cdot S_1 = H \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_redex\_l} \qquad \frac{\Xi, (\Gamma; \Delta \vdash H \cdot S_1 = \overline{U \cdot S_2} : a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash H \cdot S_1 = U \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_redex\_r}$$

$$\frac{\Xi \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \langle\rangle = \langle\rangle : \top) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_unit} \qquad \frac{\Xi, (\Gamma; \Delta \vdash U_1 = V_1 : A), (\Gamma; \Delta \vdash U_2 = V_2 : B) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \langle U_1, U_2 \rangle = \langle V_1, V_2 \rangle : A \,\&\, B) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_pair}$$

$$\frac{\Xi, (\Gamma; \Delta, x{:}A \vdash U = V : B) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \hat{\lambda} x{:}A. U = \hat{\lambda} x{:}A. V : A \multimap B) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_llam} \qquad \frac{\Xi, (\Gamma, x{:}A; \Delta \vdash U = V : B) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \lambda x{:}A. U = \lambda x{:}A. V : A \to B) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_ilam}$$

Rigid–Rigid

$$\frac{c{:}A \ \mathit{in}\ \Sigma \quad \Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash c \cdot S_1 = c \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_rr\_con}$$

$$\frac{\Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta, x{:}A \vdash x \cdot S_1 = x \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_rr\_lvar} \qquad \frac{\Xi, (\Gamma, x{:}A; \Delta \vdash S_1 = S_2 : A > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma, x{:}A; \Delta \vdash x \cdot S_1 = x \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_rr\_ivar}$$

Rigid–Flex

$$\frac{F{:}A \ \mathit{in}\ \Phi \quad h{:}B \ \mathit{in}\ \Sigma, \Gamma \mathit{or}\ \Delta \quad \Xi, (\Gamma; \Delta \vdash F \cdot S_2 = h \cdot S_1 : a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash h \cdot S_1 = F \cdot S_2 : a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_rf}$$

Flex–Rigid

$$\frac{F{:}A \ \mathit{in}\ \Phi \quad \cdot;\cdot \vdash c \cdot S_2 \,/\, A \Uparrow^{\iota} S_1 \hookrightarrow V \quad [V/F](\Xi, (\Gamma; \Delta \vdash F \cdot S_1 = c \cdot S_2 : a)) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash F \cdot S_1 = c \cdot S_2 : a) \setminus \Xi_{ff}, (\Theta \circ V/F)}\ \textbf{pu\_fr\_imit}$$

$$\frac{F{:}A \ \mathit{in}\ \Phi \quad h{:}B \ \mathit{in}\ \Sigma, \Gamma \mathit{or}\ \Delta \quad \cdot;\cdot \vdash A \Uparrow^{\pi} S_1 \hookrightarrow V \quad [V/F](\Xi, (\Gamma; \Delta \vdash F \cdot S_1 = h \cdot S_2 : a)) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash F \cdot S_1 = h \cdot S_2 : a) \setminus \Xi_{ff}, (\Theta \circ V/F)}\ \textbf{pu\_fr\_proj}$$

Flex–Flex

$$\frac{}{\Xi_{ff} \setminus \Xi_{ff}, \cdot}\ \textbf{pu\_ff}$$

Spine traversal

$$\frac{\Xi \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \cdot \vdash \text{NIL} = \text{NIL} : a > a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_nil}$$

$$\frac{\Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A_1 > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \pi_1\, S_1 = \pi_1\, S_2 : A_1 \,\&\, A_2 > a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_fst} \qquad \frac{\Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A_2 > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash \pi_2\, S_1 = \pi_2\, S_2 : A_1 \,\&\, A_2 > a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_snd}$$

$$\frac{\Xi, (\Gamma; \Delta' \vdash U_1 = U_2 : A_1), (\Gamma; \Delta'' \vdash S_1 = S_2 : A_2 > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta', \Delta'' \vdash U_1 \hat{;} S_1 = U_2 \hat{;} S_2 : A_1 \multimap A_2 > a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_lapp}$$

$$\frac{\Xi, (\Gamma; \cdot \vdash U_1 = U_2 : A_1), (\Gamma; \Delta \vdash S_1 = S_2 : A_2 > a) \setminus \Xi_{ff}, \Theta}{\Xi, (\Gamma; \Delta \vdash U_1 ; S_1 = U_2 ; S_2 : A_1 \to A_2 > a) \setminus \Xi_{ff}, \Theta}\ \textbf{pu\_iapp}$$

Figure 8: Pre-Unification in $S^{\to\multimap\&\top}$, Equation Manipulation

lowermost and uppermost parts of Figure 8, respectively. Then, possible redices are weak head-reduced so that both sides of the equation have either a constant, a parameter or a logical variable as their head (rules **pu_redex_l** and **pu_redex_r**). When these rules can both be used, i.e. if both sides of the equation are redices, applying them in any order yields the same result (this a form of "don't care" non-determinism): we can for example adopt the convention that the left-hand side is always weak head-reduced first.

Following the standard terminology, we call a weak-head normal atomic term $H \cdot S$ *rigid* if H is a

Imitation – term construction

$$\frac{c:A \ in\ \Sigma \quad \Gamma;\Delta \vdash A \downharpoonleft^{\iota} S' \hookrightarrow S}{\Gamma;\Delta \vdash c\cdot S' \,/\, a \Uparrow^{\iota} \mathrm{NIL} \hookrightarrow c \cdot S}\ \textbf{fri\_con}$$

$$\frac{\Gamma;\Delta \vdash U \,/\, A_1 \Uparrow^{\iota} S \hookrightarrow V_1 \quad \Gamma;\Delta \vdash A_2 \hookrightarrow V_2}{\Gamma;\Delta \vdash U \,/\, A_1 \,\&\, A_2 \Uparrow^{\iota} \pi_1\, S \hookrightarrow \langle V_1, V_2\rangle}\ \textbf{fri\_pair1} \qquad \frac{\Gamma;\Delta \vdash U \,/\, A_2 \Uparrow^{\iota} S \hookrightarrow V_2 \quad \Gamma;\Delta \vdash A_1 \hookrightarrow V_1}{\Gamma;\Delta \vdash U \,/\, A_1 \,\&\, A_2 \Uparrow^{\iota} \pi_2\, S \hookrightarrow \langle V_1, V_2\rangle}\ \textbf{fri\_pair2}$$

$$\frac{\Gamma;\Delta, x:A \vdash U \,/\, B \Uparrow^{\iota} S \hookrightarrow V}{\Gamma;\Delta \vdash U \,/\, A \multimap B \Uparrow^{\iota} U \,\hat{;}\, S \hookrightarrow \hat{\lambda} x:A.\, V}\ \textbf{fri\_llam} \qquad \frac{\Gamma, x:A;\Delta \vdash U \,/\, B \Uparrow^{\iota} S \hookrightarrow V}{\Gamma;\Delta \vdash U \,/\, A \to B \Uparrow^{\iota} U ; S \hookrightarrow \lambda x:A.\, V}\ \textbf{fri\_ilam}$$

Imitation – spine construction

$$\frac{}{\Gamma;\cdot \vdash a \downharpoonleft^{\iota} \mathrm{NIL} \hookrightarrow \mathrm{NIL}}\ \textbf{fri\_nil}$$

$$\frac{\Gamma;\Delta \vdash A_1 \downharpoonleft^{\iota} S' \hookrightarrow S}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \downharpoonleft^{\iota} \pi_1\, S' \hookrightarrow \pi_1\, S}\ \textbf{fri\_fst} \qquad \frac{\Gamma;\Delta \vdash A_2 \downharpoonleft^{\iota} S' \hookrightarrow S}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \downharpoonleft^{\iota} \pi_2\, S' \hookrightarrow \pi_2\, S}\ \textbf{fri\_snd}$$

$$\frac{\Gamma;\Delta' \vdash B \downharpoonleft^{\iota} S' \hookrightarrow S \quad \Gamma;\Delta'' \vdash A \hookrightarrow V}{\Gamma;\Delta',\Delta'' \vdash A \multimap B \downharpoonleft^{\iota} U \,\hat{;}\, S' \hookrightarrow V \,\hat{;}\, S}\ \textbf{fri\_lapp} \qquad \frac{\Gamma;\Delta \vdash B \downharpoonleft^{\iota} S' \hookrightarrow S \quad \Gamma;\cdot \vdash A \hookrightarrow V}{\Gamma;\Delta \vdash A \to B \downharpoonleft^{\iota} U ; S' \hookrightarrow V ; S}\ \textbf{fri\_iapp}$$

Projection – term construction

$$\frac{\Gamma;\Delta \vdash A \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma;\Delta, x:A \vdash a \Uparrow^{\pi} \mathrm{NIL} \hookrightarrow x \cdot S}\ \textbf{frp\_lvar} \qquad \frac{\Gamma, x:A;\Delta \vdash A \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma, x:A;\Delta \vdash a \Uparrow^{\pi} \mathrm{NIL} \hookrightarrow x \cdot S}\ \textbf{frp\_ivar}$$

$$\frac{\Gamma;\Delta \vdash A_1 \Uparrow^{\pi} S \hookrightarrow V_1 \quad \Gamma;\Delta \vdash A_2 \hookrightarrow V_2}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \Uparrow^{\pi} \pi_1\, S \hookrightarrow \langle V_1, V_2\rangle}\ \textbf{frp\_pair1} \qquad \frac{\Gamma;\Delta \vdash A_2 \Uparrow^{\pi} S \hookrightarrow V_2 \quad \Gamma;\Delta \vdash A_1 \hookrightarrow V_1}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \Uparrow^{\pi} \pi_2\, S \hookrightarrow \langle V_1, V_2\rangle}\ \textbf{frp\_pair2}$$

$$\frac{\Gamma;\Delta, x:A \vdash B \Uparrow^{\pi} S \hookrightarrow V}{\Gamma;\Delta \vdash A \multimap B \Uparrow^{\pi} U \,\hat{;}\, S \hookrightarrow \hat{\lambda} x:A.\, V}\ \textbf{frp\_llam} \qquad \frac{\Gamma, x:A;\Delta \vdash B \Uparrow^{\pi} S \hookrightarrow V}{\Gamma;\Delta \vdash A \to B \Uparrow^{\pi} U ; S \hookrightarrow \lambda x:A.\, V}\ \textbf{frp\_ilam}$$

Projection – spine construction

$$\frac{}{\Gamma;\cdot \vdash a \downharpoonleft^{\pi} a \hookrightarrow \mathrm{NIL}}\ \textbf{frp\_nil}$$

$$\frac{\Gamma;\Delta \vdash A_1 \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \downharpoonleft^{\pi} a \hookrightarrow \pi_1\, S}\ \textbf{frp\_fst} \qquad \frac{\Gamma;\Delta \vdash A_2 \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma;\Delta \vdash A_1 \,\&\, A_2 \downharpoonleft^{\pi} a \hookrightarrow \pi_2\, S}\ \textbf{frp\_snd}$$

$$\frac{\Gamma;\Delta' \vdash B \downharpoonleft^{\pi} a \hookrightarrow S \quad \Gamma;\Delta'' \vdash A \hookrightarrow V}{\Gamma;\Delta',\Delta'' \vdash A \multimap B \downharpoonleft^{\pi} a \hookrightarrow V \,\hat{;}\, S}\ \textbf{frp\_lapp} \qquad \frac{\Gamma;\Delta \vdash B \downharpoonleft^{\pi} a \hookrightarrow S \quad \Gamma;\cdot \vdash A \hookrightarrow V}{\Gamma;\Delta \vdash A \to B \downharpoonleft^{\pi} a \hookrightarrow V ; S}\ \textbf{frp\_iapp}$$

Figure 9: Pre-Unification in $S^{\to\multimap\&\top}$, Generation of Substitutions

constant or a parameter, and *flexible* if it is a logical variable. Since the sides of a canonical equation ξ of base type can be only either rigid or flexible, we have four possibilities:

Rigid-Rigid: If the head of both sides of ξ is the same constant or parameter, we unify the spines.

Rigid-Flex: We reduce this case to the next by swapping the sides of the equation.

Flex-Rigid: Consider first the equation $\Gamma;\Delta \vdash F \cdot S_1 = c \cdot S_2 : a$ where the head c is a constant. Solving this equation requires instantiating F to a term V such that the root $V \cdot S_1$ reduces to a term having c as its head; the resulting spine and S_2 are then unified, as in the rigid-rigid case. We can construct V in two manners: the first, *imitation*, builds V around the constant c itself. The second, *projection*, constructs V around a bound variable x that will be substituted via β-reduction to some subterm of S_1 that might eventually be instantiated to c. In both cases, the head c or x of V is buried under a layer of constructors corresponding to the type of F (or, more to the point, to the source type of S_1); it is intended to access the subterms of S_1 once β-reduction is performed.

Constructors

$$\frac{\Gamma; \Delta \vdash a \hookrightarrow S, A \quad F : A \text{ ``new''}}{\Gamma; \Delta \vdash a \hookrightarrow F \cdot S}\ \textbf{raise\_root}$$

$$\frac{}{\Gamma; \Delta \vdash \top \hookrightarrow \langle\rangle}\ \textbf{raise\_unit} \qquad \frac{\Gamma; \Delta \vdash A_1 \hookrightarrow V_1 \quad \Gamma; \Delta \vdash A_2 \hookrightarrow V_2}{\Gamma; \Delta \vdash A_1 \,\&\, A_2 \hookrightarrow \langle V_1, V_2\rangle}\ \textbf{raise\_pair}$$

$$\frac{\Gamma; \Delta, x : A \vdash B \hookrightarrow V}{\Gamma; \Delta \vdash A \multimap B \hookrightarrow \hat{\lambda} x : A.\, V}\ \textbf{raise\_llam} \qquad \frac{\Gamma, x : A; \Delta \vdash B \hookrightarrow V}{\Gamma; \Delta \vdash A \to B \hookrightarrow \lambda x : A.\, V}\ \textbf{raise\_ilam}$$

Spines

$$\frac{}{\cdot; \cdot \vdash a \hookrightarrow \text{NIL}, a}\ \textbf{raise\_nil}$$

$$\frac{\Gamma; \Delta \vdash a \hookrightarrow S, B}{\Gamma; \Delta, x : A \vdash a \hookrightarrow (x_\eta^A \,\hat{;}\, S), A \multimap B}\ \textbf{raise\_lapp} \qquad \frac{\Gamma; \Delta \vdash a \hookrightarrow S, B}{\Gamma, x : A; \Delta \vdash a \hookrightarrow (x_\eta^A ; S), A \to B}\ \textbf{raise\_iapp}$$

Figure 10: Pre-Unification in $S^{\to\multimap\&\top}$, Raising Variables

This head is applied to local parameters that, besides matching its type, will have the effect of reshuffling appropriately the terms composing S_1. Once V has been produced, it is substituted for every occurrence of F in the equation system and the pair V/F is added to the current substitution. Flex-rigid equations with a parameter as their rigid head are treated similarly except that imitation cannot be applied since parameters are bound within the scope of logical variables.

Given an equation $\xi = (\Gamma; \Delta \vdash F \cdot S_1 = c \cdot S_2 : a)$, the construction of the instantiating term V in the case of imitation is described in the upper part of Figure 9. The judgment

$$\Gamma'; \Delta' \vdash c \cdot S_2 \,/\, A' \Uparrow^{\iota} S_1' \hookrightarrow V'$$

builds the constructors layer of V on the basis of the type A of F (that is also the source type of S_1). Here, $c \cdot S_2$ is the right-hand side of ξ, A' and S_1' are initially set to A and S_1 respectively, and then to subexpressions of theirs as the computation of V proceeds, and Γ' and Δ' are initialized to the empty context. V', to be thought of as the "output value" of this judgment, corresponds to intermediate stages of the construction of V. Whenever this judgment is derivable, we have that $\Gamma'; \Delta' \vdash_\Sigma V' : A'$ and $\Gamma; \Delta^* \vdash_\Sigma S_1' : A' > a$ are derivable. In the latter invariant, Δ^* is some submultiset of the linear context Δ of ξ, and a is the type of this equation.

As V is constructed, the local parameters bound by linear and intuitionistic λ-abstraction (rules **fri_llam** and **fri_ilam**) are stored in the accumulators Γ' and Δ' respectively. When A' has the form $A_1' \,\&\, A_2'$ (rules **fri_pair1** and **fri_pair2**), V' must be a pair $\langle V_1', V_2'\rangle$ and S_1' must start with a projection. The subterm V_i' that is projected away can be arbitrary as long as it has type A_i' and uses up all local parameters in $\Gamma'; \Delta'$; this is achieved by means of the variable raising judgment discussed below.

When a base type is eventually reached (rule **fri_con**), the right-hand side $c \cdot S_2$ of the original equation is accessed, the constant c is installed as the head of V and its spine S is constructed by looking at the spine S_2 and inserting the local parameters accumulated in $\Gamma'; \Delta'$. The spine S is built by the judgment

$$\Gamma''; \Delta'' \vdash B' \Downarrow^{\iota} S_2' \hookrightarrow S'$$

where B', S_2', Γ'' and Δ'' are initially set to the type B of c, the spine S_2 and the accumulators Γ' and Δ' respectively, and then to subexpressions (subcontexts) as the computation of S proceeds. The "output value" S' corresponds to intermediate stages of the construction of S. The invariants for this judgment are $\Gamma''; \Delta'' \vdash_\Sigma S' : B' > a$ and $\Gamma; \Delta^* \vdash_\Sigma S_1' : B' > a$, where again Δ^* is a subcontext of Δ.

S is constructed by mimicking the structure of S_2 in the sense that both will consist of the same sequence of spine constructors although possibly applied to different arguments. This invariant relates S' and S'_2 as well and will be formalized as $S' \sim S'_2$ in Section 3.4.2. Notice the use of the variable raising judgment (discussed below) in rules **fri_lapp** and **fri_iapp** to construct appropriate η-long arguments with new logical variables as heads applied to the parameters in $\Gamma''; \Delta''$.

The construction of V in the case of projection, displayed in the lower part of Figure 9, is similar. Given an equation $\xi = (\Gamma; \Delta \vdash F \cdot S_1 = h \cdot S_2 : a)$ with h a constant or a parameter, the instantiating term V for F is constructed by means of the judgment

$$\Gamma'; \Delta' \vdash A' \Uparrow^{\pi} S'_1 \hookrightarrow V'.$$

Here, A' is initialized to the type A of F, S'_1 to the spine S_1 and both accumulators Γ' and Δ' to the empty context. The "output value" V' represents intermediate stages of the calculation of V. The main invariants for this judgment are similar to the case of imitation. Observe that, differently from imitation, the right-hand side of ξ is not taken into consideration in this judgment, and therefore in the construction of V. There can be a combinatorial explosion in the number of instantiating terms V that are generated in this way. The absence of guidance from the term $h \cdot S_2$ will cause most of them to be discarded. This is a major source of inefficiency and divergence in a Huet-like algorithm.

The head of V relative to S_1 is set to a local parameter x from Γ' or Δ' (rules **frp_lvar** and **frp_ivar**, respectively). The corresponding spine S is constructed by means of the judgment

$$\Gamma''; \Delta'' \vdash A' \downarrow^{\pi} a \hookrightarrow S'.$$

Here, A' is initialized to the type A of x, Γ'' to Γ', and Δ'' to Δ' (after withdrawing $x : A$ from it, if x is linear). Here, a is the type of the original equation ξ. As in previous cases, S' is the "output value" of this subprocedure and corresponds to intermediate stages of the construction of S. Whenever this judgment is derivable, there is always a derivation of $\Gamma''; \Delta'' \vdash_{\Sigma} S' : A' > a$.

The main difference with respect to the analogous imitation judgment is that the spine S is built on the basis of the type A of the projected parameter (rules **frp_lvar** and **frp_ivar**) rather than relative to the spine in the right-hand side of the equation. This leads to a form of non-determinism for product types not present in the case of imitation (rules **frp_fst** and **frp_snd**).

The purpose of the variable raising judgment

$$\Gamma'; \Delta' \vdash A \hookrightarrow V,$$

displayed in Figure 10, is to produce an η-long term V of type A with new logical variables as its heads (rule **raise_root**) and the parameters accumulated in $\Gamma'; \Delta'$ in the corresponding spines. Notice that functional types yield new local parameters (rules **raise_llam** and **raise_ilam**). Whenever this judgment is derivable, there is a derivation of $\Gamma'; \Delta' \vdash_{\Sigma} V : A$.

The spines themselves are constructed by means of the judgment

$$\Gamma'; \Delta' \vdash a \hookrightarrow S, A$$

which, given Γ', Δ' and a, builds a spine S mapping heads of type A to roots of type a by non-deterministically rearranging the parameters present in $\Gamma'; \Delta'$. We have $\Gamma'; \Delta' \vdash_{\Sigma} a : S > A$ as an invariant for this judgment.

Observe that, if $\Gamma'; \Delta'$ contains n assumptions altogether, this judgment has $n!$ derivations, which yield as many spines S and types A. The actual permutation that is picked is however unimportant since it only affects the type (A) of the "new" logical variable F in rule **raise_root**. Therefore, the choices present in rules **raise_lapp** and **raise_iapp** (and choosing between them) is a form of "don't care" non-determinism.

Flex-Flex: Similarly to $\lambda^{\rightarrow}$, a system composed uniquely of flex-flex equations is always solvable in $S^{\rightarrow -\circ \& \top}$. Indeed, every logical variable F in it can be instantiated to a term V_F consisting of a layer of constructors as dictated by the type of F, but with every root set to $G_a \cdot \langle\rangle \,\hat{;}\, \mathrm{NIL}$ (i.e. $G_a\,\hat{}\,\langle\rangle$ in $\lambda^{\rightarrow -\circ \& \top}$), where G_a is a common new logical variable of type $\top \multimap a$, for each base type a. Then, after normalization, every equation ξ reduces to $\Gamma_\xi; \Delta_\xi \vdash G_a \cdot \langle\rangle \,\hat{;}\, \mathrm{NIL} = G_a \cdot \langle\rangle \,\hat{;}\, \mathrm{NIL} : a$ which is linearly valid, although extensionally solvable only if a ground substitution term for each needed G_a can indeed be constructed. When this situation is encountered, the procedure terminates with success, but without instantiating the logical variables appearing in it. The substitution constructed up to this point, called a *pre-unifier*, is returned.

The possibility of achieving an algorithm à la Huet depends crucially on flex-flex equations being always solvable. If this property does not hold, as in some sublanguages of $S^{\rightarrow -\circ \& \top}$ we will discuss in this paper, these equations must be analyzed with techniques similar to [JP76] or [Mil91].

We will now discuss a number of simple examples in order to gain familiarity with this algorithm. We will focus our attention on the flex-rigid and rigid-rigid cases.

Example 1: In the signature $\Sigma_1 = (c\!:\!a)$ and pool $\Phi_1 = (F\!:\!a \rightarrow a)$, consider the following equation ξ_1, written in the syntax of $\lambda^{\rightarrow -\circ \& \top}$ for simplicity:

$$\cdot;\cdot \vdash F\; c = c : a.$$

(this equation corresponds to $\cdot;\cdot \vdash F \cdot (c \cdot \mathrm{NIL}); \mathrm{NIL} = c \cdot \mathrm{NIL} : a$ in $S^{\rightarrow -\circ \& \top}$.)

ξ_1 has the two following solutions, again expressed in the syntax of $\lambda^{\rightarrow -\circ \& \top}$ (and of $S^{\rightarrow -\circ \& \top}$ in parentheses); we use bracketed indices to distinguish these solutions:

$$F^{(1)} \longleftarrow \lambda x'\!:\!a.\, c \qquad (F^{(1)} \longleftarrow \lambda x'\!:\!a.\, c \cdot \mathrm{NIL})$$
$$F^{(2)} \longleftarrow \lambda x'\!:\!a.\, x' \qquad (F^{(2)} \longleftarrow \lambda x'\!:\!a.\, x' \cdot \mathrm{NIL})$$

The first is obtained by imitation as witnessed by the presence of the constant c as the head of the instantiating term for F. The second is the result of projection, indicated by bound variable x'.

Example 2: In a signature Σ_2 identical to Σ_1, but with the pool $\Phi_2 = (F\!:\!a \multimap a)$, consider the equation ξ_2:

$$\cdot;\cdot \vdash F\,\hat{}\,c = c : a$$

that differs from ξ_1 only by F standing for a linear rather than an intuitionistic function. This equation has a single solution obtained by projection:

$$F \longleftarrow \hat{\lambda} x'\!:\!a.\, x' \qquad (F \longleftarrow \hat{\lambda} x'\!:\!a.\, x' \cdot \mathrm{NIL})$$

Indeed the instantiating term

$$\hat{\lambda} x'\!:\!a.\, c \qquad (\hat{\lambda} x'\!:\!a.\, c \cdot \mathrm{NIL})$$

corresponding to the solution obtained by imitation in the previous example, is not linearly valid since the parameter x' is never used. The impossibility to apply rule **fri_nil** prevents our pre-unification algorithm from producing this term as a solution: this rule expects an empty linear local parameters accumulator while in this case it contains $x'\!:\!a$.

Example 3: Next, we analyze in depth one of the equations considered in Section 3.2:

$$x\!:\!a, y\!:\!a; \cdot \vdash F\,\hat{}\,x\,\hat{}\,y = c\,\hat{}(G_1\; x\; y)\,\hat{}(G_2\; x\; y) : a.$$

The signature of this equation, ξ_3, is $\Sigma_3 = (c\!:\!a \multimap a \multimap a)$ and the variable pool at hand is $\Phi_3 = (F\!:\! a \multimap a \multimap a, G_1\!:\!a \rightarrow a \rightarrow a, G_2\!:\!a \rightarrow a \rightarrow a)$. The application of our algorithm yields the following four instantiating terms for F, all obtained by imitation, and, after weak head-normalization, the equations to their right, where Γ stands for the intuitionistic context $(x:a, y:a)$.

$$
\begin{array}{ll}
F^{(1)} \longleftarrow \hat{\lambda} x':a.\,\hat{\lambda} y':a.\, c\,\hat{}\,(F_1\,\hat{}\,x'\,\hat{}\,y')\,\hat{}\,F_2 & \xi_3^{(1)}\!: \Gamma;\cdot \vdash c\,\hat{}\,(F_1\,\hat{}\,x\,\hat{}\,y)\,\hat{}\,F_2 = c\,\hat{}\,(G_1\;x\;y)\,\hat{}\,(G_2\;x\;y) : a \\
F^{(2)} \longleftarrow \hat{\lambda} x':a.\,\hat{\lambda} y':a.\, c\,\hat{}\,(F_1\,\hat{}\,x')\,\hat{}\,(F_2\,\hat{}\,y') & \xi_3^{(2)}\!: \Gamma;\cdot \vdash c\,\hat{}\,(F_1\,\hat{}\,x)\,\hat{}\,(F_2\,\hat{}\,y) = c\,\hat{}\,(G_1\;x\;y)\,\hat{}\,(G_2\;x\;y) : a \\
F^{(3)} \longleftarrow \hat{\lambda} x':a.\,\hat{\lambda} y':a.\, c\,\hat{}\,(F_1\,\hat{}\,y')\,\hat{}\,(F_2\,\hat{}\,x') & \xi_3^{(3)}\!: \Gamma;\cdot \vdash c\,\hat{}\,(F_1\,\hat{}\,y)\,\hat{}\,(F_2\,\hat{}\,x) = c\,\hat{}\,(G_1\;x\;y)\,\hat{}\,(G_2\;x\;y) : a \\
F^{(4)} \longleftarrow \hat{\lambda} x':a.\,\hat{\lambda} y':a.\, c\,\hat{}\,F_1\,\hat{}\,(F_2\,\hat{}\,x'\,\hat{}\,y') & \xi_3^{(4)}\!: \Gamma;\cdot \vdash c\,\hat{}\,F_1\,\hat{}\,(F_2\,\hat{}\,x\,\hat{}\,y) = c\,\hat{}\,(G_1\;x\;y)\,\hat{}\,(G_2\;x\;y) : a
\end{array}
$$

The logical variables F_1 and F_2 appearing in the instantiating terms for F are produced by the variable raising judgment in rule **fri_lapp**. They contribute to the variable pool $\Phi_3^{(i)}$ of equations $\xi_3^{(i)}$ with types $(F_1 : a \multimap a \multimap a, F_2 : a)$, $(F_1 : a \multimap a, F_2 : a \multimap a)$, $(F_1 : a \multimap a, F_2 : a \multimap a)$ and $(F_1 : a, F_2 : a \multimap a \multimap a)$, respectively.

Each of the $\xi_3^{(i)}$ is a rigid-rigid equation with the constant c as its head. It is therefore processed by rule **pu_rr_con**. Two uses of rule **pu_lapp** followed by rule **pu_nil** produce the following four sets of flex-flex residual equations:

$$
\begin{array}{ll}
\Xi_3^{(1)} : (\Gamma;\cdot \vdash F_1\,\hat{}\,x\,\hat{}\,y = G_1\;x\;y : a, & \Gamma;\cdot \vdash F_2 = G_2\;x\;y : a) \\
\Xi_3^{(2)} : (\Gamma;\cdot \vdash F_1\,\hat{}\,x = G_1\;x\;y : a, & \Gamma;\cdot \vdash F_2\,\hat{}\,y = G_2\;x\;y : a) \\
\Xi_3^{(3)} : (\Gamma;\cdot \vdash F_1\,\hat{}\,y = G_1\;x\;y : a, & \Gamma;\cdot \vdash F_2\,\hat{}\,x = G_2\;x\;y : a) \\
\Xi_3^{(4)} : (\Gamma;\cdot \vdash F_1 = G_1\;x\;y : a, & \Gamma;\cdot \vdash F_2\,\hat{}\,x\,\hat{}\,y = G_2\;x\;y : a)
\end{array}
$$

Each of these situations triggers the application of rule **pu_ff** and the pre-unification procedure terminates returning the above instantiating terms for F, and the residual flex-flex equation systems $\Xi^{(i)}$ as constraints. At this point, our algorithm stops.

Notice that, in this specific case (we are dealing with pattern equations, see Section 4.2), the residual equation systems could be further simplified, obtaining the following solutions for G_1 and G_2, which, together with the corresponding instantiation for F, constitute four solutions for the original equation ξ_3.

$$
\begin{array}{ll}
G_1^{(1)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_1\,\hat{}\,x'\,\hat{}\,y' & G_2^{(1)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_2 \\
G_1^{(2)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_1\,\hat{}\,x' & G_2^{(2)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_2\,\hat{}\,y' \\
G_1^{(3)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_1\,\hat{}\,y' & G_2^{(3)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_2\,\hat{}\,x' \\
G_1^{(4)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_1 & G_2^{(4)} \longleftarrow \lambda x':a.\,\lambda y':a.\,F_2\,\hat{}\,x'\,\hat{}\,y'
\end{array}
$$

The variables F_1 and F_2 can be instantiated with any term of the appropriate type, assuming that such terms can be constructed. Observe that this cannot be achieved with the constant c alone.

Example 4: As our next example, we consider the equation ξ_4, written in the syntax of $S^{\rightarrow\multimap\&\top}$ in parentheses:

$$
\cdot; x:a \vdash \mathrm{FST}\,(F\,\hat{}\,x) = f\,\hat{}\,x : a \qquad\qquad (\cdot; x:a \vdash F \cdot x\,\hat{;}\,\pi_1\,\mathrm{NIL} = f \cdot x\,\hat{;}\,\mathrm{NIL} : a)
$$

where F has type $a \multimap (a \,\&\, a)$ in the pool Φ_4 and the current signature is $\Sigma_4 = (f : a \multimap a)$. We have one solution obtained by imitation

$$
F \longleftarrow \hat{\lambda} x':a.\,\langle f\,\hat{}\,(F_1\,\hat{}\,x'), F_2\,\hat{}\,x'\rangle \qquad\qquad (F \longleftarrow \hat{\lambda} x':a.\,\langle f \cdot (F_1 \cdot (x'\,\hat{;}\,\mathrm{NIL}))\,\hat{;}\,\mathrm{NIL}, F_2 \cdot x'\,\hat{;}\,\mathrm{NIL}\rangle)
$$

The logical variables F_1 and F_2, both of type $a \multimap a$, have again been introduced by the raising procedure. The origin of the first is in rule **fri_lapp**, while the second is a by-product of the application of rule **fri_pair1**. Since x' is a linear parameter, it must occur linearly in both subexpressions of the additive pairing construct. The fact that x' is an argument to F_1 and F_2 ensures that it will be used linearly by any instantiating term for these variables.

Substituting the above term for F and performing weak head-normalization yields the following rigid-rigid equation ξ_4'

$$
\cdot; x:a \vdash f\,\hat{}\,(F_1\,\hat{}\,x) = f\,\hat{}\,x : a
$$

which, after applying rules **pu_rr_con**, **pu_lapp** and **pu_nil**, reduces to an equation similar to the one analyzed in our second example above. The overall substitution is

$$F_1 \longleftarrow \hat{\lambda} x' : a.\, x', \quad F \longleftarrow \hat{\lambda} x' : a.\, \langle f\,\hat{}\,x', F_2\,\hat{}\,x' \rangle$$

and no flex-flex equation is produced.

Example 5: The next example is intended to demonstrate how complex a situation can be when logical variables have functional parameters. The signature Σ_5 of this simple instance is $(c : a, f : a \to a)$. We have also the variable pool $\Phi_5 = (F : (a \to a) \multimap a)$. The equation ξ_5 is the following:

$$\cdot; \cdot \vdash F\,\hat{}\,(\lambda y : a.\, f\ y) = f\,\hat{}\,c : a$$

The use of imitation produces the following substitution for F and equation ξ_5', where we have made an implicit use of rule **pu_redex_l** followed by **pu_rr_con**, **pu_lapp** and **pu_nil**:

$$F^{(1)} \longleftarrow \hat{\lambda} x : a \to a.\, f\,\hat{}\,(F_1\,\hat{}\,(\lambda z : a.\, x\ z)) \qquad \xi_5' \colon \cdot; \cdot \vdash F_1\,\hat{}\,(\lambda y : a.\, f\ y) = c : a$$

where F_1 has type $(a \to a) \multimap a$. Imitation cannot be applied to ξ_5' because c does not accept arguments while the linear argument of F_1 must be consumed somehow. Projection yields the following substitution–equation pair (after simplification):

$$F_1 \longleftarrow \hat{\lambda} x : a \to a.\, x\,\hat{}\,F_1' \qquad \cdot; \cdot \vdash f\,\hat{}\,F_1' = c : a$$

where the type of F_1' is a. The equation on the right-hand side is clearly non solvable and indeed no rule can be applied to further reduce it. We must backtrack to our original equation ξ_5.

We are therefore left to attempting projection, which yields the following instantiating term for F, that, after substitution and weak head-normalization, gives rise to the equation ξ_5' on the right-hand side.

$$F^{(2)} \longleftarrow \hat{\lambda} x : a \to a.\, x\,\hat{}\,F_2 \qquad \xi_5'' \colon \cdot; \cdot \vdash f\,\hat{}\,F_2 = f\,\hat{}\,c : a$$

Again, F_2 derives from variable raising in rule **fri_iapp**. ξ_5'' is a rigid-rigid equation: the application of rules **pu_rr_con**, **pu_iapp** and **pu_nil** reduces it to the flex-rigid (first-order) equation

$$\cdot; \cdot \vdash F_2 = c : a$$

with the obvious solution

$$F_2 \longleftarrow c$$

obtained by imitation. The solution returned by our pre-unification algorithm is therefore, after composing these two substitutions,

$$F_2 \longleftarrow c, \quad F \longleftarrow \hat{\lambda} x : a \to a.\, x\,\hat{}\,c.$$

Notice that F_2 is not mentioned anywhere and could therefore be dropped. No residual flex-flex equation is produced.

Example 6: As our final example, consider the flex-flex equation ξ_6:

$$x : a, y : a; \cdot \vdash F_1\,\hat{}\,x = F_2\,\hat{}\,y : a$$

in some signature Σ_6 and with respect to variable pool $\Phi_6 = (F_1 : a \multimap a, F_2 : a \multimap a)$. Our pre-unification algorithm returns this equation untouched as a constraint by means of rule **pu_ff**.

Since it is a flex-flex equation, ξ_6 has the solution

$$F_1 \longleftarrow \hat{\lambda} x' : a.\, G\,\hat{}\,\langle\rangle, \quad F_2 \longleftarrow \hat{\lambda} x' : a.\, G\,\hat{}\,\langle\rangle$$

where G is a new logical variable of type $\top \multimap a$. The relevance of this substitution as a solution for ξ_6 depends on the specific applications: it is an *open solution* in the sense that it mentions logical variables (G in this case). The existence of *ground* or *closed* solutions that do not mention any logical variables depends on whether Σ_6 is equipped with constants permitting the construction of at least one (ground) term of the appropriate type, $\top \multimap a$ in our case.

3.4 Soundness and Completeness

The procedure we just described is not guaranteed to terminate for generic equation systems since flex-rigid steps can produce arbitrarily complex new equations. However, it is sound in the sense that if a unifier or pre-unifier is returned the system is solvable (where free variables are allowed in the second case). It is also non-deterministically complete, i.e., every solution to the original system is an instance of a unifier or pre-unifier which can be found with our procedure.

We dedicate this section to proving these properties. The relatively straightforward proof of soundness can be found in Section 3.4.1. Proving completeness is much more involved since it requires gaining a deep understanding of the auxiliary judgments defined in Figures 9–10. We first give some definitions that will be needed for this proof in Section 3.4.2, and then prove the completeness theorem itself and a number of auxiliary lemmas in Section 3.4.3.

3.4.1 Soundness

Proving the soundness of our linear pre-unification algorithm is particularly simple since we do not need to deal with the intricacies of instantiating-term formation. The proof that it returns a solution when the original system is indeed solvable proceeds by a simple induction.

Theorem 3.2 (*Soundness of linear pre-unification*)

If $\mathcal{X} :: \Xi \setminus \Xi_{\mathit{ff}}, \Theta$ *and there is a substitution* Θ_{ff} *such that the multiset of equality judgments* $[\Theta_{\mathit{ff}}]\Xi_{\mathit{ff}}$ *has a derivation* $\vec{\mathcal{E}}$*, then* $[\Theta_{\mathit{ff}} \circ \Theta]\Xi$ *is derivable.*

Proof.

We proceed by induction on the structure of $\mathcal{X}$. The last rule applied in $\mathcal{X}$ can belong to one of the following four categories.

Simplification rules: We group under this category any rule that does not involve directly logical variables. More specifically, we have all the inference rules in the 'term traversal', 'spine traversal' and 'rigid-rigid' segments of Figure 8. These cases are handled trivially since there is a perfect match between these rules and corresponding equality rules.

As an example, we will carry out the case concerning rule **pu_rr_lvar**.

Let $\xi = (\Gamma; \Delta, x\!:\!A \vdash x \cdot S_1 = x \cdot S_2 : a)$ and $\xi' = (\Gamma; \Delta \vdash S_1 = S_2 : A > a)$. Then, we have that:

$$\mathcal{X} = \cfrac{\begin{array}{c}\mathcal{X}' \\ \Xi', \xi' \setminus \Xi_{\mathit{ff}}, \Theta\end{array}}{\Xi', \xi \setminus \Xi_{\mathit{ff}}, \Theta}\,\textbf{pu\_rr\_lvar}$$

where $\Xi = \Xi', \xi$ and there is a substitution Θ_{ff} such that $\vec{\mathcal{E}} :: [\Theta_{\mathit{ff}}]\Xi_{\mathit{ff}}$.

By induction hypothesis, the multiset of equality judgments $[\Theta_{\mathit{ff}} \circ \Theta](\Xi', \xi')$ has a derivation $\vec{\mathcal{E}}'$. Let $\mathcal{E}'_{\xi'}$ be a derivation of $[\Theta_{\mathit{ff}} \circ \Theta]\xi'$, i.e.

$$\mathcal{E}'_{\xi'} :: \Gamma; \Delta \vdash [\Theta_{\mathit{ff}} \circ \Theta]S_1 = [\Theta_{\mathit{ff}} \circ \Theta]S_2 : A > a$$

by definition of substitution application. Then, by rule **Seq_lvar**, there is a derivation $\vec{\mathcal{E}}'_{\xi}$ of

$$\Gamma; \Delta, x\!:\!A \vdash [\Theta_{\mathit{ff}} \circ \Theta](x \cdot S_1) = [\Theta_{\mathit{ff}} \circ \Theta](x \cdot S_2) : a$$

i.e. of $[\Theta_{\mathit{ff}} \circ \Theta]\xi$, that together with the remaining elements of $\vec{\mathcal{E}}'$ constitutes the desired multiset of derivations for $[\Theta_{\mathit{ff}} \circ \Theta](\Xi', \xi)$.

Rigid-flex rule: We use this label to indicate rule **pu_rf**. We prove this case by relying on the fact that the equality judgment admits symmetry (Lemma 2.19).

Let h be a constant or a parameter, $\xi = (\Gamma; \Delta \vdash h \cdot S_1 = F \cdot S_2 : a)$ and $\xi' = (\Gamma; \Delta \vdash F \cdot S_2 = h \cdot S_1 : a)$. Then, we have that:

$$\mathcal{X} = \cfrac{\begin{array}{c}\mathcal{X}' \\ \Xi', \xi' \setminus \Xi_{f\!f}, \Theta\end{array}}{\Xi', \xi \setminus \Xi_{f\!f}, \Theta}\ \textbf{pu\_rf}$$

where $\Xi = \Xi', \xi$.

By induction hypothesis, there is an equality derivation $\vec{\mathcal{E}}'$ of $[\Theta_{f\!f} \circ \Theta](\Xi', \xi')$, which contains a derivation $\mathcal{E}'_{\xi'}$ of $[\Theta_{f\!f} \circ \Theta]\xi'$. Since by Lemma 2.19 equality is symmetric, there is a derivation $\mathcal{E}'_{\xi}$ of

$$\Gamma; \Delta \vdash [\Theta_{f\!f} \circ \Theta](h \cdot S_1) = [\Theta_{f\!f} \circ \Theta](F \cdot S_2) : a$$

and this concludes this part of the proof.

Flex-rigid rules: We consider here the flex-rigid family of rules from Figure 8. We will exemplify the treatment of this class by considering rule **pu_fr_imit**.

Let $\xi = (\Gamma; \Delta \vdash F \cdot S_1 = c \cdot S_2 : a)$. Then,

$$\mathcal{X} = \cfrac{\cdots \quad \begin{array}{c}\mathcal{X}' \\ [V/F](\Xi', \xi) \setminus \Xi_{f\!f}, \Theta'\end{array}}{\Xi', \xi \setminus \Xi_{f\!f}, (\Theta' \circ V/F)}\ \textbf{pu\_fr\_imit}$$

where $\Xi = \Xi', \xi$ and $\Theta = \Theta' \circ V/F$.

By induction hypothesis, there is an equality derivation $\vec{\mathcal{E}}'$ of $[\Theta_{f\!f} \circ \Theta']([V/F]\Xi)$. By definition of substitution composition, this expression rewrites to $[(\Theta_{f\!f} \circ \Theta') \circ V/F]\Xi$. Since substitution composition is associative, this is equivalent to $[\Theta_{f\!f} \circ (\Theta' \circ V/F)]\Xi$, and this concludes this case of the proof.

Success rule: The one remaining possibility as the last inference of $\mathcal{X}$ is rule **pu_ff**. We have therefore the following derivation $\mathcal{X}$:

$$\mathcal{X} = \cfrac{}{\Xi_{f\!f} \setminus \Xi_{f\!f}, \cdot}\ \textbf{pu\_ff}$$

with $\Xi = \Xi_{f\!f}$ and $\Theta = \cdot$.

By assumption, we know that $\vec{\mathcal{E}} :: [\Theta_{f\!f}]\Xi_{f\!f}$. Then, by a well-known property of substitutions, we have also that $\vec{\mathcal{E}} :: [\Theta_{f\!f} \circ \cdot]\Xi_{f\!f}$, which is what we had to prove. ☑

It is not difficult to generalize this procedure to full unification (as, for example, in [SG89]), although we fail to see its practical value.

3.4.2 Preliminary Definitions for the Completeness Theorem

In this section, we have grouped several definitions and minor properties we will rely on in the proof of the completeness theorem. We need approximate forms of typing and equality for spines, and the definitions of the orderings we will base the inductive proof of the theorem on.

Approximation of Spine Typing

We observed earlier that the derivability of a spine equality judgment $\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$ does not imply in general that the typing judgments $\Gamma; \Delta \vdash_\Sigma S_i : A > a$ have derivations, for $i = 1, 2$. However, for this judgment to hold, the structure of the spines S_i cannot be arbitrary. We denote the minimal requirement expressed in the rules for equality in Figure 5 by means of the relation $S \div A$, which we read *spine S respects type A*. It is defined as follows:

$$\begin{array}{rl} \mathrm{NIL} \div a & \textit{always} \\ \pi_1\, S \div A_1 \,\&\, A_2 & \textit{if} \quad S \div A_1 \\ \pi_2\, S \div A_1 \,\&\, A_2 & \textit{if} \quad S \div A_2 \\ U \,\hat{;}\, S \div A_1 \multimap A_2 & \textit{if} \quad S \div A_2 \\ U ; S \div A_1 \to A_2 & \textit{if} \quad S \div A_2 \end{array}$$

It is easy to prove that if $\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$ is derivable, then $S_i \div A$ holds, for $i = 1, 2$, according to the above definition. We will use this notion when dealing with equations as an approximation of typing, and when the spine at hand contains logical variables and is therefore not typable in the given signature. Similar definitions can be made for terms, but we will not need them.

Approximation of Spine Equality

In the auxiliary lemmas to the completeness theorem, we will often need to assume the existence of different stages of instantiation of the same spine. Relying on equality judgments for this purpose is feasible, but results in obscure statements. Instead, we capture the minimal compatibility requirements for two spines S_1 and S_2 to have a common instance by means of the relation $S_1 \sim S_2$, defined as follow:

$$\begin{array}{rl} \mathrm{NIL} \sim \mathrm{NIL} & \textit{always} \\ \pi_1\, S_1 \sim \pi_1\, S_2 & \textit{if} \quad S_1 \sim S_2 \\ \pi_2\, S_1 \sim \pi_2\, S_2 & \textit{if} \quad S_1 \sim S_2 \\ U_1 \,\hat{;}\, S_1 \sim U_2 \,\hat{;}\, S_2 & \textit{if} \quad S_1 \sim S_2 \\ U_1 ; S_1 \sim U_2 ; S_2 & \textit{if} \quad S_1 \sim S_2 \end{array}$$

It is easy to prove that whenever the judgment $\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$ is derivable, then $S_1 \sim S_2$ holds. Notice also that, if $S_1 \sim S_2$, then there is a type A such that $S_1 \div A$ and $S_2 \div A$ hold. The opposite entailment fails because of the presence of product types.

Relative Heads

Next, we wish to identify the head of a canonical term U with respect to a spine S, where both might contain logical variables. In the simply typed λ-calculus $\lambda^{\to}$, an accompanying spine would be unnecessary since every term has exactly one head. In $S^{\to\multimap\&\top}$, the presence of pairs complicates this situation. We rely on a spine to locate the head we are interested in among the many the term at hand might contain. The *head of U relative to S*, written $H_S(U)$, is defined as follows:

$$\begin{array}{rcl} H_{\mathrm{NIL}}(x \cdot S) & = & x \\ H_{\mathrm{NIL}}(c \cdot S) & = & c \\ H_{\mathrm{NIL}}(F \cdot S) & = & F \\ H_{\pi_1\, S}(\langle U_1, U_2 \rangle) & = & H_S(U_1) \\ H_{\pi_2\, S}(\langle U_1, U_2 \rangle) & = & H_S(U_2) \\ H_{V \,\hat{;}\, S}(\hat{\lambda} x : A.\, U) & = & H_S(U) \\ H_{V ; S}(\lambda x : A.\, U) & = & H_S(U) \end{array}$$

Notice that this function is partial: it is undefined in the situations not listed in this definition. In particular, our assumption that U is in canonical form is essential since we did not provide a case for redices. However, it is easy to prove that whenever U has some type A and there is a derivation of $\Gamma; \Delta \vdash_\Sigma S : Aa$ for some contexts Γ, Δ and base type a, then $H_S(U)$ is defined.

In the following, we will rely on a simple property of this notion:

Lemma 3.3 (*Relative heads*)

i. If $H_S(U) = c$, *then* $\mathrm{Can}(U \cdot S) = c \cdot S'$ *for some spine* S';

ii. If $H_S(U) = F$, *then* $\mathrm{Can}(U \cdot S) = F \cdot S'$ *for some spine* S'.

Proof.

By induction on the structure of U and S. An auxiliary induction on the reduction sequence is needed to cope with functional object. ☑

A similar property does not hold for parameters since β-reduction can change a bound parameter in U to an arbitrary term.

Instantiating-Term Ordering

We conclude this section with the definition of the two ordering relations we will use to carry on the inductive argument in the proof of the completeness theorem. The first of these orderings, denoted $Us \sqsubset Vs$, where Us and Vs are multisets of terms, specifies Us differs from Vs only by the fact that some terms in Us are subterms of a term V in Vs, abstracting from the presence of constructors.

An example will help gain some intuition about this notion. We want for instance that

$$(\lambda x\!:\!a.\,\lambda y\!:\!a.\,x,\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,y)\ \sqsubset\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,c\,x\,y$$

since both x and y are subterms of $c\,x\,y$. It will be useful to express this example according to the syntax of $S^{\rightarrow\multimap\&\top}$:

$$(\lambda x\!:\!a.\,\lambda y\!:\!a.\,x\cdot \mathrm{NIL},\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,y\cdot \mathrm{NIL})\ \sqsubset\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,c\cdot(x\cdot \mathrm{NIL});(y\cdot \mathrm{NIL});\mathrm{NIL}.$$

In the proof of the completeness theorem, Us and Vs will be the images of two substitutions. The former will have to be shown smaller than the latter in order to apply the induction hypothesis.

We define the $\sqsubset$ relation in stages on different entities, but take the liberty to overload this symbol as well as the auxiliary $\sqsubseteq$. The distinction should be clear from the context. We have the following definition:

$\boldsymbol{U =_{raise} V}$**:** We write $U =_{raise} V$ to denote the fact that two terms U and V differ only by the presence of leading abstractions in U. V will always be a root. This relation is formally defined as follows.

$$\begin{array}{rl} U =_{raise} U & \\ \hat{\lambda} x\!:\!A.\,U =_{raise} V & \textit{if}\ \ U =_{raise} V \\ \lambda x\!:\!A.\,U =_{raise} V & \textit{if}\ \ U =_{raise} V \end{array}$$

In the example above, we have that

$$\begin{array}{cccc} & \lambda x\!:\!a.\,\lambda y\!:\!a.\,x\cdot \mathrm{NIL} =_{raise} x\cdot \mathrm{NIL} & \textit{and} & \lambda x\!:\!a.\,\lambda y\!:\!a.\,y\cdot \mathrm{NIL} =_{raise} y\cdot \mathrm{NIL}, \\ \textit{i.e.} & \lambda x\!:\!a.\,\lambda y\!:\!a.\,x =_{raise} x & \textit{and} & \lambda x\!:\!a.\,\lambda y\!:\!a.\,y =_{raise} y. \end{array}$$

$\boldsymbol{Us \sqsubseteq V}$**:** We recursively extend the above relation so that its right-hand side operates on terms of arbitrary type, and not just on roots, and its left-hand side is a multiset of terms.

$$\begin{array}{rl} \cdot \sqsubseteq \langle\rangle & \textit{always} \\ U \sqsubseteq H\cdot S & \textit{if}\ \ U =_{raise} H\cdot S \\ Us \sqsubseteq \langle V_1, V_2\rangle & \textit{if}\ \ Us = (Us_1, Us_2),\ Us_1 \sqsubseteq V_1\ \textit{and}\ Us_2 \sqsubseteq V_2 \\ Us \sqsubseteq \hat{\lambda} x\!:\!A.\,V & \textit{if}\ \ Us \sqsubseteq V \\ Us \sqsubseteq \lambda x\!:\!A.\,V & \textit{if}\ \ Us \sqsubseteq V \end{array}$$

In the previous example, the first of these specifications allows us to conclude that

$$\lambda x\!:\!a.\,\lambda y\!:\!a.\,x\cdot \mathrm{NIL} \sqsubseteq x\cdot \mathrm{NIL} \qquad \textit{and} \qquad \lambda x\!:\!a.\,\lambda y\!:\!a.\,y\cdot \mathrm{NIL} \sqsubseteq y\cdot \mathrm{NIL}.$$

$Us \sqsubseteq S$: We extend the above relation so that it matches all the arguments of a spine with a given multiset of terms.

$$\begin{array}{rll} \cdot \sqsubseteq \text{NIL} & \textit{always} & \\ Us \sqsubseteq \pi_1\, S & \textit{if} & Us \sqsubseteq S \\ Us \sqsubseteq \pi_2\, S & \textit{if} & Us \sqsubseteq S \\ Us \sqsubseteq V \,\hat{;}\, S & \textit{if} & Us = (Us', Us''),\ Us' \sqsubseteq V \ \textit{and}\ Us'' \sqsubseteq S \\ Us \sqsubseteq V \,;\, S & \textit{if} & Us = (Us', Us''),\ Us' \sqsubseteq V \ \textit{and}\ Us'' \sqsubseteq S \end{array}$$

With respect to our current example, we have that

$$(\lambda x\!:\!a.\,\lambda y\!:\!a.\,x \cdot \text{NIL},\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,y \cdot \text{NIL}) \sqsubseteq (x \cdot \text{NIL}); (y \cdot \text{NIL}); \text{NIL}.$$

$Us \sqsubset V$: The following definition extends the last relation to roots and inductively to arbitrary terms. Note that while the previous specifications could relate a term to itself, this is not possible here: $\sqsubset$ is strict.

$$\begin{array}{rll} Us \sqsubset H \cdot S & \textit{if} & Us \sqsubseteq S \\ Us \sqsubset \langle V_1, V_2\rangle & \textit{if} & Us = (Us_1, Us_2),\ Us_1 \sqsubset V_1 \ \textit{and}\ Us_2 \sqsubseteq V_2,\ \textit{or} \\ & & \qquad\qquad\qquad\quad Us_1 \sqsubseteq V_1 \ \textit{and}\ Us_2 \sqsubset V_2 \\ Us \sqsubset \hat{\lambda} x\!:\!A.\,V & \textit{if} & Us \sqsubset V \\ Us \sqsubset \lambda x\!:\!A.\,V & \textit{if} & Us \sqsubset V \end{array}$$

In relation with our example, we have

$$(\lambda x\!:\!a.\,\lambda y\!:\!a.\,x \cdot \text{NIL},\ \lambda x\!:\!a.\,\lambda y\!:\!a.\,y \cdot \text{NIL}) \sqsubset \lambda x\!:\!a.\,\lambda y\!:\!a.\,c \cdot (x \cdot \text{NIL}); (y \cdot \text{NIL}); \text{NIL}.$$

$Us \sqsubset Vs$: Finally, we extend this relation so that the right-hand side is a multiset of terms.

$$Us', Us'' \sqsubset V, Us'' \quad \textit{if} \quad Us' \sqsubset V$$

This definition allows us to complete the example presented at the beginning of the discussion. It is trivially obtained from our last relation by taking Us'' to be the empty multiset.

The ordering we will rely on in the proof of the completeness theorem is $Us \sqsubset Vs$. In order to do so, we must show that it is not possible to construct an infinite descending $\sqsubset$-chain at any multiset Us.

Lemma 3.4 (*Well-foundedness of* $\sqsubset$)

$Us \sqsubset Vs$ is a well-founded ordering.

Proof.

After proper generalization to take into account the several involved relations, this very simple proof proceeds by induction over the above definition. ☑

Derivation Ordering

The second ordering relation we need is among multisets of equality derivations obtained by applying a substitution to an equation system. Given systems Ξ_i, substitutions Θ_i and multiset derivations $\vec{\mathcal{E}_i}$ of $[\Theta_i]\Xi_i$, for $i = 1, 2$, we indicate this relation as $(\Xi_1, \Theta_1, \vec{\mathcal{E}_1}) \prec (\Xi_2, \Theta_2, \vec{\mathcal{E}_2})$. It is a variant of the usual multiset ordering constructed over the notion of subderivation.

We have the following formal derivation: $(\Xi_1, \Theta_1, \vec{\mathcal{E}_1}) \prec (\Xi_2, \Theta_2, \vec{\mathcal{E}_2})$ holds if and only if $\vec{\mathcal{E}_1} :: [\Theta_1]\Xi_1$, $\vec{\mathcal{E}_2} :: [\Theta_2]\Xi_2$, and any one of the following cases applies:

- $\vec{\mathcal{E}_1}$ is a submultiset of $\vec{\mathcal{E}_2}$.

- $\vec{\mathcal{E}}_1 = \vec{\mathcal{E}}'_1, \vec{\mathcal{E}}''_1$, where $\vec{\mathcal{E}}'_1 :: [\Theta_1]\Xi'_1$, $\vec{\mathcal{E}}''_1 :: [\Theta_1]\Xi''_1$ and $\Xi_1 = \Xi'_1, \Xi''_1$,
  $\vec{\mathcal{E}}_2 = \mathcal{E}'_2, \vec{\mathcal{E}}''_2$, where $\mathcal{E}'_2 :: [\Theta_2]\xi'_2$, $\vec{\mathcal{E}}''_2 :: [\Theta_2]\Xi''_2$ and $\Xi_2 = \xi'_2, \Xi''_2$,
  each $\mathcal{E}'_1$ in $\vec{\mathcal{E}}'_1$ is a subderivation of $\mathcal{E}_2$, and
  $(\Xi''_1, \Theta_1, \vec{\mathcal{E}}''_1) \prec (\Xi''_2, \Theta_2, \vec{\mathcal{E}}''_2)$.

- $\vec{\mathcal{E}}_1 = \mathcal{E}_1, \vec{\mathcal{E}}'_1$, where $\mathcal{E}_1 :: [\Theta_1]\xi_1$, $\xi_1 = (\Gamma; \Delta \vdash h \cdot S = F \cdot S' : a)$, $\vec{\mathcal{E}}'_1 :: [\Theta_1]\Xi'_1$ and $\Xi_1 = \xi_1, \Xi'_1$,
  $\vec{\mathcal{E}}_2 = \mathcal{E}_2, \vec{\mathcal{E}}'_2$, where $\mathcal{E}_2 :: [\Theta_2]\xi_2$, $\xi_2 = (\Gamma; \Delta \vdash F \cdot S' = h \cdot S : a)$, $\vec{\mathcal{E}}'_2 :: [\Theta_2]\Xi'_2$ and $\Xi_2 = \xi_2, \Xi'_2$,
  h is a rigid head,
  $(\Xi'_1, \Theta_1, \vec{\mathcal{E}}'_1) \prec (\Xi'_2, \Theta_2, \vec{\mathcal{E}}'_2)$.

- $\vec{\mathcal{E}}_1 = \mathcal{E}_1, \vec{\mathcal{E}}'_1$, where $\mathcal{E}_1 :: [\Theta_1]\xi_1$, $\xi_1 = (\Gamma; \Delta \vdash U \cdot S = H \cdot S' : a)$, $\vec{\mathcal{E}}'_1 :: [\Theta_1]\Xi'_1$ and $\Xi_1 = \xi_1, \Xi'_1$,
  $\vec{\mathcal{E}}_2 = \mathcal{E}_2, \vec{\mathcal{E}}'_2$, where $\mathcal{E}_2 :: [\Theta_2]\xi_2$, $\xi_2 = (\Gamma; \Delta \vdash \overline{U \cdot S} = H \cdot S' : a)$, $\vec{\mathcal{E}}'_2 :: [\Theta_2]\Xi'_2$ and $\Xi_2 = \xi_2, \Xi'_2$,
  $(\Xi'_1, \Theta_1, \vec{\mathcal{E}}'_1) \prec (\Xi'_2, \Theta_2, \vec{\mathcal{E}}'_2)$.

- $\vec{\mathcal{E}}_1 = \mathcal{E}_1, \vec{\mathcal{E}}'_1$, where $\mathcal{E}_1 :: [\Theta_1]\xi_1$, $\xi_1 = (\Gamma; \Delta \vdash H \cdot S' = U \cdot S : a)$, $\vec{\mathcal{E}}'_1 :: [\Theta_1]\Xi'_1$ and $\Xi_1 = \xi_1, \Xi'_1$,
  $\vec{\mathcal{E}}_2 = \mathcal{E}_2, \vec{\mathcal{E}}'_2$, where $\mathcal{E}_2 :: [\Theta_2]\xi_2$, $\xi_2 = (\Gamma; \Delta \vdash H \cdot S' = \overline{U \cdot S} : a)$, $\vec{\mathcal{E}}'_2 :: [\Theta_2]\Xi'_2$ and $\Xi_2 = \xi_2, \Xi'_2$,
  $(\Xi'_1, \Theta_1, \vec{\mathcal{E}}'_1) \prec (\Xi'_2, \Theta_2, \vec{\mathcal{E}}'_2)$.

The first two points of this definition correspond to the usual concept of multiset ordering, relative to the notion of subderivation. The third point specifies, roughly, that a flex-rigid equation is to be considered smaller than the symmetric rigid-flex equation. We interpret the last two points as indicating that weak head-reducing one of the sides of an equation yields a smaller equation.

This ordering is well-founded and therefore it is possible to base an inductive proof on it.

Lemma 3.5 (*Well-foundedness of* $\prec$)

$\prec$ *is a well-founded ordering.*

Proof.

This simple proof proceeds by induction on the above definition. ☑

3.4.3 Non-Deterministic Completeness

On the basis of the definitions given in the previous section, we will now state and prove that our pre-unification algorithm is non-deterministically complete with respect to the notion of equality discussed in Section 2.4. This task is not easy since we need to formulate proper lemmas for each of the judgments that are involved in the construction of an instantiating term for a logical variable. There are six such judgments and therefore we will need six auxiliary lemmas, each stated in a far more general form than necessary at the point where they are used.

Prior to doing so, we will need the following technical result according to which all logical variables appearing in an instantiating term U for a logical variable F are "new", i.e. different from every variable appearing in the equation system Ξ at hand or in the substitution constructed so far. For the sake of conciseness, our formalization does not keep an accurate account of the logical variables in use; it is straightforward to augment it with this information, but then tedious to carry around. Therefore, we will rely on the informal notion of "new" variable we just introduced.

Assumption 3.6 (*Freshness of substitution terms*)

i. If $\Gamma; \Delta \vdash a \hookrightarrow S, A$, *then every logical variable in* S *is "new";*

ii. If $\Gamma; \Delta \vdash A \hookrightarrow V$, *then every logical variable in* V *is "new";*

iii. If $\Gamma; \Delta \vdash B \Downarrow^{\iota} S' \hookrightarrow S$ *or* $\Gamma; \Delta \vdash A \Downarrow^{\pi} a \hookrightarrow S$, *then every logical variable occurring in* S *is "new";*

iv. If $\Gamma; \Delta \vdash U \,/\, A \Uparrow^{\iota} S \hookrightarrow V$ *or* $\Gamma; \Delta \vdash A \Uparrow^{\pi} S \hookrightarrow V$*, then every logical variable occurring in* V *is "new".* □

The validity of this fact is easily ascertained by inspection of the rules in Figures 9 and 10.

We start with the following lemma that characterizes the behavior of the spine variable raising judgment $\Gamma; \Delta \vdash a \hookrightarrow S, A$ defined in Figure 10. In its general form, it states that every well-typed term can be obtained from a redex whose spine part can be produced by means of that judgment and whose head is in the $=_{raise}$ relation with the original term.

Lemma 3.7 (*Spines in variable raising*)

If $\Gamma; \Delta \vdash_{\Sigma} U : a$ *with* U *canonical, then, for all contexts* Γ_1*,* Γ_2*,* Δ_1 *and* Δ_2 *such that* $(\Gamma_1, \Gamma_2); (\Delta_1, \Delta_2) = \Gamma; \Delta$*, there exist a type* A*, a canonical term* V *and a canonical spine* S *such that*

- $\Gamma_1; \Delta_1 \vdash_{\Sigma} V : A$,
- $\Gamma_2; \Delta_2 \vdash_{\Sigma} S : A > a$,
- $\Gamma_2; \Delta_2 \vdash a \hookrightarrow S, A$,
- $\mathrm{Can}(V \cdot S) = U$,
- $V =_{raise} U$.

Proof.

Given a partition $(\Gamma_1, \Gamma_2); (\Delta_1, \Delta_2)$ of the context $\Gamma; \Delta$, we proceed by induction on the structure of $\Gamma_2; \Delta_2$. There are three cases to consider:

$\Gamma_2 = \cdot$ and $\Delta_2 = \cdot$:

We have therefore that $\Gamma_1; \Delta_1 = \Gamma; \Delta$. Now, simply set $A = a$, $V = U$ and $S = \textsc{nil}$. Then,

- $\Gamma; \Delta \vdash_{\Sigma} U : a$ by assumption,
- $\cdot; \cdot \vdash_{\Sigma} \textsc{nil} : a > a$ by rule **lS_nil**,
- $\cdot; \cdot \vdash a \hookrightarrow \textsc{nil}, a$ by rule **raise_nil**,
- $\mathrm{Can}(U \cdot \textsc{nil}) = U$ by rule **Sr_nil** (notice that U must be a root), and
- $U =_{raise} U$ by definition of $\_ =_{raise} \_$.

This concludes this case of the proof.

$\Delta_2 = \Delta_2', x : B$, Γ_2 arbitrary:

By induction hypothesis, there are a type A', a canonical term V' and a canonical spine S' such that

- $\mathcal{U} :: \Gamma_1; \Delta_1, x : B \vdash_{\Sigma} V' : A'$,
- $\mathcal{S} :: \Gamma_2; \Delta_2' \vdash_{\Sigma} S' : A' > a$,
- $\mathcal{R} :: \Gamma_2; \Delta_2' \vdash a \hookrightarrow S', A'$,
- $\mathrm{Can}(V' \cdot S') = U$, and
- $V' =_{raise} U$.

We obtain the desired result by taking $A = B \multimap A'$, $V = \hat{\lambda} x : B. V'$ and $S = x_{\eta}^{B} \,\hat{;}\, S'$. Clearly, V is canonical, and so is S since both x_{η}^{B} and S' are. Moreover,

- $\Gamma_1; \Delta_1 \vdash_{\Sigma} \hat{\lambda} x : B. V' : B \multimap A'$ by rule **lS_llam** on $\mathcal{U}$,
- $\Gamma_2; \Delta_2', x : B \vdash_{\Sigma} (x_{\eta}^{B} \,\hat{;}\, S') : B \multimap A' > a$ by rule **lS_lapp** on $\mathcal{S}$ and a derivation of the judgment

$$\Gamma_2; x : B \vdash_{\Sigma} x_{\eta}^{B} : B,$$

which exists by virtue of Corollary 2.26 and weakening (Lemma 2.1).

- $\Gamma_2; \Delta'_2, x:B \vdash a \hookrightarrow x^B_\eta \mathbin{\hat{;}} S', B \multimap A'$ by rule **raise_lapp**,
- $\mathrm{Can}((\hat{\lambda} x:B.\, V') \cdot (x^B_\eta \mathbin{\hat{;}} S')) = \mathrm{Can}([x^B_\eta/x]V' \cdot S') = \mathrm{Can}(V' \cdot S') = U$ where the second step makes use of Corollary 2.28, and
- $\hat{\lambda} x:B.\, V' =_{raise} U$ since $V' =_{raise} U$.

$\Gamma_2 = \Gamma'_2, x:B,\ \Delta_2$ arbitrary:

We proceed as in the previous case. ☑

In the sequel, we will always use the special instance of the previous lemma obtained by choosing Γ_1 and Δ_1 to be the empty context, as expressed by the following corollary.

Corollary 3.8 (*Spines in variable raising*)

If $\Gamma; \Delta \vdash_\Sigma U : a$ with U canonical, then there exist a type A, a canonical term V and a canonical spine S such that

- $\cdot; \cdot \vdash_\Sigma V : A$,
- $\Gamma; \Delta \vdash_\Sigma S : A > a$,
- $\Gamma; \Delta \vdash a \hookrightarrow S, A$,
- $\mathrm{Can}(V \cdot S) = U$,
- $V =_{raise} U$.

Proof.

A proof of this corollary is obtained from the previous lemma by choosing $\Gamma_1 = \Delta_1 = \cdot$, $\Gamma_2 = \Gamma$ and $\Delta_2 = \Delta$. ☑

This corollary is used only in the following lemma that highlights aspects of the behavior of the variable raising judgment $\Gamma; \Delta \vdash A \hookrightarrow V$, defined in Figure 10. In particular, the substitution Θ it postulates is only defined on the ("new") variables in the term V.

Lemma 3.9 (*Variable raising*)

If $\Gamma; \Delta \vdash_\Sigma U : A$ with U canonical, then there is a canonical term V and a canonical substitution Θ such that

- $\Gamma; \Delta \vdash A \hookrightarrow V$,
- $\Gamma; \Delta \vdash_\Sigma [\Theta]V = U : A$,
- $\mathrm{Im}(\Theta) \sqsubseteq U$.

Proof.

The proof proceeds by induction on the structure of A. We will analyze the most significant cases. Recall that we requires the domain of a substitution to be disjoint from its range.

$A = a$: By the previous corollary, there exist a type B, a canonical term U' and a canonical spine S such that

- $\cdot; \cdot \vdash_\Sigma U' : B$,
- $\Gamma; \Delta \vdash_\Sigma S : B > a$,
- $\Gamma; \Delta \vdash a \hookrightarrow S, B$,
- $\mathrm{Can}(U' \cdot S) = U$,
- $U' =_{raise} U$.

Let $\Theta = U'/F$ and $V = F \cdot S$. Both Θ and V are clearly canonical. Then,

- $\Gamma; \Delta \vdash a \hookrightarrow V$ by rule **raise_root**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta]V = U : a$ by the completeness of staged equality (Theorem 2.18) since $[\Theta]V = [U'/F](F \cdot S) = U' \cdot S$ and $\mathrm{Can}(U' \cdot S) = U$.
- $\mathrm{Im}(\Theta) \sqsubseteq U$ by definition of $\sqsubseteq$ since $\mathrm{Im}(\Theta) = U'$ and $U' =_{raise} U$.

$\boldsymbol{A = \top}$**:** By inversion on the typing rules, we have that $U = \langle\rangle$. Set $\Theta = \cdot$ and $V = \langle\rangle$. Indeed,

- $\Gamma; \Delta \vdash \top \hookrightarrow \langle\rangle$ by rule **raise_unit**.
- $\Gamma; \Delta \vdash_\Sigma \langle\rangle = \langle\rangle : a$ by rules **Seq_unit**.
- $\cdot \sqsubseteq \langle\rangle$ by definition of $\sqsubseteq$.

$\boldsymbol{A = A_1 \,\&\, A_2}$**:** Then, by inversion on rule **lS_pair**, $U = \langle U_1, U_2\rangle$ and, for $i = 1, 2$, $\Gamma; \Delta \vdash_\Sigma U_i : A_i$ is derivable and U_i is canonical. By induction hypothesis, there are canonical terms V_i and canonical substitutions Θ_i such that

- $\Gamma; \Delta \vdash A_i \hookrightarrow V_i$,
- $\Gamma; \Delta \vdash_\Sigma [\Theta_i]V_i = U_i : A_i$, and
- $\mathrm{Im}(\Theta_i) \sqsubseteq U_i$.

By Assumption 3.6, we have that V_i, and therefore Θ_i, mention distinct logical variables. Thus, we can form the substitution (Θ_1, Θ_2) without violating the requirement that the domain and the range of a substitution be disjoint. Moreover, $[\Theta_1, \Theta_2]V_i = [\Theta_i]V_i$ and $\mathrm{Im}(\Theta_1, \Theta_2) = (\mathrm{Im}(\Theta_1), \mathrm{Im}(\Theta_2))$. Then the term $\langle V_1, V_2\rangle$ and the substitution Θ_1, Θ_2 are canonical, and moreover

- $\Gamma; \Delta \vdash A_1 \,\&\, A_2 \hookrightarrow \langle V_1, V_2\rangle$ by rule **raise_pair**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta_1, \Theta_2]\langle V_1, V_2\rangle = \langle U_1, U_2\rangle : A_1 \,\&\, A_2$, since $[\Theta_1, \Theta_2]\langle V_1, V_2\rangle = \langle [\Theta_1, \Theta_2]V_1, [\Theta_1, \Theta_2]V_2\rangle = \langle [\Theta_1]V_1, [\Theta_2]V_2\rangle$.
- $\mathrm{Im}(\Theta_1, \Theta_2) = (\mathrm{Im}(\Theta_1), \mathrm{Im}(\Theta_2)) \sqsubseteq \langle U_1, U_2\rangle$ by definition.

$\boldsymbol{A = A_1 \multimap A_2}$**:** Then, by inversion, $U = \hat\lambda x : A_1.\, U'$ for U' canonical, and $\Gamma; \Delta, x : A_1 \vdash_\Sigma U' : A_2$ is derivable. By induction hypothesis, there is a canonical term V' and a canonical substitution Θ such that

- $\Gamma; \Delta, x{:}A_1 \vdash A_2 \hookrightarrow V'$,
- $\Gamma; \Delta, x{:}A_1 \vdash_\Sigma [\Theta]V' = U' : A_2$, and
- $\mathrm{Im}(\Theta) \sqsubseteq U'$.

Then $\hat\lambda x{:}A_1.\, V'$ is canonical and

- $\Gamma; \Delta \vdash A_1 \multimap A_2 \hookrightarrow \hat\lambda x{:}A_1.\, V'$ by rule **raise_llam**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta](\hat\lambda x{:}A_1.\, V') = \hat\lambda x{:}A_1.\, U' : A_1 \multimap A_2$ by rule **Seq_llam** and definition of substitution application.
- $\mathrm{Im}(\Theta) \sqsubseteq \hat\lambda x{:}A_1.\, U'$ by definition.

$\boldsymbol{A = A_1 \to A_2}$**:** The proof proceeds similarly to the previous case. ☑

We will now consider the judgments having the function of building the terms and spines of an instantiating term obtained by projection and imitation. These four judgments were defined in Figure 9 and rely on the variable raising judgments. The corresponding lemmas will use the result we just obtained.

We begin with a characterization of the judgment $\Gamma; \Delta \vdash A \downarrow^\pi a \hookrightarrow S_o$ that builds the spine S_o of an instantiating term obtained by projection.

Lemma 3.10 (*Spines in projection*)

If $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$ *for* S *canonical, then there is a canonical spine* S_o *and a canonical substitution* Θ *such that*

- $\Gamma; \Delta \vdash A \downarrow^\pi a \hookrightarrow S_o$,

- $\Gamma; \Delta \vdash_\Sigma [\Theta]S_o = S : A > a$,
- $\mathrm{Im}(\Theta) \sqsubseteq S$.

Proof.

This proof proceeds by induction over the structure of the type A, or, equivalently, of the derivation $\mathcal{S}$. We have the following cases depending on the last rule applied in $\mathcal{S}$:

lS_nil: Then, by inversion,

$$\mathcal{S} = \frac{}{\Gamma; \cdot \vdash_\Sigma \mathrm{NIL} : a > a}\,\mathbf{lS\_nil}$$

with $A = a$, $S = \mathrm{NIL}$ and $\Delta = \cdot$.

Then, take $S_o = \mathrm{NIL}$ and $\Theta = \cdot$, which are trivially canonical. Thus

- $\Gamma; \cdot \vdash a \downarrow^\pi a \hookrightarrow \mathrm{NIL}$ by rule **frp_nil**.
- $\Gamma; \cdot \vdash_\Sigma [\Theta]\mathrm{NIL} = \mathrm{NIL} : a > a$ by rule **Seq_nil**.
- $\cdot \sqsubseteq \mathrm{NIL}$ by definition of $\sqsubseteq$.

lS_fst: We have that

$$\mathcal{S} = \frac{\begin{array}{c}\mathcal{S}'\\ \Gamma; \Delta \vdash_\Sigma S' : A_1 > a\end{array}}{\Gamma; \Delta \vdash_\Sigma \pi_1\, S' : A_1 \,\&\, A_2 > a}\,\mathbf{lS\_fst}$$

with $A = A_1 \,\&\, A_2$ and $S = \pi_1\, S'$ for S' canonical.

By induction hypothesis on $\mathcal{S}'$, there is a canonical spine S'_o and a canonical substitution Θ such that $\Gamma; \Delta \vdash A_1 \downarrow^\pi a \hookrightarrow S'_o$ and $\Gamma; \Delta \vdash_\Sigma [\Theta]S'_o = S' : A_1 > a$ are derivable, and $\mathrm{Im}(\Theta) \sqsubseteq S'$. Then,

- $\Gamma; \Delta \vdash A_1 \,\&\, A_2 \downarrow^\pi a \hookrightarrow \pi_1\, S'_o$ by rule **frp_fst**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta](\pi_1\, S'_o) = \pi_1\, S' : A_1 \,\&\, A_2 > a$ by rule **Seq_fst** and the definition of substitution application.
- $\mathrm{Im}(\Theta) \sqsubseteq \pi_1\, S'$ by definition.

Observe that $\pi_1\, S'_o$ is canonical since S'_o is.

lS_snd: We reason symmetrically.

lS_lapp: By inversion, we have that

$$\mathcal{S} = \frac{\begin{array}{cc}\mathcal{U} & \mathcal{S}'\\ \Gamma; \Delta' \vdash_\Sigma U : A_1 & \Gamma; \Delta'' \vdash_\Sigma S' : A_2 > a\end{array}}{\Gamma; \Delta', \Delta'' \vdash_\Sigma U \,\hat{;}\, S' : A_1 \multimap A_2 > a}\,\mathbf{lS\_lapp}$$

where $A = A_1 \multimap A_2$, $S = U \,\hat{;}\, S'$ and $\Delta = \Delta', \Delta''$. Both U and S' are canonical.

By the variable raising lemma 3.9 applied to $\mathcal{U}$, there are a canonical term V and a canonical substitution Θ' such that $\Gamma; \Delta' \vdash A_1 \hookrightarrow V$, $\Gamma; \Delta' \vdash_\Sigma [\Theta']V = U : A_1$ and $\mathrm{Im}(\Theta') \sqsubseteq U$.

By induction hypothesis on $\mathcal{S}'$, there are a canonical spine S'_o and a canonical substitution Θ'' such that $\Gamma; \Delta'' \vdash A_2 \downarrow^\pi a \hookrightarrow S'_o$ and $\Gamma; \Delta'' \vdash_\Sigma [\Theta'']S'_o = S' : A_2 > a$ are derivable, and $\mathrm{Im}(\Theta'') \sqsubseteq S'_o$.

By Assumption 3.6, we have that V and S'_o (or equivalently Θ' and Θ'') mention distinct logical variables. Therefore, the domain and the range of the substitution (Θ', Θ'') are disjoint and, moreover, $([\Theta']V \,\hat{;}\, [\Theta'']S'_o) = [\Theta', \Theta''](V \,\hat{;}\, S'_o)$ and $\mathrm{Im}(\Theta', \Theta'') = (\mathrm{Im}(\Theta'), \mathrm{Im}(\Theta''))$. Then,

- $\Gamma; \Delta', \Delta'' \vdash A_1 \multimap A_2 \downarrow^\pi a \hookrightarrow V \,\hat{;}\, S'_o$ by rule **frp_lapp**.
- $\Gamma; \Delta', \Delta'' \vdash_\Sigma [\Theta']V \,\hat{;}\, [\Theta'']S'_o = U \,\hat{;}\, S' : A_1 \multimap A_2 > a$ by rule **Seq_lapp**.
- $\mathrm{Im}(\Theta', \Theta'') = (\mathrm{Im}(\Theta'), \mathrm{Im}(\Theta'')) \sqsubseteq U \,\hat{;}\, S'$ by definition.

Moreover, $V\,\hat{;}\,S'_o$ and (Θ', Θ'') are canonical.

lS_iapp: This part of the proof is similar to the previous case. ☑

On the basis of this result, we have the following lemma which describes how an instantiating term V for a logical variable F is obtained. Observe that this property postulates the validity of an instance of the strict relation $\sqsubset$, while the previous lemma made use of the non-strict form $\sqsubseteq$.

Lemma 3.11 (*Projection*)

If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$ for U canonical, $S \div A$ and $H_S(U) = x$, then there exist a canonical term V and a canonical substitution Θ such that

- $\Gamma; \Delta \vdash A \Uparrow^\pi S \hookrightarrow V$,
- $\Gamma; \Delta \vdash_\Sigma [\Theta]V = U : A$,
- $\mathrm{Im}(\Theta) \sqsubset U$.

Proof.

The proof proceeds by induction on the type A and inversion on the structure of S.

$\boldsymbol{S = \textsc{nil}}$**:** Then, by definition of $\div$, we have that $A = a$.

By inversion on $\mathcal{U}$, we obtain that $U = H \cdot S'$. Since U is in canonical form and $H_S(U) = x$, we have that $H = x$. The parameter x can be either linear or intuitionistic: this gives rise to two subcases.

$\boldsymbol{x : B}$ **in** $\boldsymbol{\Delta}$**:** By inversion on rule **lS_lvar**, there is a derivation of $\Gamma; \Delta' \vdash_\Sigma S' : B > a$, where $\Delta = \Delta', x : B$. By the previous lemma, there are a canonical spine S_o and a canonical substitution Θ such that $\Gamma; \Delta' \vdash B \downarrow^\pi a \hookrightarrow S_o$, $\Gamma; \Delta' \vdash_\Sigma [\Theta]S_o = S' : B > a$ and $\mathrm{Im}(\Theta) \sqsubseteq S'$.

Then, for $V = x \cdot S_o$, which is certainly a canonical term, we can therefore conclude that

- $\Gamma; \Delta', x : B \vdash a \Uparrow^\pi \textsc{nil} \hookrightarrow x \cdot S_o$ by rule **frp_lvar**.
- $\Gamma; \Delta', x : B \vdash_\Sigma [\Theta](x \cdot S_o) = x \cdot S' : a$ by rule **Seq_lvar**.
- $\mathrm{Im}(\Theta) \sqsubset x \cdot S'$, by definition.

$\boldsymbol{x : B}$ **in** $\boldsymbol{\Gamma}$**:** Similar.

$\boldsymbol{S = \pi_1\, S'}$**:** By definition of $\div$, we have that $A = A_1 \,\&\, A_2$ and $S' \div A_1$.

By inversion on rule **lS_pair**, we also have that $U = \langle U_1, U_2 \rangle$ and $\mathcal{U}_i :: \Gamma; \Delta \vdash_\Sigma U_i : A_i$, for $i = 1, 2$. Clearly, since U is canonical, so are U_1 and U_2. Moreover, by definition of relative head, we have that $H_S(U) = H_{S'}(U_1)$.

Then, by induction hypothesis on A_1, there are a canonical term V_1 and a canonical substitution Θ_1 such that $\Gamma; \Delta \vdash A_1 \Uparrow^\pi S' \hookrightarrow V_1$, $\Gamma; \Delta \vdash_\Sigma [\Theta_1]V_1 = U_1 : A_1$ and $\mathrm{Im}(\Theta_1) \sqsubset U_1$.

By the variable raising lemma 3.9 applied to $\mathcal{U}_2$, there are a canonical term V_2 and a canonical substitution Θ_2 such that $\Gamma; \Delta \vdash A_2 \hookrightarrow V_2$, $\Gamma; \Delta \vdash_\Sigma [\Theta_2]V_2 = U_2 : A_2$ and $\mathrm{Im}(\Theta_2) \sqsubseteq U_2$.

By Assumption 3.6, we have that V_1 and V_2, and consequently Θ_1 and Θ_2, do not to have logical variables in common. Therefore, the substitution (Θ_1, Θ_2) satisfies our disjointness requirement and, moreover, $[\Theta_1, \Theta_2]V_i = [\Theta_i]V_i$ and $\mathrm{Im}(\Theta_1, \Theta_2) = (\mathrm{Im}(\Theta_1), \mathrm{Im}(\Theta_2))$. A consequence of this fact is that (Θ_1, Θ_2) is canonical. $\langle V_1, V_2 \rangle$ is canonical as well, since both components are. Moreover,

- $\Gamma; \Delta \vdash A_1 \,\&\, A_2 \Uparrow^\pi \pi_1\, S' \hookrightarrow \langle V_1, V_2 \rangle$ by rule **frp_pair1**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta_1, \Theta_2]\langle V_1, V_2 \rangle = \langle U_1, U_2 \rangle : A_1 \,\&\, A_2$ by rule **Seq_pair**.
- $\mathrm{Im}(\Theta_1, \Theta_2) = (\mathrm{Im}(\Theta_1), \mathrm{Im}(\Theta_2)) \sqsubset \langle U_1, U_2 \rangle$ since $\mathrm{Im}(\Theta_1) \sqsubset U_1$ and $\mathrm{Im}(\Theta_2) \sqsubseteq U_2$.

$\boldsymbol{S = \pi_2\, S'}$**:** We proceed symmetrically to the previous case.

$\boldsymbol{S = U' \,\hat{;}\, S'}$**:** We have that $A = A_1 \multimap A_2$ and $S' \div A_2$.

By inversion on rule **lS_llam**, $U = \hat{\lambda} x : A_1 .\, U''$ and $\Gamma; \Delta, x : A_1 \vdash_\Sigma U'' : A_2$. Clearly, U'' is canonical.

By induction hypothesis on A_2, there are a canonical term V' and a canonical substitution Θ such that $\Gamma; \Delta, x : A_1 \vdash A_2 \Uparrow^\pi S' \hookrightarrow V'$, $\Gamma; \Delta, x : A_1 \vdash_\Sigma [\Theta]V' = U'' : A_2$ and $\mathrm{Im}(\Theta) \sqsubset U''$. Then, $\hat{\lambda} x : A_1 .\, V'$ is canonical and

- $\Gamma; \Delta \vdash A_1 \multimap A_2 \Uparrow^\pi U' \,\hat{;}\, S' \hookrightarrow \hat{\lambda} x : A_1 .\, V'$ by rule **frp_llam**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta](\hat{\lambda} x : A_1 .\, V') = \hat{\lambda} x : A_1 .\, U'' : A_1 \multimap A_2$ by rule **Seq_llam**.
- $\mathrm{Im}(\Theta) \sqsubset \hat{\lambda} x : A_1 .\, U''$ by definition.

$\boldsymbol{S = U'; S'}$**:** We proceed as in the previous case. ☑

Similar results hold for the imitation judgments and the same observations apply. The premisses of the lemmas below are slightly more complicated than in the case of projection since the imitation judgments mention more information. However, this does not add complexity to the proofs.

Lemma 3.12 (*Spines in imitation*)

If $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$ *for* S *canonical and* $S \sim \hat{S}$, *then there is a canonical spine* S_o *and a canonical substitution* Θ *such that*

- $\Gamma; \Delta \vdash A \Downarrow^\iota \hat{S} \hookrightarrow S_o$,
- $\Gamma; \Delta \vdash_\Sigma [\Theta]S_o = S : A > a$,
- $\mathrm{Im}(\Theta) \sqsubseteq S$.

Proof.

We proceed by induction on the structure of A in very similar fashion to the way we handled the proof of the analogous result in the case of projection (Lemma 3.10). The major difference is manifested by the treatment of the conjunctive cases. We illustrate this point by carrying out the proof in the case $\mathcal{S}$ ends in rule **lS_fst**.

lS_fst: We have

$$\mathcal{S} = \cfrac{\begin{array}{c}\mathcal{S}' \\ \Gamma; \Delta \vdash_\Sigma S' : A_1 > a\end{array}}{\Gamma; \Delta \vdash_\Sigma \pi_1 S' : A_1 \,\&\, A_2 > a}\,\textbf{lS\_fst}$$

with $A = A_1 \,\&\, A_2$ and $S = \pi_1 S'$ for S' canonical.

Since $S \sim \hat{S}$, we have that $\hat{S} = \pi_1 \hat{S}'$ and $S' \sim \hat{S}'$. We can therefore apply the induction hypothesis on A_1. We obtain that there are a canonical spine S'_o and a canonical substitution Θ such that $\Gamma; \Delta \vdash A_1 \Downarrow^\iota \hat{S}' \hookrightarrow S'_o$ and $\Gamma; \Delta \vdash_\Sigma [\Theta]S'_o = S' : A_1 > a$ are derivable, and $\mathrm{Im}(\Theta) \sqsubseteq S'_o$. Then,

- $\Gamma; \Delta \vdash A_1 \,\&\, A_2 \Downarrow^\iota \pi_1 \hat{S}' \hookrightarrow \pi_1 S'_o$ by rule **fri_fst**.
- $\Gamma; \Delta \vdash_\Sigma [\Theta](\pi_1 S'_o) = \pi_1 S' : A_1 \,\&\, A_2 > a$ by rule **Seq_fst** and the definition of substitution application.
- $\mathrm{Im}(\Theta) \sqsubseteq \pi_1 S'$ by definition.

Clearly, $\pi_1 S'_o$ is canonical. ☑

The above result is used in the following lemma. It describes the properties of the judgment $\Gamma; \Delta \vdash c \cdot \hat{S} \,/\, A \Uparrow^\iota S \hookrightarrow V$ which constructs an instantiating term V by imitation. Recall that, by the relative heads lemma 3.3, $H_S(U) = c$ entails that $\mathrm{Can}(U \cdot S) = c \cdot S^*$ for some canonical spine S^*, but the opposite implication does not hold. Therefore we need both premisses in the property below in order to expose S^*.

Lemma 3.13 (*Imitation*)

If $\mathcal{U} :: \Gamma;\Delta \vdash_\Sigma U : A$ *for* U *canonical,* $S \div A$*,* $H_S(U) = c$ *for* $c{:}B$ *in* Σ*,* $\mathrm{Can}(U \cdot S) = c \cdot S^*$*, and* $S^* \sim \hat{S}$*, then there exist a canonical term* V *and a canonical substitution* Θ *such that*

- $\Gamma;\Delta \vdash c \cdot \hat{S} \,/\, A \Uparrow^{\iota} S \hookrightarrow V$,
- $\Gamma;\Delta \vdash_\Sigma [\Theta]V = U : A$,
- $\mathrm{Im}(\Theta) \sqsubset U$.

Proof.

This proof is conducted similarly to the case of projection we analyzed in Lemma 3.11, i.e. by induction on the type A and inversion on the structure of S. The main difference appears in the base case, i.e. when $S = \text{NIL}$. We will analyze this case only.

$\boldsymbol{S} = \text{NIL}$**:** By definition of $\div$, we have that $A = a$.

By rule **Sr_nil** $\mathrm{Can}(U \cdot \text{NIL}) = U = c \cdot S^*$ for $c{:}B$ in Σ and some spine S^* (by inversion, U must be a root). By inversion on rule **lS_con**, $\mathcal{S}^* :: \Gamma;\Delta \vdash_\Sigma S^* : B > a$ is derivable. Moreover, S^* is canonical.

By the previous lemma applied to $\mathcal{S}^*$, there are a canonical spine S_o and a canonical substitution Θ such that $\Gamma;\Delta \vdash a \Downarrow^{\iota} \hat{S} \hookrightarrow S_o$, $\Gamma;\Delta \vdash_\Sigma [\Theta]S_o = S^* : B > a$ and $\mathrm{Im}(\Theta) \sqsubseteq S^*$.

We obtain the desired conclusion by the following observations:

- $\Gamma;\Delta \vdash c \cdot \hat{S} \,/\, a \Uparrow^{\iota} \text{NIL} \hookrightarrow c \cdot S_o$ by rule **fri_con**.
- $\Gamma;\Delta \vdash_\Sigma c \cdot [\Theta]S_o = c \cdot S^* : A$ by rule **Seq_con**, from which we get $\Gamma;\Delta \vdash_\Sigma c \cdot [\Theta]S_o = U : A$ by rule **Seq_redex_r**.
- $\mathrm{Im}(\Theta) \sqsubset c \cdot S^*$ by definition.

Moreover, $c \cdot S'_o$ is canonical. ☑

With the help of the various properties we just proved, we can tackle the proof of the non-deterministic completeness of our linear pre-unification algorithm with respect to the notion of staged equality defined in Figure 5, and therefore, by Theorem 2.18, with respect to definitional equality for $S^{\to\multimap\&\top}$. This result is expressed in the following theorem.

Theorem 3.14 (*Completeness of linear pre-unification*)

Given a system of well-typed equations Ξ *and a well-typed canonical substitution* Θ *such that* $\vec{\mathcal{E}} :: [\Theta]\Xi$*, there are substitutions* Θ_{ff} *and* Θ'*, and a system of flex-flex equations* Ξ_{ff} *such that*

- $\Theta = \mathrm{Can}(\Theta_{\mathit{ff}} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{\mathit{ff}} :: [\Theta_{\mathit{ff}}]\Xi_{\mathit{ff}}$*, and*
- $\mathcal{X} :: \Xi \setminus \Xi_{\mathit{ff}}, \Theta'$.

Proof.

We prove this theorem by nested induction on the image of Θ considered relative to the well-founded ordering $\sqsubset$ and on the triple $(\Xi, \Theta, \vec{\mathcal{E}})$ relative to the well-founded ordering $\prec$; both orderings were defined in the previous section. Therefore, we allow ourselves to appeal to the induction hypothesis every time we are considering a situation characterized by a system of equations Ξ', a substitution Θ' and a multiset of derivations $\vec{\mathcal{E}}'$ such that

1. $\mathrm{Im}(\Theta') \sqsubset \mathrm{Im}(\Theta)$, $\vec{\mathcal{E}}'$ and $\vec{\mathcal{E}}$ are arbitrarily related and so are Ξ' and Ξ, or
2. $\Theta' = \Theta$, but $(\Xi', \Theta, \vec{\mathcal{E}}') \prec (\Xi, \Theta, \vec{\mathcal{E}})$.

We distinguish the following (non-exclusive) cases based on the contents of Ξ.

$\Xi = \Xi'_{ff}$: Ξ consists only of flex-flex equations.

Simply take $\Theta_{ff} = \Theta$, $\Theta' = \cdot$ and $\Xi_{ff} = \Xi$. We obtain the desired result as follows:

- $\Theta = \mathrm{Can}(\Theta \circ \cdot)_{|\,\mathrm{dom}(\Theta)}$ since $\Theta \circ \cdot = \Theta$, and $\Theta_{|\,\mathrm{dom}(\Theta)} = \Theta$ and moreover Θ has been assumed canonical.
- Use $\vec{\mathcal{E}}$ as $\vec{\mathcal{E}}_{ff}$.
- $\Xi_{ff} \setminus \Xi_{ff}, \cdot$ by rule **pu_ff**.

$\Xi = \Xi', \xi$ with $\xi = (\Gamma; \Delta \vdash S_1 = S_2 : A > a)$: Ξ contains a spine equation. We further distinguish cases on the structure of the type A. We analyze three representative situations. The remaining cases are handled similarly. Let $\mathcal{E}$ be the assumed derivation of $[\Theta]\xi$ and $\vec{\mathcal{E}}' :: [\Theta]\Xi'$, so that $\vec{\mathcal{E}} = \vec{\mathcal{E}}', \mathcal{E}$.

$A = a'$: By inversion on rule **Seq_nil**, we have

$$\mathcal{E} = \frac{}{\Gamma; \cdot \vdash_\Sigma \mathrm{NIL} = \mathrm{NIL} : a > a}\,\textbf{Seq\_nil}$$

where $[\Theta]S_1 = [\Theta]S_2 = \mathrm{NIL}$, $a' = a$ and $\Delta = \cdot$.

We can apply the induction hypothesis on Ξ' and Θ since $\vec{\mathcal{E}}'$ is a submultiset of $\vec{\mathcal{E}}$ and therefore $(\Xi', \Theta, \vec{\mathcal{E}}') \prec (\Xi, \Theta, \vec{\mathcal{E}})$. Thus, we deduce that there are substitutions Θ_{ff} and Θ' and a system of flex-flex equations Ξ_{ff} such that

- $\Theta = \mathrm{Can}(\Theta_{ff} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{ff} :: [\Theta_{ff}]\Xi_{ff}$ and
- $\mathcal{X}' :: \Xi' \setminus \Xi_{ff}, \Theta'$.

In order to conclude this case, simply apply rule **pu_nil** to $\mathcal{X}'$ to obtain the desired derivation $\mathcal{X}$ of $(\Xi', \xi) \setminus \Xi_{ff}, \Theta'$.

$A = A_1 \,\&\, A_2$: By inversion, there are two subcases to consider: either $\mathcal{E}$ ends in rule **Seq_fst**, or in rule **Seq_snd**. We will examine the first of these alternatives. The second is handled similarly. By definition of substitution application, we have

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}' \\ \Gamma; \Delta \vdash_\Sigma [\Theta]S'_1 = [\Theta]S'_2 : A_1 > a\end{array}}{\Gamma; \Delta \vdash_\Sigma [\Theta](\pi_1\, S'_1) = [\Theta](\pi_1\, S'_2) : A_1 \,\&\, A_2 > a}\,\textbf{Seq\_fst}$$

with $S_1 = \pi_1\, S'_1$ and $S_2 = \pi_1\, S'_2$. Let $\xi' = (\Gamma; \Delta \vdash S'_1 = S'_2 : A_1 > a)$.

By definition, $((\Xi', \xi'), \Theta, (\vec{\mathcal{E}}', \mathcal{E}')) \prec (\Xi, \Theta, \vec{\mathcal{E}})$. By induction hypothesis on Θ and (Ξ', ξ'), there are Θ', Θ_{ff} and Ξ_{ff} such that

- $\Theta = \mathrm{Can}(\Theta_{ff} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{ff} :: [\Theta_{ff}]\Xi_{ff}$ and
- $\mathcal{X}' :: \Xi', \xi' \setminus \Xi_{ff}, \Theta'$.

The derivation $\mathcal{X}$ is constructed by applying rule **pu_fst** to $\mathcal{X}'$:

$$\frac{\Xi', (\Gamma; \Delta \vdash S'_1 = S'_2 : A_1 > a) \setminus \Xi_{ff}, \Theta_{ff}}{\Xi', (\Gamma; \Delta \vdash \pi_1\, S'_1 = \pi_1\, S'_2 : A_1 \,\&\, A_2 > a) \setminus \Xi_{ff}, \Theta_{ff}}\,\textbf{pu\_fst}.$$

$A = A_1 \multimap A_2$: By inversion, we have that

$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}' \\ \Gamma; \Delta' \vdash_\Sigma [\Theta]U_1 = [\Theta]U_2 : A_1\end{array} \quad \begin{array}{c}\mathcal{E}'' \\ \Gamma; \Delta'' \vdash_\Sigma [\Theta]S'_1 = [\Theta]S'_2 : A_2 > a\end{array}}{\Gamma; \Delta', \Delta'' \vdash_\Sigma [\Theta](U_1 \,\hat{;}\, S'_1) = [\Theta](\pi_1\, U_2 S'_2) : A_1 \multimap A_2 > a}\,\textbf{Seq\_lapp}$$

with $\Delta = \Delta', \Delta''$, $S_1 = U_1 \mathbin{\hat{;}} S_1'$ and $S_2 = U_2 \mathbin{\hat{;}} S_2'$. Let $\xi' = (\Gamma; \Delta' \vdash U_1 = U_2 : A_1)$ and $\xi'' = (\Gamma; \Delta'' \vdash S_1' = S_2' : A_1 > a)$.

By definition, $((\Xi', \xi', \xi''), \Theta, (\vec{\mathcal{E}}', \mathcal{E}')) \prec (\Xi, \Theta, \vec{\mathcal{E}})$. By induction hypothesis, there are Θ', Θ_{ff} and Ξ_{ff} such that

- $\Theta = \mathrm{Can}(\Theta_{ff} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{ff} :: [\Theta_{ff}]\Xi_{ff}$ and
- $\mathcal{X}' :: \Xi', \xi', \xi'' \setminus \Xi_{ff}, \Theta'$.

The required derivation $\mathcal{X}$ is then obtained by applying rule **pu_lapp** to $\mathcal{X}'$.

$\Xi = \Xi, \xi$ with $\xi = (\Gamma; \Delta \vdash U_1 = U_2 : A)$: Ξ contains a term equation that is not flex-flex. Again, we proceed by cases on the structure of A. The situations in which A is not a base type are handled similarly to the case of spines above. We will not go into further details. More interesting are the cases where A is some base type a.

By inversion, $U_i = H_i \cdot S_i$ for $i = 1, 2$. We will distinguish cases on the nature of the heads H_1 and H_2. We first consider the situations where either or both are terms, so that U_1 or U_2 is a redex. Once these cases are taken care of, H_i can be either a constant, a parameter or a logical variable. Then, we distinguish three cases depending on whether H_1 and H_2 are rigid or flexible heads (by assumption, H_1 and H_2 cannot be both flexible).

Again, we will indicate with $\mathcal{E}$ and $\vec{\mathcal{E}}'$ the assumed derivations of $[\Theta]\xi$ and $[\Theta]\Xi'$, respectively. We have that $\vec{\mathcal{E}} = \vec{\mathcal{E}}', \mathcal{E}$.

Redex-redex: Let $H_1 = V_1$ and $H_2 = V_2$. By inversion on the structure of $\mathcal{E}$, this derivation can end either in rule **Seq_redex_l** or **Seq_redex_r**. We will assume that the first of these rules is used. The other alternative is treated symmetrically. Therefore,

$$\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}' \\ \Gamma; \Delta \vdash_\Sigma \overline{[\Theta](V_1 \cdot S_1)} = [\Theta](H_2 \cdot S_2) : a\end{array}}{\Gamma; \Delta \vdash_\Sigma [\Theta](V_1 \cdot S_1) = [\Theta](H_2 \cdot S_2) : a}\,\textbf{Seq\_redex\_l}.$$

Given a generic term U and substitution Θ, an easy induction on the structure of U suffices to show that $\overline{[\Theta]U} = \overline{[\Theta]\overline{U}}$. Thus, $\mathcal{E}'$ is also a derivation of $\Gamma; \Delta \vdash_\Sigma \overline{[\Theta](\overline{V_1 \cdot S_1})} = [\Theta](H_2 \cdot S_2) : a$. Therefore, by rule **Seq_redex_l**, there is a derivation $\mathcal{E}''$ of

$$\Gamma; \Delta \vdash_\Sigma [\Theta](\overline{V_1 \cdot S_1}) = [\Theta](H_2 \cdot S_2) : a.$$

Let $\xi'' = (\Gamma; \Delta \vdash \overline{V_1 \cdot S_1} = H_2 \cdot S_2 : a)$.

By the definition of $\prec$ from Section 3.1, we have that $((\Xi', \xi''), \Theta, (\vec{\mathcal{E}}', \mathcal{E}'')) \prec (\Xi, \Theta, \vec{\mathcal{E}})$. Therefore, we can apply the induction hypothesis, obtaining that there are substitutions Θ_{ff} and Θ' and a system of flex-flex equations Ξ_{ff} such that

- $\Theta = \mathrm{Can}(\Theta_{ff} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{ff} :: [\Theta_{ff}]\Xi_{ff}$ and
- $\mathcal{X}' :: \Xi', \xi'' \setminus \Xi_{ff}, \Theta'$.

Then, by applying rule **pu_redex_l** to $\mathcal{X}'$, we obtain the desired derivation $\mathcal{X}$ of $\Xi \setminus \Xi_{ff}, \Theta'$.

Redex-any: We proceed similarly to the previous case.

Any-redex: The treatment of this case is again similar.

Rigid-rigid: We proceed similarly to the cases of spine equations and term equations of composite type.

Rigid-flex: By assumption, we have that $\mathcal{E} :: \Gamma; \Delta \vdash_\Sigma [\Theta](h \cdot S_1) = [\Theta](F \cdot S_2) : a$, where h is some rigid head.

Since, by Lemma 2.19, the equality judgment induces a congruence over terms, there is a derivation $\mathcal{E}'$ of $\Gamma; \Delta \vdash_\Sigma [\Theta](F \cdot S_2) = [\Theta](h \cdot S_1) : a$. Let $\xi' = \Gamma; \Delta \vdash F \cdot S_2 = h \cdot S_1 : a$.

Now, by definition, $((\Xi', \xi'), \Theta, (\vec{\mathcal{E}}', \mathcal{E}')) \prec (\Xi, \Theta, \vec{\mathcal{E}})$. Therefore, we can apply the induction hypothesis and obtain substitutions Θ' and Θ_{ff}, and a flex-flex equation system Ξ_{ff} such that

- $\Theta = \mathrm{Can}(\Theta_{ff} \circ \Theta')_{|\,\mathrm{dom}(\Theta)}$,
- $\vec{\mathcal{E}}_{ff} :: [\Theta_{ff}]\Xi_{ff}$ and
- $\mathcal{X}' :: \Xi', \xi' \setminus \Xi_{ff}, \Theta'$.

where $\xi' = (\Gamma; \Delta \vdash F \cdot S_2 = h \cdot S_1 : a)$. We obtain the desired derivation Ξ' by applying rule **pu_rf** to $\mathcal{X}'$.

Flex-rigid: Then, $\xi = (\Gamma; \Delta \vdash F \cdot S_1 = h \cdot S_2 : a)$, where F has type A' in the current variable pool Φ. From the existence of $\mathcal{E}$, we infer that $\Theta = (\Theta^*, U/F)$ for some canonical term U and substitution Θ^*. Moreover, since Θ is assumed to be well-typed, $\cdot; \cdot \vdash_{\Sigma,\Phi} U : A'$ has a derivation $\mathcal{U}$.

We will distinguish cases on the value of $H_{S_1}(U)$, which exists since A' is the type of U and the source type of S_1.

$H_{S_1}(U) = G$, for some logical variable G.
This case cannot arise since otherwise Θ would not be a solution of Ξ. Indeed, by the relative heads lemma (Lemma 3.3), $\mathrm{Can}([\Theta](F \cdot S_1)) = \mathrm{Can}(U \cdot [\Theta]S_1) = G \cdot S_1'$, for some canonical spine S_1'. On the other hand, $\mathrm{Can}([\Theta](h \cdot S_2)) = h \cdot S_2'$, and $h \neq G$, by assumption.

$H_{S_1}(U) = x$, for some parameter x such that $x : B$ appears in Γ or in Δ. In this situation, the resolution of ξ (and Ξ) proceeds by projection.
We omit the easy proof by induction on $\mathcal{E}$ that $S_1 \div A'$. Moreover, by assumption, U is canonical, $H_{S_1}(U) = x$ and $\mathcal{U} :: \cdot; \cdot \vdash_{\Sigma,\Phi} U : A'$. We are therefore in the conditions of applying the projection lemma (Lemma 3.11). We deduce then that there exist a canonical term V and a canonical substitution $\hat{\Theta}$ such that

- $\cdot; \cdot \vdash A \Uparrow^\pi S \hookrightarrow V$,
- $\cdot; \cdot \vdash_\Sigma [\hat{\Theta}]V = U : A'$,
- $\mathrm{Im}(\hat{\Theta}) \sqsubset U$.

By Assumption 3.6, we have that V and $\hat{\Theta}$ mention logical variables that are distinct from any variable appearing in Ξ or Θ. In particular, $(\hat{\Theta} \circ \Theta^*) = (\hat{\Theta}, \Theta^*)$, and $[\hat{\Theta}]\Theta^* = \Theta^*$ and also $[\Theta^*]V = V$. From this fact, we can deduce the following sequence of equalities:

$$
\begin{array}{lll}
 & \Theta^*, [\hat{\Theta}]V/F & \\
= & [\hat{\Theta}]\Theta^*, [\hat{\Theta}]V/F & \textit{since } [\hat{\Theta}]\Theta^* = \Theta^*, \\
= & (\hat{\Theta} \circ (\Theta^*, V/F))_{|\,F,\mathrm{dom}(\Theta^*)} & \textit{by definition of composition,} \\
= & (\hat{\Theta} \circ (\Theta^* \circ V/F))_{|\,\mathrm{dom}(\Theta)} & \textit{since } [\Theta^*]V = V \textit{ and } \mathrm{dom}(\Theta) = (F, \mathrm{dom}(\Theta^*)), \\
= & ((\hat{\Theta} \circ \Theta^*) \circ V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{by the associativity of substitution composition.}
\end{array}
$$

By a simple induction, it is possible to ascertain that $[\Theta]\Xi$, i.e. $[\Theta^*, U/F]\Xi$, has a derivation if and only if $[\Theta^*, [\hat{\Theta}]V/F]\Xi$ has one. Therefore, by the above equalities and the fact that Ξ contains only variables that are in $\mathrm{dom}(\Theta)$, we have that there is a derivation of $[(\hat{\Theta} \circ \Theta^*) \circ V/F]\Xi$, i.e., by definition of substitution application, of

$$[\hat{\Theta} \circ \Theta^*]([V/F]\Xi).$$

Since $\mathrm{Im}(\hat{\Theta}) \sqsubset U$ and $(\hat{\Theta} \circ \Theta^*) = (\hat{\Theta}, \Theta^*)$, we have that $\mathrm{Im}(\hat{\Theta} \circ \Theta^*) = (\mathrm{Im}(\hat{\Theta}), \mathrm{Im}(\Theta^*)) \sqsubset (U, \mathrm{Im}(\Theta^*)) = \mathrm{Im}(\Theta^*, U/F) = \mathrm{Im}(\Theta)$. Notice also that the substitution $\hat{\Theta} \circ \Theta^*$ is canonical since it corresponds to $(\hat{\Theta}, \Theta^*)$ and both components are canonical. We can therefore apply the induction hypothesis obtaining that there exist substitutions Θ'' and Θ_{ff} and a flex-flex system Ξ_{ff} such that

- $\hat{\Theta} \circ \Theta^* = \mathrm{Can}(\Theta_{ff} \circ \Theta'')_{|\,\mathrm{dom}(\hat{\Theta} \circ \Theta^*)}$,

- $\vec{\mathcal{E}}_{\!f\!f} :: [\Theta_{\!f\!f}]\Xi_{\!f\!f}$, and
- $\mathcal{X}' :: [V/F]\Xi \setminus \Xi_{\!f\!f}, \Theta''$.

Since, by the soundness of staged equality (Theorem 2.17), $U = \mathrm{Can}([\hat{\Theta}]V)$, the sequence of equalities above entails also that $\Theta = \mathrm{Can}((\hat{\Theta} \circ \Theta^*) \circ V/F)_{|\,\mathrm{dom}(\Theta)}$.

In order to conclude this subcase of the proof, we take $\Theta' = \Theta'' \circ V/F$, while keeping $\Theta_{\!f\!f}$ and $\Xi_{\!f\!f}$ unchanged. Then,

- $$\begin{array}{lll} & \Theta & \\ = & \mathrm{Can}((\hat{\Theta} \circ \Theta^*) \circ V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{by the above observation,} \\ = & \mathrm{Can}(\mathrm{Can}(\Theta_{\!f\!f} \circ \Theta'')_{|\,\mathrm{dom}(\hat{\Theta}\circ\Theta^*)} \circ V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{by induction hypothesis,} \\ = & \mathrm{Can}(\mathrm{Can}(\Theta_{\!f\!f} \circ \Theta'') \circ V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{since } \mathrm{dom}(\hat{\Theta} \circ \Theta^*) \subseteq \mathrm{dom}(\Theta), \\ = & \mathrm{Can}((\Theta_{\!f\!f} \circ \Theta'') \circ V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{because of the outer normalization,} \\ = & \mathrm{Can}((\Theta_{\!f\!f} \circ \Theta''), [\Theta_{\!f\!f} \circ \Theta'']V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{by definition of application,} \\ = & \mathrm{Can}(\Theta_{\!f\!f}, [\Theta_{\!f\!f}]\Theta'', [\Theta_{\!f\!f}]([\Theta'']V/F))_{|\,\mathrm{dom}(\Theta)} & \textit{by definition of composition,} \\ = & \mathrm{Can}(\Theta_{\!f\!f}, [\Theta_{\!f\!f}](\Theta'', [\Theta'']V/F)_{|\,\mathrm{dom}(\Theta)} & \textit{by definition of application,} \\ = & \mathrm{Can}(\Theta_{\!f\!f} \circ (\Theta'' \circ V/F))_{|\,\mathrm{dom}(\Theta)} & \textit{by definition of application,} \end{array}$$
- $\vec{\mathcal{E}}_{\!f\!f}$ remains unchanged.
- $\mathcal{X} :: \Xi \setminus \Xi_{\!f\!f}, (\Theta'' \circ V/F)$ by rule **pu_fr_proj** applied to $\mathcal{X}'$.

$\boldsymbol{H_{S_1}(U) = c}$, for some constant c of type B declared in Σ. The equation ξ will be processed by imitation.

We proceed similarly to the case we just analyzed, but rely on the imitation lemma (Lemma 3.13) rather than on the projection lemma. Moreover, we conclude the proof with an appeal to rule **pu_fr_imit**. ☑

3.5 Non-Determinism

Huet's pre-unification algorithm for $\lambda^{\rightarrow}$ is inherently non-deterministic since unification problems in this language usually do not admit most general unifiers. Indeed, when solving flex-rigid equations, we may have to choose between imitation and projection steps and, in the latter case, we might be able to project on different arguments. These are forms of "don't know" non-determinism. The presence of a linear context in $S^{\rightarrow\multimap\&\top}$ and of constructs that operate on it gives rise to a number of new phenomena not present in $\lambda^{\rightarrow}$ unification.

First of all, the manner equations are rewritten in Figure 8 is constrained by the usual context management policy of linear logic. In particular, linear heads in rigid-rigid equations are removed from the context prior to unifying their spines (rule **pu_rr_lvar**). Moreover, when simplifying equations among pairs, the linear context is copied to the two subproblems (**pu_pair**), and equations involving $\langle\rangle$ can always be elided (**pu_unit**). Finally, when solving spine equations, the linear context must be distributed among the linear operands (**pu_lapp**) so that it is empty when the end of the spine is reached (**pu_nil**). As expected, equations among intuitionistic operands are created with an empty linear context (**pu_iapp**). Context splitting in rule **pu_lapp** represents a new form of "don't know" non-determinism not present in Huet's algorithm. Standard techniques of lazy context management [CHP96] can however be used in order to handle it efficiently and deterministically in an actual implementation.

A new inherent form of non-determinism arises in the generation of the spine of substitution terms. Recall that such a term V is constructed in two phases: first, we build its constructor layer, recording local intuitionistic and linear parameters in two accumulators Γ' and Δ', respectively, as λ-abstractions are introduced (first and third parts of Figure 9). Then, we construct a spine on the basis of the available type informations (second and fourth quarter of Figure 9), installing a fresh logical variable as the head of every operand. The contents of Γ' and Δ' must then be distributed as if they were contexts. In particular, we must split Δ' among the linear operands (rules **fri_llam** and **frp_llam**) so that, when the end of spine is generated, no linear parameter is left (rules **fri_nil** and **frp_nil**). Lazy strategies are not

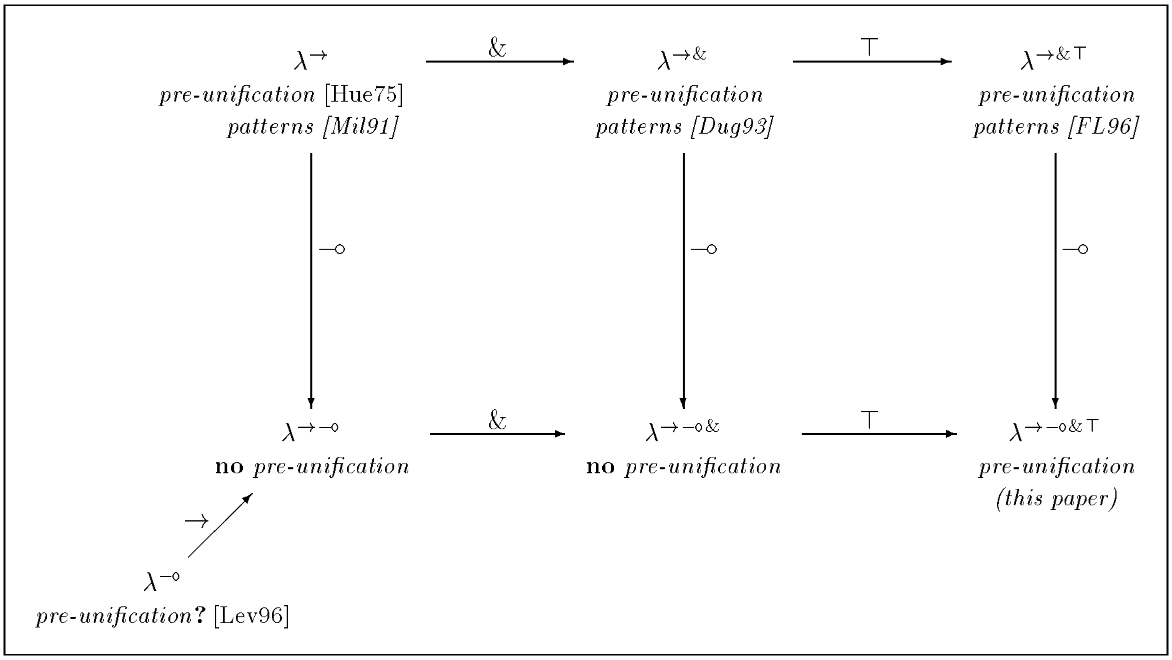

Figure 11: Sublanguages of $\lambda^{\to\multimap\&\top}$

viable in general this time because the heads of these operands are logical variables. Therefore, we must be prepared to non-deterministically consider all possible splits.

This situation is illustrated by the equation

$$x : a, y : a; \cdot \vdash F\,\hat{}\,x\,\hat{}\,y = c\,\hat{}\,(G_1\;x\;y)\,\hat{}\,(G_2\;x\;y) : a.$$

discussed in Section 3.2. An imitation step instantiates F to a term of the form $\hat{\lambda} x' : A.\, \hat{\lambda} y' : B.\, c\,\hat{}\,M_1\,\hat{}\,M_2$ where each of the *linear* parameters x' and y' must appear either in M_1 or in M_2, but not in both. This produces the four solutions presented in Section 3.2. An actual implementation would avoid this additional non-determinism by postponing the choices between the four imitations. A detailed treatment of the necessary constraints between variables occurrences is beyond the scope of this paper (see Section 4.2 for further discussion; a similar technique is used in [HP97]).

4 Discussion

In this section, we consider various sublanguages of $S^{\to\multimap\&\top}$ (or equivalently $\lambda^{\to\multimap\&\top}$) obtained by eliding some of the type operators and the corresponding term constructors and destructors (Section 4.1). We also discuss problems and sketch solutions towards the efficient implementation of a unification procedure for $\lambda^{\to\multimap\&\top}$ (Section 4.2). Finally, we compare our work to related endeavors in the literature (Section 4.3).

4.1 Sublanguages

The omission of one or more of the type operators $\to$, $\multimap$, $\&$ and $\top$ and of the corresponding term constructs from $\lambda^{\to\multimap\&\top}$ (or $S^{\to\multimap\&\top}$) results in a number of λ-calculi with different properties.

First of all, the elision of $\multimap$, $\&$ and $\top$ reduces $\lambda^{\to\multimap\&\top}$ to $\lambda^{\to}$. The few applicable rules in Figures 8–10 constitute then a new presentation of Huet's procedure [Hue75]. The combined use of inference rules and of a spine calculus results in an elegant formulation that can be translated almost immediately into an efficient implementation.

Since linear objects in $\lambda^{\to\multimap\&\top}$ are created and consumed by linear abstraction and application, respectively, every sublanguage not containing $\multimap$ is purely intuitionistic. In particular, $\lambda^{\rightarrow\&}$ coincides with the simply-typed λ-calculus with pairs while $\lambda^{\rightarrow\&\top}$ corresponds to its extension with a unit type and unit element; the latter calculus is tightly related to the notion of Cartesian closed categories [AL91]. Unification in the restricted setting of higher-order patterns has been studied for these two languages in [Dug93] and [FL96], respectively. The appropriate restrictions of the rules in Figures 8–10 implement a general pre-unification procedure for these calculi. Differently from these proposals, our algorithm can solve any unification problem that admits a solution. However, we can guarantee neither termination in the general case, nor efficiency when dealing with higher-order patterns.

The languages $\lambda^{\rightarrow\multimap\&}$ and $\lambda^{\rightarrow\multimap}$ are particularly interesting since the natural restriction of our pre-unification procedure is unsound for them in the following sense: We cannot apply our success criterion since not all flex-flex equations are solvable in this setting. Consider, for example,

$$x\!:\!a, y\!:\!a; \cdot \vdash F\,\hat{}\,x = G\,\hat{}\,y : a.$$

This equation has no solution since F must be instantiated with a term that, after β-reduction, will explicitly use x, and G to a term that must mention y. Furthermore, whether a flex-flex equation has a solution in $\lambda^{\rightarrow\multimap\&}$ or $\lambda^{\rightarrow\multimap}$ is in general undecidable, since, for example, $F\,\hat{}\,M_1 = F\,\hat{}\,M_2$ is equivalent to the generic unification problem $M_1 = M_2$. The situation is clearly different in $\lambda^{\rightarrow\multimap\&\top}$ where $\langle\rangle$ is always available as an information sink in order to eliminate unused linear parameters. However, the usual assumption that there exist closed terms of every type may not be reasonable in $\lambda^{\rightarrow\multimap\&\top}$, and care must be taken in each application regarding the treatment of logical variables which may have no valid ground instances. In conclusion, pre-unification procedures in the sense of Huet are not achievable in the calculi with $\multimap$ but without $\top$.

Finally, a restricted form of unification in the purely linear calculus $\lambda^{\multimap}$ has been studied in [Lev96]. The above counterexamples clearly apply also in this setting, but we have no result about the decidability of higher-order unification in this fragment.

Figure 11 summarizes the taxonomy of sublanguages of $\lambda^{\rightarrow\multimap\&\top}$ we just discussed, their relationships and their properties as far as the existence of a pre-unification algorithm is concerned. We have also inserted references to works on the notion pattern for those languages for which this issue has been the object of research. Patterns in linear language have not been investigated yet. Some considerations can be found in the next section.

4.2 Towards a Practical Implementation

Huet's algorithm for pre-unification in $\lambda^{\rightarrow}$ has been implemented in general proof search engines such as *Isabelle* [NP92] and logic programming languages such as λ*Prolog* [NM88] and shown itself to be reasonably efficient in practice. However, the non-determinism it introduces remains a problem, especially in logic programming. This issue is exacerbated in $\lambda^{\rightarrow\multimap\&\top}$ due to its additional resource non-determinism during imitation and projections.

For $\lambda^{\rightarrow}$, this problem has been addressed by Miller's language of higher-order patterns L_λ [Mil91], which allows occurrences of logical variables to be applied to distinct parameters only. This syntactic restriction guarantees decidability and the existence of most general unifiers. An algorithm that solves equations in the pattern fragment but postpones as constraints any non L_λ equation has been successfully implemented in the higher-order logic programming language *Elf* [Pfe91a]. Unfortunately, an analogous restriction for $\lambda^{\rightarrow\multimap\&\top}$ which would cover the situations arising in practice does not admit most general unifiers. A simple example illustrating this is

$$x\!:\!a; \cdot \vdash F\,\hat{}\,x = c\,\hat{}\,(F_1\,\hat{}\,x)\,\hat{}\,(F_2\,\hat{}\,x) : a.$$

which has the two most general solutions

$$\begin{array}{ll} F \longleftarrow \hat{\lambda} x'\!:\!a.\, c\,\hat{}\,(F_1\,\hat{}\,x')\,\hat{}\,(G_2\,\hat{}\,\langle\rangle), & F_2 \longleftarrow \hat{\lambda} x''\!:\!a.\, G_2\,\hat{}\,\langle\rangle \\ F \longleftarrow \hat{\lambda} x'\!:\!a.\, c\,\hat{}\,(G_1\,\hat{}\,\langle\rangle)\,\hat{}\,(F_2\,\hat{}\,x'), & F_1 \longleftarrow \hat{\lambda} x''\!:\!a.\, G_1\,\hat{}\,\langle\rangle \end{array}$$

neither of which is an instance of the other. This situation is common and occurs in several of our case studies. For certain flex-flex pattern equations, the set of most general unifiers cannot even be described finitely in the language of patterns under any reasonable definition of this notion. This is illustrated by

$$x{:}a, y{:}a; \cdot \vdash F_1\,\hat{}\,\langle x, y\rangle = F_2\,\hat{}\,x\,\hat{}\,y : a.$$

for which the generic solution

$$\begin{aligned} F_1 &\longleftarrow \hat{\lambda} w{:}a \,\&\, a.\, G\,\hat{}\,\langle G_1\,\hat{}\,(\textsc{fst}\ w)\,\hat{}\,\langle\rangle, G_2\,\hat{}\,(\textsc{snd}\ w)\,\hat{}\,\langle\rangle\rangle \\ F_2 &\longleftarrow \hat{\lambda} u{:}a.\, \hat{\lambda} v{:}a.\, G\,\hat{}\,\langle G_1\,\hat{}\,u\,\hat{}\,\langle\rangle, G_2\,\hat{}\,v\,\hat{}\,\langle\rangle\rangle \end{aligned}$$

(which is not a pattern), can be instantiated to infinitely many pattern substitutions by properly choosing a term for the new logical variable G.

Despite these difficulties, the natural generalization of the notion of higher-order pattern introduced by [Dug93] and [FL96] for products to the linear case leads to a decidable unification problem for $\lambda^{\rightarrow\multimap\&\top}$. On this fragment (whose description is beyond the scope of the present paper), termination of the pre-unification algorithm in Section 3 is assured if we also incorporate an appropriate occurs-check as in the simply-typed case. Branching can furthermore be avoided by maintaining linear flex-flex equations as constraints and by using additional constraints between occurrences of parameters. In the first example above, the solution would be

$$F \longleftarrow \hat{\lambda} x'{:}a.\, c\,\hat{}\,(F_1\,\hat{}\,x')\,\hat{}\,(F_2\,\hat{}\,x')$$

with the additional constraint that if x' occurs in $F_1\,\hat{}\,x'$ then it must be absorbed (by $\langle\rangle$) in $F_2\,\hat{}\,x'$ and *vice versa* [HP97]. The second equation above would simply be postponed as a solvable equational constraint. Based on our experience with constraint simplification in *Elf* [Pfe91a] and preliminary experiments, we believe that this will be a practical solution. In particular, the use of explicit substitutions, investigated in [DHKP96] relatively to *Elf*, seems to provide a hook for the required linearity constraints.

4.3 Related Work

So far, only a very limited amount of research has been dedicated to unification algorithms for linear languages. To our knowledge, the only strictly related work, besides the extensive treatment in this paper, is due to Levy. In [Lev96], he studies a generalization of the *contextual unification* problem that corresponds to second-order unification in a formalism akin to the purely linear language $\lambda^{\multimap}$. He provides a sound and complete unification algorithm (flex-flex equations are indeed simplified) and proves its termination for three specific classes of equations. However, he does not discuss the decidability of the general instance of the problem, which, to our knowledge, is still open. In the context of $\lambda^{\multimap}$, our work is more general since the appropriate rules in Figures 8–10 apply to equations of arbitrary order. However, we achieve only pre-unifiers since we keep flex-flex equations as constraints. Instead, when Levy's procedure terminates, it always produces a fully worked-out solution.

Most research on higher-order unification has focused on the simply typed λ-calculus $\lambda^{\rightarrow}$. The most influential work is still the seminal paper [Hue75] by Huet. The individuation of the pattern fragment by Miller [Mil91] and of a terminating and unitary algorithm for it had extensive applications and will influence the direction of our future work. These ideas have been extended in [Pfe91b] to more general languages such as the calculus of constructions [CH88], which includes dependent types, polymorphism and type constructors definition.

Of some relevance in our context is Prehofer's thesis [Pre95] where he considers the specific case of unification in $\lambda^{\rightarrow}$ where the occurrences of logical variables are subject to linear restrictions.

Duggan in [Dug93] extends Miller's work to a calculus akin to $\lambda^{\rightarrow\&}$ that includes product types and impredicative polymorphism [Pfe91b]. These two additions are orthogonal. The basic intuition behind Duggan's treatment of the pairing constructs is that distinct projection sequences applied to a given parameter can be viewed as distinct parameters as far as Miller's definition of patterns is concerned. He implicitly formalizes this idea by giving an alternative formulation of this calculus that emphasizes the role of projections.

Fettig and Löchner push this idea further in [FL96] by defining a calculus that replaces the need for projections with the possibility of abstracting over pairs and more generally tuples. Therefore, they admit terms of the form $\lambda\langle x_1, \ldots x_n\rangle.\, M$. In this setting, their notion of pattern resembles Miller's original proposal. They present a pattern unification procedure for $\lambda^{\rightarrow\&}$ and prove its soundness, completeness and termination. They extend these results to $\lambda^{\rightarrow\&\top}$.

5 Conclusion and Future Work

In this technical report, we have studied the problem of higher-order unification in the context of the linear simply typed λ-calculus $\lambda^{\rightarrow\multimap\&\top}$. A pre-unification algorithm in the style of Huet has been presented for the equivalent spine calculus $S^{\rightarrow\multimap\&\top}$ and new sources of inherent non-determinism due to linearity were pointed out. Moreover, sublanguages of $\lambda^{\rightarrow\multimap\&\top}$ were analyzed and it was shown that pre-unification procedures are not achievable for some of them.

We are currently investigating the computational properties of the natural adaptation of Miller's higher-order patterns to $\lambda^{\rightarrow\multimap\&\top}$. Preliminary examples show that many common unifiable equations do not have most general unifiers due to non-trivial interferences among $\multimap$, $\&$ and $\top$. However, we believe that these problems can be solved through constraint simplification and propagation techniques in a calculus of explicit substitutions.

References

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist.* MIT Press, 1991.

[Bar80] H. P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics.* North-Holland, 1980.

[Bar96] Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Laboratory for Foundations of Computer Sciences, University if Edinburgh, September 1996.

[Cer96] Iliano Cervesato. *A Linear Logical Framework.* PhD thesis, Dipartimento di Informatica, Università di Torino, February 1996.

[CH88] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[CHP96] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Proceedings of the 5th International Workshop on Extensions of Logic Programming*, pages 67–81, Leipzig, Germany, March 1996. Springer-Verlag LNAI 1050.

[CP96] Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, pages 264–275, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

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

[DHKP96] Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 259–273, Bonn, Germany, September 1996. MIT Press.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. MIT Press, 1990.

[Dug93] Dominic Duggan. Unification with extended patterns. Technical Report CS-93-37, University of Waterloo, Waterloo, Ontario, Canada, July 1993. Revised March 1994 and September 1994.

[FL96] Roland Fettig and Bernd Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In H. Ganzinger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications*, pages 347–361, New Brunswick, New Jersey, July 1996. Springer-Verlag LNCS 1103.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[Her95a] Hugo Herbelin. A λ-calculus structure isomorphic to Genzten-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, Eighth Workshop — CSL'94*, pages 61–75, Kazimierz, Poland, 1995. Springer Verlag LNCS 933.

[Her95b] Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes.* PhD thesis, Université Paris 7, 1995.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[HP94] James Harland and David Pym. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2):175–207, April 1994.

[HP97] James Harland and David Pym. Resource distribution via boolean constraints. In W. McCune, editor, *Proceedings of the Fourteenth International Conference on Automated Deduction — CADE-14*, Townsville, Australia, July 1997. To appear.

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[IP96] Samin Ishtiaq and David Pym. A relevant analysis of natural deduction, December 1996. Manuscript.

[JP76] D. C. Jensen and T. Pietrzykowski. Mechanizing ω-order type theory through unification. *Theoretical Computer Science*, 3:123–171, 1976.

[Lev96] Jordi Levy. Linear second-order unification. In H. Ganzinger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications*, pages 332–346, New Brunswick, New Jersey, July 1996. Springer-Verlag LNCS 1103.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Mil96] Dale Miller. A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[Min98] Grigori Mints. Linear lambda-terms and natural deduction. *Studia Logica*, 1998. To appear.

[MP92] Spiro Michaylov and Frank Pfenning. An empirical study of the runtime behavior of higher-order logic programs. In D. Miller, editor, *Proceedings of the Workshop on the λProlog Programming Language*, pages 257–271, Philadelphia, Pennsylvania, July 1992. University of Pennsylvania. Available as Technical Report MS-CIS-92-86.

[NM88] Gopalan Nadathur and Dale Miller. An overview of λProlog. In Kenneth A. Bowen and Robert A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, August 1988. MIT Press.

[NP92] Tobias Nipkow and Lawrence C. Paulson. Isabelle-91. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, pages 673–676, Saratoga Springs, NY, 1992. Springer-Verlag LNAI 607. System abstract.

[Pfe91a] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe91b] Frank Pfenning. Unification and anti-unification in the Calculus of Constructions. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, Amsterdam, The Netherlands, July 1991.

[Pre95] Christian Prehofer. *Solving Higher-Order Equations: From Logic to Programming*. PhD thesis, Technische Universität München, March 1995.

[SG89] Wayne Snyder and Jean H. Gallier. Higher order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.

Notation

ξ	Equation
Γ	Intuitionistic context
Δ	Linear context
Θ	Substitution
Ξ	Equation system
Ξ_{ff}	Flex-flex equation system
Σ	Signature
Φ	Pool
$A, \quad B$	Type
$F, \quad G$	Logical variable
H	Head
$M, \quad N$	Term ($\lambda^{\rightarrow\multimap\&\top}$)
S	Spine
$\widetilde{S}$	Partial spine
$U, \quad V$	Term ($S^{\rightarrow\multimap\&\top}$)
a	Base type
c	Constant
h	Rigid head
$x, \quad y, \quad z, \quad f, \quad u, \quad v, \quad w$	Variables (parameters)
$\mathcal{E}$	Equality derivation
$\vec{\mathcal{E}}$	Multiset equality derivation
$\mathcal{H}$	η-expansion derivation
$\mathcal{R}$	Variable raising derivation
$\mathcal{S}$	Spine typing derivation
$\widetilde{\mathcal{S}}$	Partial spine typing derivation
$\mathcal{U}$	Term typing derivation ($S^{\rightarrow\multimap\&\top}$)
$\mathcal{W}$	Reduction derivation
$\mathcal{X}$	Unification derivation
$c : A$	Constant declaration
$x : A$	Variable (parameter) declaration
$F : A$	Logical variable typing
$\top$	Unit type
$A \& B$	Additive product
$A \multimap B$	Linear arrow
$A \rightarrow B$	Intuitionistic arrow
$\langle\rangle$	Unit element
$\langle M, N\rangle$	Addivite pairing ($\lambda^{\rightarrow\multimap\&\top}$)
$\hat{\lambda} x : A.\, M$	Linear λ-abstraction ($\lambda^{\rightarrow\multimap\&\top}$)
$\lambda x : A.\, M$	Intuitionistic λ-abstraction ($\lambda^{\rightarrow\multimap\&\top}$)
$\textsc{fst}\ M$	First projection ($\lambda^{\rightarrow\multimap\&\top}$)
$\textsc{snd}\ M$	Second projection ($\lambda^{\rightarrow\multimap\&\top}$)

$M\,\hat{}\,N$	Linear application ($\lambda^{\rightarrow\multimap\&\top}$)
$M\ N$	Intuitionistic application ($\lambda^{\rightarrow\multimap\&\top}$)
$[M/x]N$	Meta-level substitution ($\lambda^{\rightarrow\multimap\&\top}$)
$\mathrm{Can}(M)$	Canonical form ($\lambda^{\rightarrow\multimap\&\top}$)
$H \cdot S$	Root
$\langle U, V\rangle$	Addivite pairing ($S^{\rightarrow\multimap\&\top}$)
$\hat{\lambda} x\!:\!A.\,U$	Linear λ-abstraction ($S^{\rightarrow\multimap\&\top}$)
$\lambda x\!:\!A.\,U$	Intuitionistic λ-abstraction ($S^{\rightarrow\multimap\&\top}$)
NIL	End of spine
$\pi_1\, S$	First projection ($S^{\rightarrow\multimap\&\top}$)
$\pi_2\, S$	Second projection ($S^{\rightarrow\multimap\&\top}$)
$U \,\hat{;}\, S$	Linear application ($S^{\rightarrow\multimap\&\top}$)
$U \,;\, S$	Intuitionistic application ($S^{\rightarrow\multimap\&\top}$)
$[V/x]U$	Meta-level substitution in terms ($S^{\rightarrow\multimap\&\top}$)
$[v/x]S$	Meta-level substitution in spines ($S^{\rightarrow\multimap\&\top}$)
$\mathrm{Can}(U)$	Canonical form ($S^{\rightarrow\multimap\&\top}$)
$\mathrm{HNF}(U), \quad \overline{U}$	(Weak) head-normal form
x^A_η	Variable η-expansion
$H \cdot \widetilde{S}$	Partial root
$\widetilde{S} \mathbin{@} \widetilde{S}'$	Partial spine concatenation
U/F	Substitution item
$F \longleftarrow U$	Displayed substitution item
$\mathrm{dom}(\Theta)$	Substitution domain
$\mathrm{Im}(\Theta)$	Substitution image
$\mathrm{rg}(\Theta)$	Substitution range
Θ_{Fs}	Substitution restriction
$[\Theta]U$	Substitution application (term)
$[\Theta]S$	Substitution application (spine)
$[\Theta]\Theta'$	Substitution application (substitution)
$[\Theta]\xi$	Substitution application (equation)
$[\Theta]\Xi$	Substitution application (equation system)
$\Theta \circ \Theta'$	Substitution composition
$\mathrm{Can}(\Theta)$	Canonical form (substitution)
$\Gamma;\Delta \vdash\ U = V : A$	Term equation
$\Gamma;\Delta \vdash\ S_1 = S_2 : A > a$	Spine equation
$\Gamma;\Delta \vdash\ F_1 \cdot S_1 = F_2 \cdot S_2 : a$	Flex-flex equation
$\mathcal{J} :: J$	Judgment derivability
$\Gamma;\Delta \vdash_\Sigma\ M : A$	Typing ($\lambda^{\rightarrow\multimap\&\top}$)
$\Gamma;\Delta \vdash_\Sigma\ U : A$	Term typing ($S^{\rightarrow\multimap\&\top}$)
$\Gamma;\Delta \vdash_\Sigma\ S : A > a$	Spine typing

$U \longrightarrow V$	Reduction (terms)
$S_1 \longrightarrow S_2$	Reduction (spines)
$U \longrightarrow^* V$	Iterated reduction (terms)
$S_1 \longrightarrow^* S_2$	Iterated reduction (spines)
$U \xrightarrow{hr} V$	Head-reduction
$U \xrightarrow{hr}{}^* V$	Iterated head-reduction
$U \xrightarrow{whr} V$	Weak-head reduction
$U \xrightarrow{whr}{}^* V$	Iterated weak-head reduction
$\Gamma; \Delta \vdash_\Sigma U = V : A$	Staged equality (terms)
$\Gamma; \Delta \vdash_\Sigma S_1 = S_2 : A > a$	Staged equality (spines)
$\Gamma; \Delta \vdash_\Sigma H \cdot \widetilde{S} \mathrel{\tilde{:}} A$	Partial root typing
$\Gamma; \Delta \vdash_\Sigma \widetilde{S} \mathrel{\tilde{:}} B > A$	Partial spine typing
$H_1 \cdot \widetilde{S}_1 \xrightarrow{hr} H_2 \cdot \widetilde{S}_2$	Head-reduction for partial roots
$H_1 \cdot \widetilde{S}_1 \xrightarrow{hr}{}^* H_2 \cdot \widetilde{S}_2$	Iterated head-reduction for partial roots
$x \xrightarrow{A} \widetilde{S} \rhd U$	η-expansion
$\Xi \setminus \Xi_{ff}, \Theta$	Unification problem
$\Gamma; \Delta \vdash c \cdot S' \,/\, A \Uparrow^\iota S \hookrightarrow V$	Imitation (terms)
$\Gamma; \Delta \vdash B \Downarrow^\iota S \hookrightarrow S$	Imitation (spines)
$\Gamma; \Delta \vdash A \Uparrow^\pi S \hookrightarrow V$	Projection (terms)
$\Gamma; \Delta \vdash A \Downarrow^\pi a \hookrightarrow S$	Projection (spines)
$\Gamma; \Delta \vdash A \hookrightarrow V$	Variable raising (terms)
$\Gamma; \Delta \vdash a \hookrightarrow S, A$	Variable raising (spines)
$S \div A$	Approximate spine typing
$S_1 \sim S_2$	Approximate spine equality
$H_S(U)$	Relative head
$U =_{raise} V$	Instantiating-term ordering (abstraction raising)
$Us \sqsubseteq V$	Instantiating-term ordering (multiset-term)
$Us \sqsubseteq S$	Instantiating-term ordering (multiset-spine)
$Us \sqsubset V$	Strict instantiating-term ordering (multiset-term)
$Us \sqsubset Vs$	Strict instantiating-term ordering (multiset-multiset)
$(\Xi_1, \Theta_1, \vec{\mathcal{E}}_1) \prec (\Xi_2, \Theta_2, \vec{\mathcal{E}}_2)$	Multiset equality derivation ordering

List of Statements

Lemma 2.1	(Intuitionistic weakening)	5
Lemma 2.2	(Promotion)	6
Lemma 2.3	(Subject reduction)	7
Lemma 2.4	(Substitution)	7
Lemma 2.5	(Confluence)	7
Theorem 2.6	(Strong normalization)	7
Lemma 2.7	(Reduction subsumes head-reduction)	9
Theorem 2.8	(Strong normalization for head-reduction)	9
Lemma 2.9	(Local confluence for head-reduction)	10
Lemma 2.10	(Confluence of head-reduction)	10
Theorem 2.11	(Uniqueness of head-normal forms)	10
Lemma 2.12	(Subject reduction for head-reduction)	11
Lemma 2.13	(Characterization of head-normal forms)	11
Lemma 2.14	(Soundness of $(\overline{\ldots})$)	11
Lemma 2.15	(Completeness of $(\overline{\ldots})$)	11
Lemma 2.16	(Connection between head-normal forms and canonical forms)	12
Theorem 2.17	(Soundness of staged equality)	14
Theorem 2.18	(Completeness of staged equality)	14
Lemma 2.19	(Equality induces a congruence)	14
Lemma 2.20	(Partial typing conservatively extends typing)	16
Lemma 2.21	(Associativity of partial spine concatenation)	17
Lemma 2.22	(Transitivity of partial spine typing)	17
Lemma 2.23	(Concatenation)	17
Lemma 2.24	(Functionality of η-expansion)	18
Lemma 2.25	(Well-typedness of η-expansion)	18
Corollary 2.26	(Well-typedness of η-expansion)	20
Lemma 2.27	(Reduction of η-expanded variables)	20
Corollary 2.28	(Canonical reduction of η-expanded variables)	23
Property 3.1	(Substitutions)	25
Theorem 3.2	(Soundness of linear pre-unification)	34
Lemma 3.3	(Relative heads)	36
Lemma 3.4	(Well-foundedness of $\sqsubset$)	38
Lemma 3.5	(Well-foundedness of $\prec$)	39
Assumption 3.6	(Freshness of substitution terms)	39
Lemma 3.7	(Spines in variable raising)	40
Corollary 3.8	(Spines in variable raising)	41
Lemma 3.9	(Variable raising)	41
Lemma 3.10	(Spines in projection)	42
Lemma 3.11	(Projection)	44
Lemma 3.12	(Spines in imitation)	45
Lemma 3.13	(Imitation)	46
Theorem 3.14	(Completeness of linear pre-unification)	46

Index

$\lambda^{\rightarrow}$, 1
  higher-order patterns, 2
  unification, 2, 51, 53
$\lambda^{\rightarrow\multimap}$, 52
$\lambda^{\rightarrow\multimap\&}$, 52
$\lambda^{\rightarrow\multimap\&\top}$, 2–4
  atomic term, *4*
  canonical form, *4*
  normal form, 4
  reduction semantics, 3
  syntax, 3
  term constructors, 2
  term destructors, 2
  translation to $S^{\rightarrow\multimap\&\top}$, 5
  type constructors, 2
  typing semantics, 3
$\lambda^{\rightarrow\&}$, 52–54
$\lambda^{\rightarrow\&\top}$, 52, 54
$\lambda^{\multimap}$, 52
$S^{\rightarrow\multimap\&\top}$, 4–24
  reduction semantics, 6
  syntax, 5
  translation to $\lambda^{\rightarrow\multimap\&\top}$, 5
  typing semantics, 5

abstract Böhm tree, 5
approximate spine equality, 36
approximate spine typing, 35

base type, 3
β-reduction
  in $\lambda^{\rightarrow\multimap\&\top}$, 3
  in $S^{\rightarrow\multimap\&\top}$, 6

canonical form, 7
commutative conversions, 4
constant, 3
context, *3*
  intuitionistic, *3*
  linear, *3*

derivation ordering, 38

Elf, 2, 52
end of spine, 5
equality, 12–15
  canonical, *8*
  staged, *12*
  well-typed, *13*
equation, *24*
  flex-flex, 24, 31
  flex-rigid, 28
  rigid-flex, 28
  rigid-rigid, 28
  spine, 24
  system, 24
  term, 24
  well-typed, 25
η-expansion, 15–24
η-long form
  in $\lambda^{\rightarrow\multimap\&\top}$, *4*
  in $S^{\rightarrow\multimap\&\top}$, 5
existential variable, *see* logical variable

Forum, 1

head, 4
  flexible, 28
  relative, *see* relative head
  rigid, 27
head-normal form, *8*, 8–12
head-reduction, 8
higher-order patterns
  in $\lambda^{\rightarrow}$, 2, 52
  in $\lambda^{\rightarrow\&}$, 52
  in $\lambda^{\rightarrow\&\top}$, 52
  in *Elf*, 52
  in the calculus of constructions, 53
higher-order unification, *see* unification

imitation, 28
instantiating-term ordering, 37
Isabelle, 52

L_λ, 52
λ*Prolog*, 52
linear logic, 1
  intuitionistic, 3
linear types, 2
LLF, 1, 2
logical variable, 24
Lolli, 1, 2
Lygon, 1

meta-variable, *see* logical variable

non-determinism, 50–51
  context distribution, 50
  context splitting, 50
  equation choice, 26
  product type, 30
  rule choice, 50

parameter, 3, 24
partial root, *16*

partial spine, *15*
    concatenation, *16*
pool, 25
pre-unification procedure, 24
pre-unifier, 24, 31
projection, 28

relative head, 36
RLF, 1
root, *5*
    partial, *see* partial root

signature, *3*
solution, 24
spine, *5*
    partial, *see* partial spine
spine calculus, *see* $S^{\rightarrow\multimap\&\top}$
substitution, 25
    application, 25
    canonical form, 25
    composition, 25
    domain, 25
    image, 25
    meta-level, 3
    range, 25
    well-typed, 25

unification, 1, 24–51
    problem, *24*
    procedure, 24
unifier, 24

variable, *see* parameter *or* logical variable

weak head-normal form, *8*, 8–12
weak head-reduction, 8