Linear Higher-Order Pre-Unification

(Extended abstract)

Iliano Cervesato and Frank Pfenning*
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{*iliano*|*fp*}*@cs.cmu.edu*

Abstract

We develop an efficient representation and 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 ($\top$). Potential applications lie in proof search, logic programming, and logical frameworks based on linear type theories. We also show that, surprisingly, a similar pre-unification algorithm does not exist for certain sublanguages.

1 Introduction

Linear logic [Gir87] refines more traditional logical formalism with direct means of expressing common situations, especially those revolving around a notion of 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* [Mil94], 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 in [IP96], introduce linearity not only at the level of formulas, but also within terms. An implementation of these languages should perform higher-order unification on linear terms in order to instantiate variables.

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. Nevertheless sound and complete (although possibly non-terminating) procedures have been proposed in order to enumerate solutions. In particular, Huet's pre-unification algorithm [Hue75] factorizes 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 is Miller's higher-order patterns [Mil89], that has been implemented in the higher-order logic programming language *Elf* [Pfe91]. 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].

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

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

Appeared in the proceedings of the *CADE-13 Workshop on Proof Search in Type-Theoretic Languages* (D. Galmiche editor), pp. 41–50, New Brunswick, NJ, July 30th 1996.

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

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

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

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

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

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

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*. Its theoretical relevance derives from the fact that it is the biggest linear λ-calculus that admits unique long $\beta\eta$-normal forms.

Our presentation is organized as follows. In Section 2, we define $\lambda^{\rightarrow\multimap\&\top}$ and give an equivalent formulation better suited for our purposes. The pre-unification algorithm is the subject of Section 3. We discuss the unification problem in sublanguages of $\lambda^{\rightarrow\multimap\&\top}$ 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 Lambda Calculus

This section defines the simply-typed linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ (Subsection 2.1) and presents an equivalent formulation, $S^{\rightarrow\multimap\&\top}$ (Subsection 2.2), that turns out convenient for describing and implementing unification.

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 to relate each type constructor (left) to the corresponding term operators (center), with constructors preceding destructors.

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

As usual, we rely on a signature and a context to assign types to constants and free variables, respectively.

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

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 $\lambda^{\rightarrow}$. As usual, we identify terms that differ only in the name of their bound variables and write $[M/x]N$ for the capture-avoiding substitution of M for x in the term N. Contexts and signatures are treated as multisets; we promote "," to denote their union and omit writing "$\cdot$" when not necessary. Finally, we require variables and constants to be declared at most once in a context and in a signature, respectively.

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

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

where Γ and Δ are called the *intuitionistic* and the *linear* context, respectively. The signature Σ is treated purely intuitionistically. 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 $(\rightarrow\ \multimap\&\top)$ fragment of intuitionistic linear logic, $ILL^{\rightarrow\multimap\&\top}$ [HM94], in a natural deduction style formulation. $\lambda^{\rightarrow\multimap\&\top}$ and $ILL^{\rightarrow\multimap\&\top}$ are related by a weak form of the Curry-Howard isomorphism. Indeed, the interactions of rules λ**_unit** and λ**_lapp** can flatten distinct proofs to the same $\lambda^{\rightarrow\multimap\&\top}$ term.

Similarly to $\lambda^{\rightarrow}$, $\lambda^{\rightarrow\multimap\&\top}$ enjoys a number of highly desirable properties [Cer96]. In particular, since every extension introduces commutative conversions, it is the largest linear λ-calculus for which strong normalization holds and yields unique normal forms. We write $\overline{M}$ for the canonical form of the term M, defined as the η-expansion of its β-normal form. In particular, $\overline{x}$ corresponds to the η-long form of the variable x. In the following, we will insist in dealing always with fully η-expanded terms. We call a term of base type *atomic*.

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 $\lambda^{\rightarrow}$ since every η-long normal term has the form

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

where the head t is a constant or a variable and $(t\, M_1\ \ldots M_m)$ is atomic. The usual parentheses saving conventions hide the fact that t is indeed deeply buried in the sequence of application and therefore not immediately accessible. A similar notational trick is not achievable in $\lambda^{\rightarrow\multimap\&\top}$ since on the one hand a non-atomic term can have several heads, possibly none, and on the other hand destructors can be interleaved arbitrarily in an atomic term.

The *spine calculus* $S^{\rightarrow\multimap\&\top}$ permits recovering both efficient head accesses and notational convenience. Every atomic term M of $\lambda^{\rightarrow\multimap\&\top}$ 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, $M = (t\, M_1\ \ldots M_m)$ is written $U = t \cdot (U_1;\ \ldots U_m; \text{NIL})$ in this language, where ";" represents application, 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. The merits of this approach are currently assessed in an experimental implementation. The following grammar describes the syntax of $S^{\rightarrow\multimap\&\top}$: we write constructors as in $\lambda^{\rightarrow\multimap\&\top}$, but use new symbols to distinguish a spine operator from the corresponding term destructor.

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

We adopt the same syntactic conventions as in $\lambda^{\rightarrow\multimap\&\top}$ and often write V for terms in $S^{\rightarrow\multimap\&\top}$. Terms are allowed as heads in order to construct β-redices. Indeed, normal terms have either a constant or a variable as their heads. The typing judgments for terms and spines are denoted $\Gamma;\Delta \vdash_{\Sigma} U : A$ and $\Gamma;\Delta \vdash_{\Sigma} S : A > B$ respectively, where the latter expresses the fact that given a head H of type A, the root $H \cdot S$ has type B. For reasons of space, we omit the typing rules for these judgments, although they will indirectly appear in the inference system for pre-unification.

There exists a structural translation of terms in $\lambda^{\rightarrow\multimap\&\top}$ to terms in $S^{\rightarrow\multimap\&\top}$, and vice versa. Space constraints do not allow presenting this mapping and the proofs of soundness and completeness for the respective typing derivations. We instead describe it by means of examples by giving the translation of the β-reduction rules of $\lambda^{\rightarrow\multimap\&\top}$ (on the left) into $S^{\rightarrow\multimap\&\top}$ (on the right):

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\textsc{fst}\,\langle M,N\rangle & \longrightarrow_\beta & M & \langle U,V\rangle\cdot(\pi_1 S) & \xrightarrow{S}_\beta & U\cdot S \\
\textsc{snd}\,\langle M,N\rangle & \longrightarrow_\beta & N & \langle U,V\rangle\cdot(\pi_2 S) & \xrightarrow{S}_\beta & V\cdot S \\
(\hat{\lambda}x\!:\!A.\,M)\,\hat{}\,N & \longrightarrow_\beta & [N/x]M & (\hat{\lambda}x\!:\!A.\,U)\cdot(V\,\hat{;}\,S) & \xrightarrow{S}_\beta & [V/x]U\cdot S \\
(\lambda x\!:\!A.\,M)\,N & \longrightarrow_\beta & [N/x]M & (\lambda x\!:\!A.\,U)\cdot(V;S) & \xrightarrow{S}_\beta & [V/x]U\cdot S
\end{array}$$

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:

$$(H\cdot S)\cdot\textsc{nil}\ \xrightarrow{S}_\beta\ \ H\cdot S$$

As for $\lambda^{\rightarrow\multimap\&\top}$, we insist on terms being in η-long form. Consequently, roots have always base type and such is the target type in a spine typing judgments. The β-reduction rules above preserve long forms so that η-expansion steps never need to be performed. We write $\overline{U}$ for the canonical form of the term U with respect to these reductions.

3 Linear Higher-Order Unification

In this section, we define the unification problem for $S^{\rightarrow\multimap\&\top}$ (Subsection 3.1) and describe a pre-unification algorithm à la Huet for it (Subsection 3.2).

3.1 The Unification Problem

Two $S^{\rightarrow\multimap\&\top}$ terms U_1 and U_2 are *equal* if they can be β-reduced to a common term V. By strong normalization, it suffices to compute $\overline{U_1}$ and $\overline{U_2}$ and check whether they are syntactically equal (modulo renaming of bound variables). We have the following equality judgments for terms and spines, respectively:

$$\Gamma;\Delta\vdash_\Sigma U_1 = U_2 : A \qquad\qquad \Gamma;\Delta\vdash_\Sigma S_1 = S_2 : A > a$$

The types can be omitted altogether if we assume the two objects in every equation we start from to have the same type. We do not show the deduction rules for these judgments. The interested reader can extract them from the non-flexible cases in Figures 2.

Equality checking becomes a *unification problem* as soon as we admit objects containing *logical 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. 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 $\lambda^{\rightarrow}$, it is undecidable whether two $S^{\rightarrow\multimap\&\top}$ terms can be unified, since its equational theory is a conservative extension of the equational theory for the simply-typed λ-calculus.

Logical variables stand for terms and cannot replace spines. Furthermore, if we keep them too in η-long form, we can restrict their occurrences to head positions. Therefore, the alterations to the definition of $S^{\rightarrow\multimap\&\top}$ required for unification are limited to enriching the syntax of heads with logical variables, that we denote F, G and H possibly subscripted. It can easily be shown that this corresponds to the usual definition. We continue to write U, V and S for terms and spines in this extended language. Finally, in order to avoid confusion, we will hereinafter call the proper variables of $S^{\rightarrow\multimap\&\top}$ *parameters*.

The machinery required in order to state a unification problem is summarized in the grammar below. We will in general solve *systems of equations* that share the same signature and a common set of logical variables. A *solution* to a unification problem is a substitution that, when applied to it, yields a system of equations that is known to be solvable. This notion subsumes unifiers as a particular case. Finally, we record the types of the logical variables in use in a *pool*.

$$\begin{array}{rrl} \textit{Equation systems:} & \Xi & ::= \ \cdot \mid \Xi, (\Gamma; \Delta \vdash U_1 = U_2 : A) \mid \Xi, (\Gamma; \Delta \vdash S_1 = S_2 : A > a) \\ \textit{Substitutions:} & \Theta & ::= \ \cdot \mid \Theta, U/F \\ \textit{Pools:} & \Phi & ::= \ \cdot \mid \Phi, F : A \end{array}$$

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 and pools as multisets. We write ξ for individual equations. The context $\Gamma; \Delta$ in an equation ξ enumerates the parameters that, differently from the contents of Σ, the substitutions for logical variables appearing in ξ are not allowed to mention directly. In particular, a logical variable cannot be replaced by a term containing free linear parameters.

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

$$\Xi \setminus \Theta$$

where, for the sake of readability, we keep the signature Σ and the current variables pool Φ implicit. The procedure we describe in the next subsection accepts Σ, Φ and Ξ as input arguments and attempts to construct a derivation of $\Xi \setminus \Theta$ for some Θ. This could terminate successfully (in which case Θ is indeed a unifier) or reduce the original problem to a subgoal consisting entirely of so-called flex-flex equations (in which case Θ is a pre-unifier, see below). It might also fail (in which case there are no unifiers) or not terminate (in which case we have no information). The above problem is sometimes written $\forall \Sigma. \exists \Phi. \forall (\Xi)$, where the inner expression means that the context $\Gamma; \Delta$ of every equation ξ is universally quantified in front of it.

3.2 A Pre-Unification Algorithm

Our adaptation of Huet's pre-unification procedure to $S^{\rightarrow \multimap \& \top}$ is summarized in Figures 2–3. We adopt a structured operational semantics presentation as a system of inference rules. 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 discuss the specific aspects brought in by linearity in the Section 4.

Given a system of canonical equations Ξ to be solved with respect to a signature Σ and a logical variables pool Φ, the procedure selects an equation ξ from Ξ and attempts to apply in a bottom up fashion one of the rules in Figure 2. 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 Ξ is empty or all equations in it are flex-flex, as described below.

Well-typed equations in canonical 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 dismembered until problems of base type are produced, as shown in the uppermost and lowermost parts of Figure 2, respectively.

Following the standard terminology, we call an atomic term $H \cdot S$ *rigid* if H is a 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 that, when applied to S_1, has c as its head; the resulting spines are then unified as in the rigid-rigid case. We can construct V in two manners: the first, *imitation*, puts c in the proper position in V and rearranges the terms appearing in S_1 in order to match the type of c. The second, *projection*, looks for some subterm that might get instantiated to c inside S_1 and installs it as the appropriate head of V, again reshuffling S_1 to match the type of c. Once V has been produced, it is substituted for every occurrence of F in the equation system and normalization is performed. 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.

Term traversal

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

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

Rigid−Rigid

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

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

Rigid−Flex

$$\frac{\Xi, (\Gamma;\Delta \vdash F \cdot S_2 = c \cdot S_1 : a) \setminus \Theta}{\Xi, (\Gamma;\Delta \vdash c \cdot S_1 = F \cdot S_2 : a) \setminus \Theta}\ \textbf{pu\_rf\_con} \qquad \frac{\Xi, (\Gamma;\Delta \vdash F \cdot S_2 = x \cdot S_1 : a) \setminus \Theta}{\Xi, (\Gamma;\Delta \vdash x \cdot S_1 = F \cdot S_2 : a) \setminus \Theta}\ \textbf{pu\_rf\_var}$$

Flex−Rigid

$$\frac{F{:}A \ in\ \Phi \quad c{:}B \ in\ \Sigma \quad \cdot;\cdot \vdash c_B \cdot S_2 \,/\, A \Uparrow S_1 \hookrightarrow V \quad \overline{[V/F](\Xi, (\Gamma;\Delta \vdash F \cdot S_1 = c \cdot S_2 : a))} \setminus \Theta}{\Xi, (\Gamma;\Delta \vdash F \cdot S_1 = c \cdot S_2 : a) \setminus \Theta, V/F}\ \textbf{pu\_fr\_con\_imit}$$

$$\frac{F{:}A \ in\ \Phi \quad c{:}B \ in\ \Sigma \quad \cdot;\cdot \vdash \cdot \,/\, A \Uparrow S_1 \hookrightarrow V \quad \overline{[V/F](\Xi, (\Gamma;\Delta \vdash F \cdot S_1 = c \cdot S_2 : a))} \setminus \Theta}{\Xi, (\Gamma;\Delta \vdash F \cdot S_1 = c \cdot S_2 : a) \setminus \Theta, V/F}\ \textbf{pu\_fr\_con\_proj}$$

$$\frac{F{:}A \ in\ \Phi \quad \cdot;\cdot \vdash \cdot \,/\, A \Uparrow S_1 \hookrightarrow V \quad \overline{[V/F](\Xi, (\Gamma;\Delta, x{:}B \vdash F \cdot S_1 = x \cdot S_2 : a))} \setminus \Theta}{\Xi, (\Gamma;\Delta, x{:}B \vdash F \cdot S_1 = x \cdot S_2 : a) \setminus \Theta, V/F}\ \textbf{pu\_fr\_lvar\_proj}$$

$$\frac{F{:}A \ in\ \Phi \quad \cdot;\cdot \vdash \cdot \,/\, A \Uparrow S_1 \hookrightarrow V \quad \overline{[V/F](\Xi, (\Gamma, x{:}B;\Delta \vdash F \cdot S_1 = x \cdot S_2 : a))} \setminus \Theta}{\Xi, (\Gamma, x{:}B;\Delta \vdash F \cdot S_1 = x \cdot S_2 : a) \setminus \Theta, V/F}\ \textbf{pu\_fr\_ivar\_proj}$$

Success

$$\frac{}{\cdot \setminus \cdot}\ \textbf{pu\_empty}$$

Spine traversal

$$\frac{\Xi \setminus \Theta}{\Xi, (\Gamma;\cdot \vdash \textsc{nil} = \textsc{nil} : a > a) \setminus \Theta}\ \textbf{pu\_nil}$$

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

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

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

Figure 2: Pre-Unification in $S^{\to\multimap\&\top}$, Equation Manipulation

The construction of the instantiating term V is described in Figure 3. It is based on the judgments

$$\Gamma;\Delta \vdash u \,/\, A \Uparrow S \hookrightarrow V \qquad \Gamma;\Delta \vdash A \Downarrow^{\iota} S' \hookrightarrow S \qquad \Gamma;\Delta \vdash A \Downarrow^{\pi} a \hookrightarrow S \qquad \Gamma;\Delta \vdash A \hookrightarrow V$$

The first builds the constructors layer of V on the basis of the type A of F. The local parameters bound by linear and intuitionistic λ-abstraction are stored in the accumulators Γ and Δ respectively. The second and the third judgments build the spine of V that, after normalization, will be matched against S_2. They apply in case of imitation and projection respectively. Finally, the fourth judgment constructs an η-long term of type A with new logical variables as heads, applied to the parameters

Term construction

$$\frac{\Gamma;\Delta \vdash A \downharpoonleft^{\iota} S' \hookrightarrow S}{\Gamma;\Delta \vdash c_A \cdot S' \,/\, a \Uparrow \textsc{nil} \hookrightarrow c \cdot S}\;\textbf{fr\_imit}$$

$$\frac{\Gamma;\Delta \vdash A \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma;\Delta, x{:}A \vdash \cdot \,/\, a \Uparrow \textsc{nil} \hookrightarrow x \cdot S}\;\textbf{fr\_lproj} \qquad \frac{\Gamma, x{:}A;\Delta \vdash A \downharpoonleft^{\pi} a \hookrightarrow S}{\Gamma, x{:}A;\Delta \vdash \cdot \,/\, a \Uparrow \textsc{nil} \hookrightarrow x \cdot S}\;\textbf{fr\_iproj}$$

$$\frac{\Gamma;\Delta \vdash u \,/\, A_1 \Uparrow S \hookrightarrow V_1 \quad \Gamma;\Delta \vdash A_2 \hookrightarrow V_2}{\Gamma;\Delta \vdash u \,/\, A_1 \,\&\, A_2 \Uparrow \pi_1 S \hookrightarrow \langle V_1, V_2\rangle}\;\textbf{fr\_fst} \qquad \frac{\Gamma;\Delta \vdash u \,/\, A_2 \Uparrow S \hookrightarrow V_2 \quad \Gamma;\Delta \vdash A_1 \hookrightarrow V_1}{\Gamma;\Delta \vdash u \,/\, A_1 \,\&\, A_2 \Uparrow \pi_2 S \hookrightarrow \langle V_1, V_2\rangle}\;\textbf{fr\_snd}$$

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

Spine construction−imitation

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

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

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

Spine construction−projection

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

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

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

Figure 3: Pre-Unification in $S^{\to\multimap\&\top}$, Generation of Substitutions

in $\Gamma;\Delta$. Its inference rules are omitted for space reasons.

Flex-Flex Similarly to $\lambda^{\to}$, a system composed uniquely of flex-flex equations is always solvable in $S^{\to\multimap\&\top}$. 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 $H \cdot \langle\rangle \,\hat{;}\, \textsc{nil}$ (i.e. $F\,\hat{}\,\langle\rangle$ in $\lambda^{\to\multimap\&\top}$), where H is a common new logical variable of type $\top \multimap a$, for some base type a. Then, after normalization, every equation ξ reduces to $\Gamma_\xi;\Delta_\xi \vdash H \cdot \langle\rangle \,\hat{;}\, \textsc{nil} = H \cdot \langle\rangle \,\hat{;}\, \textsc{nil} : a$ which is linearly valid, although extensionally solvable only if such an H 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 achievability of algorithms à la Huet depend crucially on flex-flex equations being always solvable. If this property does not hold, as in some sublanguages of $S^{\to\multimap\&\top}$ we will discuss shortly, these equations must be analyzed with techniques similar to [JP76] or [Mil89].

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, although we have not completed a formal proof at this time, we expect it to be 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 should also be 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.

4 Discussion

In this section, we analyze the role of linearity in the pre-unification procedure for $S^{\rightarrow\multimap\&\top}$ we just devised. We discuss the new sources of non-determinism (Subsection 4.1) and consider various sublanguages of $S^{\rightarrow\multimap\&\top}$ (or equivalently $\lambda^{\rightarrow\multimap\&\top}$) obtained by eliding some of the type operators and the corresponding term constructors and destructors (Subsection 4.2).

.1 on- eterminism

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. The presence in $S^{\rightarrow\multimap\&\top}$ of a linear context 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 2 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_ _l a**). Moreover, when simplifying equations among pairs, the linear context is copied to the two subproblems (**pu_pai**), 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 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. ecall that such a term V is constructed in two phases: first, we build its constructors layer, recording local intuitionistic and linear parameters in two accumulators Γ and Δ, respectively, as λ-abstractions are introduced (upper part of Figure 3). Then, we construct a spine on the basis of the available type informations (central and lower part of Figure 3), 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 **i lla** and **p lla**) so that, when the end of spine is generated, no linear parameter is left (rules **i nil** and **.p nil**). Lazy strategies are not 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.

An example might help clarifying this point. Consider the following equation, written with the syntax of $\lambda^{\rightarrow\multimap\&\top}$ for simplicity,

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

The parameters x and   are intuitionistic, but F uses them as linear objects. An imitation step will instantiate F to a term of the form $\hat{\lambda}x'\!:\!A.\,\hat{\lambda}\ \ '\!:\!B.\,c\,\hat{}\,U_1\,\hat{}\,U_2$ where each of the *linear* parameters x' and $\ \ '$ must appear either in U_1 or in U_2, but not in both. Indeed, the following four incomparable substitutions are generated:

$$\begin{array}{lll}
F & - \hat{\lambda}x'\!:\!A.\,\hat{\lambda}\ \ '\!:\!B.\,c\,\hat{}\,(F_1\,\hat{}\,x'\,\hat{}\ \ ') & \hat{}\ F_2 \\
F & - \hat{\lambda}x'\!:\!A.\,\hat{\lambda}\ \ '\!:\!B.\,c\,\hat{}\,(F_1\,\hat{}\,x') & \hat{}\ (F_2\,\hat{}\ \ ') \\
F & - \hat{\lambda}x'\!:\!A.\,\hat{\lambda}\ \ '\!:\!B.\,c\,\hat{}\,(F_1\,\hat{}\ \ ') & \hat{}\ (F_2\,\hat{}\,x') \\
F & - \hat{\lambda}x'\!:\!A.\,\hat{\lambda}\ \ '\!:\!B.\,c\,\hat{}\ F_1 & \hat{}\ (F_2\,\hat{}\,x'\,\hat{}\ \ ')
\end{array}$$

Imitation on the analogous $\lambda^{\rightarrow}$ equation would return the single substitution

$$F\ \ - \lambda x'\!:\!A.\,\lambda\ \ '\!:\!B.\,c\,(F_1\,x'\ \ ')\,(F_2\,x'\ \ ')$$

In an efficient implementation we would want to postpone the choice between the four imitations and maintain constraints on variable occurrences instead. A further discussion of this idea is beyond the scope of this extended abstract.

.2 Sublanguages

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

First of all, the elision of $\multimap$, & and $\top$ reduces $\lambda^{\rightarrow\multimap\&\top}$ to $\lambda^{\rightarrow}$. The few applicable rules in Figures 2–3 constitute then a new presentation of Huet's procedure. 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 $\lambda^{\rightarrow\multimap\&\top}$ are created and consumed by $\hat{\lambda}$ and $\hat{}$ 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. 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 2–3 implement instead a general pre-unification procedure for these calculi.

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. Indeed, we cannot apply our success criterion since not all flex-flex equations are solvable in this setting. Consider for example

$$x : A, \quad : B; \cdot \vdash F\,\hat{}\,x = G\,\hat{}\quad : a.$$

This equation has no solution since both sides must consume x and , but F will be substituted with terms that use only the parameter x, while G can be replaced only with terms mentioning uniquely . 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 get rid of unused linear parameters. However, the usual assumption that there exist closed terms of every type is not 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.

5 Conclusion and Future Work

In this extended abstract, 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. Finally, 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 techniques.

References

er liano ervesato. *A Linear Logical Framework.* Ph thesis, ipartimento di nformatica, niversita di orino, ebruary .

HP liano ervesato, oshua . Hodas, and rank Pfenning. Efficient resource management for linear logic proof search. n . yckho , H. Herre, and P. chroeder-Heister, editors, *Proceedings of the 5th International Workshop on Extensions of Logic Programming*, pages , eip ig, ermany, arch . pringer- erlag .

P liano ervesato and rank Pfenning. linear logical framework. n E. larke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, ew runswick, ew ersey, uly . EEE omputer ociety Press. o appear.

ug ominic uggan. nification with extended patterns. echnical eport - - , niversity of Waterloo, Waterloo, ntario, anada, uly . evised arch and eptember .

oland ettig and ernd ochner. nification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. n H. an inger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications*, ew runswick, ew ersey, uly . o appear.

ir ean- ves irard. inear logic. *Theoretical Computer Science*, , .

H oshua Hodas and ale iller. ogic programming in a fragment of intuitionistic linear logic. *Information and Computation*, ( ) , . preliminary version appeared in the Proceedings of the ixth nnual EEE ymposium on ogic in omputer cience, pages , msterdam, he etherlands, uly .

HP ames Harland and avid Pym. uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, ( ) , pril .

Hue erard Huet. unification algorithm for typed λ-calculus. *Theoretical Computer Science*, , .

P amin shtia and avid Pym, ay . Personal communications.

P . . ensen and . Pietr ykowski. echani ing -order type theory through unification. *Theoretical Computer Science*, , .

ev ordi evy. inear second-order unification. n H. an inger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications*, ew runswick, ew ersey, uly . o appear.

il ale iller. logic programming language with lambda-abstraction, function variables, and simple unification. n Peter chroeder-Heister, editor, *Proceedings of the International Workshop on Extensions of Logic Programming*, pages , ubingen, ermany, . pringer- erlag .

il ale iller. multiple-conclusion meta-logic. n . bramsky, editor, *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages , Paris, rance, uly .

opalan adathur and ale iller. n overview of λProlog. n enneth . owen and obert . owalski, editors, *Fifth International Logic Programming Conference*, pages , eattle, Washington, ugust . Press.

Pfe rank Pfenning. ogic programming in the logical framework. n erard Huet and ordon Plotkin, editors, *Logical Frameworks*, pages . ambridge niversity Press, .