A Linear Spine Calculus

Iliano Cervesato and Frank Pfenning[1]

April 10, 1997
CMU-CS-97-125

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present the spine calculus $S^{\rightarrow\multimap\&\top}$ as an efficient representation for the linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ which includes intuitionistic functions ($\rightarrow$), linear functions ($\multimap$), additive pairing ($\&$), and additive unit ($\top$). $S^{\rightarrow\multimap\&\top}$ enhances the representation of Church's simply typed λ-calculus as abstract Böhm trees by enforcing extensionality and by incorporating linear constructs. This approach permits procedures such as unification to retain the efficient head access that characterizes first-order term languages without the overhead of performing η-conversions at run time. Potential applications lie in proof search, logic programming, and logical frameworks based on linear type theories. We define the spine calculus, give translations of $\lambda^{\rightarrow\multimap\&\top}$ into $S^{\rightarrow\multimap\&\top}$ and vice-versa, prove their soundness and completeness with respect to typing and reductions, and show that the spine calculus is strongly normalizing and admits unique canonical forms.

[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.

Keywords: Linear Lambda Calculus, Abstract Böhm Trees, Term Assignment Systems, Uniform Provability.

Contents

List of Figures

1 Introduction

The internal representation of λ-calculi, logics and type theories has a direct impact on the efficiency of systems for symbolic computation that implement them, theorem provers and logic programming languages for example. In particular, major gains can be achieved from even small improvements of procedures that manipulate terms extensively: unification, for instance, is a well-known bottleneck in the execution time of a logic program. For languages based on first-order terms, *Prolog* for example, the natural representation of terms supports simple and fast unification algorithms. Indeed, a function symbol f applied to three arguments a, b and c, written $f(a, b, c)$ in the syntax of *Prolog*, is encoded as a record consisting of the head f and the list of its arguments. This is sensible from the point of view of unification since the head of a terms must be analyzed before its arguments. Systems embedding a higher-order term language, the logic programming languages *Elf* [Pfe91, Pfe94] and λ*Prolog* [NM88] for example, typically represent terms in a way that mimics the traditional definition of a λ-calculus. Ignoring common orthogonal optimizations such as the use of DeBruijn indices [dB72] or explicit substitutions [ACCL91], the above term is parsed and encoded as $(((f\ a)\ b)\ c)$. During unification, three applications (here represented as juxtaposition) must be traversed before accessing its head, possibly just to discover that it differs from the head of the term being unified. This representation is similarly inefficient when normalizing a term: in order to reduce $((\lambda x.\, \lambda y.\, \lambda z.\, f\ x\ y\ z)\ a\ b\ c)$ to the above term, we need again to go through three applications before exposing the first redex.

Apparently, adopting an internal representation that treats nested applications as in the first-order case (i.e., as a head together with a list of arguments) but permits λ-abstraction would improve significantly the efficiency of higher-order unification algorithms. This approach, known as the Böhm tree representation, has been studied extensively for different purposes [Bar80, Her95]. However, the complex equational theory that characterizes a λ-calculus leads to difficulties in procedures such as unification and normalization. In particular, η-conversion rules can yield instances of a same function symbol applied to a different number of arguments. This might even lead to fragmented lists of argument as the result of β-reduction (e.g. while performing unification) that need to be monitored and compacted regularly. Ultimately, abstract Böhm trees turn out to be even more complex to deal with than traditional λ-expressions. Instead, no such difficulty emerges with the trivial equational theory of first-order terms.

In this paper, we propose a variant of abstract Böhm trees that supports efficient head accesses, but that does not suffer from the drawbacks we just mentioned. This representation of λ-terms, that we call generically a *spine calculus*, is based on the observation that, in a typed λ-calculus, the use of the troublesome η-conversion rules can be limited to a preprocessing phase that expands terms to unique η-long forms, which are preserved by β-reduction. Insisting on η-long terms has the advantage of simplifying the code for procedures such as unification and normalization, of permitting easier informal descriptions of these algorithms, and more generally of reducing the complexity of studying the meta-theory of such formalisms. Moreover, λ-calculi featuring a unit type and a unit element do not admit subject reduction unless all terms are η-expanded [JG95]: this means that typing information must be stored and maintained in otherwise type-free procedures such as unification.

The benefits of the spine calculus representation, in conjunction with explicit substitutions, are currently assessed in a new implementation of the logical framework *LF* [HHP93] as the higher-order logic programming language *Twelf*, the successor of *Elf* [Pfe91, Pfe94]. *LF* is based on the type theory λ^{Π}, a refinement of Church's simply-typed λ-calculus $\lambda^{\rightarrow}$ with dependent types. In this paper, we will instead focus on the simply-typed linear λ-calculus $\lambda^{\rightarrow \multimap \& \top}$, which extends $\lambda^{\rightarrow}$ with the type constructors $\multimap$, & and $\top$, derived from the identically denoted connectives of linear logic [Gir87]. We will define the corresponding spine calculus $S^{\rightarrow\multimap\&\top}$, present translations between the two, and prove the meta-theoretical properties of $S^{\rightarrow\multimap\&\top}$ that make it adequate as an internal representation language for $\lambda^{\rightarrow\multimap\&\top}$. Notice that our analysis applies to any sublanguage of $\lambda^{\rightarrow\multimap\&\top}$, in particular to $\lambda^{\rightarrow}$ and its extension with extensional products and a unit type, $\lambda^{\rightarrow\&\top}$; moreover, it can easily be extended to the treatment of dependent types.

A similar proposal for term representation was already mentioned in passing by Howard in his seminal paper [How69]. The normal forms of the spine calculus also arise as a term assignment language for uniform proofs, which form the basis for abstract logic programming languages and is based on a much richer set of connectives [MNPS91]. A thorough investigation of a related calculus on the $\lambda^{\rightarrow}$ fragment has been conducted by Herbelin [Her95]. Schwichtenberg [Sch97] studies a version of the intuitionistic spine representation and ordinary λ-calculi in a single system which incorporates permutative conversions, instead of the wholesale translation investigated here (which is closer to an efficient implementation).

$\lambda^{\rightarrow\multimap\&\top}$ corresponds, via a natural extension of the Curry-Howard isomorphism, to the $(\rightarrow\multimap\&\top)$ fragment of intuitionistic linear logic, which constitutes the propositional core of the logic programming language *Lolli* [HM94] and of the linear logical framework *LLF* [Cer96, 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. $\lambda^{\rightarrow\multimap\&\top}$ shares similarities with the calculus proposed in [Bar96] and with the term language of the system *RLF* [IP96].

The implementation of a language based on a linear type theories such as *LLF* and *RLF* raises new challenges that do not emerge neither for intuitionistic languages such as *Elf* [Pfe94], nor in linear logic programming languages featuring plain intuitionistic terms such as *Lolli* [HM94] or *Forum* [Mil94]. In particular, the implementation of formalisms based on a linear λ-calculus must perform higher-order unification on linear terms in order to instantiate existential variables [CP97]. The spine calculus $S^{\rightarrow\multimap\&\top}$ was designed as an efficient representation for unification and normalization over the linear λ-expressions that can appear in an *LLF* specification.

The adoption 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]).

The principal contribution of this work is the definition of spine calculi (1) as a new representation technique for generic λ-calculi that permits both simple meta-reasoning and efficient implementations, and (2) as a term assignment system for the logic programming notion of uniform provability.

Our presentation is organized as follows. In Section 2, we define $\lambda^{\rightarrow\multimap\&\top}$ and present its main properties. We introduce the syntax and the typing and reduction semantics of $S^{\rightarrow\multimap\&\top}$ in Section 3. In Section 4, we give translations from the traditional presentation to the spine calculus and vice-versa and show that they are sound and complete with respect to the typing and reduction semantics of both languages. In Section 5, we state and prove the major properties of $S^{\rightarrow\multimap\&\top}$. Further remarks are made in Section 6. Finally, Section 7 summarizes the work done, discusses applications and hints at future development. In order to facilitate our description, we must assume the reader familiar with linear logic [Gir87].

2 The Linear Simply-Typed Lambda Calculus $\lambda^{\rightarrow\multimap\&\top}$

In this section, we introduce the linear simply-typed λ-calculus $\lambda^{\rightarrow\multimap\&\top}$, which augments Church's simply-typed λ-calculus $\lambda^{\rightarrow}$ [Chu40] with a number of operators from linear logic [Gir87]. More precisely, we give its syntax in Section 2.1, present its typing semantics in Section 2.2 and its reduction semantics in Section 2.3. $\lambda^{\rightarrow\multimap\&\top}$ is the simply-typed variant of the linear type theory $\lambda^{\Pi\multimap\&\top}$, thoroughly analyzed in [Cer96]. We refer the interested reader to this work for the proofs of the properties of $\lambda^{\rightarrow\multimap\&\top}$ stated

Pre−canonical terms

$$\frac{\Gamma; \Delta \vdash_\Sigma M \downarrow a}{\Gamma; \Delta \vdash_\Sigma M \Uparrow a}\,\mathsf{l\lambda\_atm}$$

$$\frac{}{\Gamma; \Delta \vdash_\Sigma \langle\rangle \Uparrow \top}\,\mathsf{l\lambda\_unit} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M \Uparrow A \quad \Gamma; \Delta \vdash_\Sigma N \Uparrow B}{\Gamma; \Delta \vdash_\Sigma \langle M, N\rangle \Uparrow A \,\&\, B}\,\mathsf{l\lambda\_pair}$$

$$\frac{\Gamma; \Delta, x{:}A \vdash_\Sigma M \Uparrow B}{\Gamma; \Delta \vdash_\Sigma \hat{\lambda} x{:}A.\, M \Uparrow A \multimap B}\,\mathsf{l\lambda\_llam} \qquad \frac{\Gamma, x{:}A; \Delta \vdash_\Sigma M \Uparrow B}{\Gamma; \Delta \vdash_\Sigma \lambda x{:}A.\, M \Uparrow A \to B}\,\mathsf{l\lambda\_ilam}$$

Pre−atomic terms

$$\frac{\Gamma; \Delta \vdash_\Sigma M \Uparrow A}{\Gamma; \Delta \vdash_\Sigma M \downarrow A}\,\mathsf{l\lambda\_redex}$$

$$\frac{}{\Gamma; \cdot \vdash_{\Sigma, c:A} c \downarrow A}\,\mathsf{l\lambda\_con} \qquad \frac{}{\Gamma; x{:}A \vdash_\Sigma x \downarrow A}\,\mathsf{l\lambda\_lvar} \qquad \frac{}{\Gamma, x{:}A; \cdot \vdash_\Sigma x \downarrow A}\,\mathsf{l\lambda\_ivar}$$

$$\text{(No rule for } \top\text{)} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M \downarrow A \,\&\, B}{\Gamma; \Delta \vdash_\Sigma \textsc{fst}\, M \downarrow A}\,\mathsf{l\lambda\_fst} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M \downarrow A \,\&\, B}{\Gamma; \Delta \vdash_\Sigma \textsc{snd}\, M \downarrow B}\,\mathsf{l\lambda\_snd}$$

$$\frac{\Gamma; \Delta' \vdash_\Sigma M \downarrow A \multimap B \quad \Gamma; \Delta'' \vdash_\Sigma N \Uparrow A}{\Gamma; \Delta', \Delta'' \vdash_\Sigma M\,\hat{}\,N \downarrow B}\,\mathsf{l\lambda\_lapp} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M \downarrow A \to B \quad \Gamma; \cdot \vdash_\Sigma N \Uparrow A}{\Gamma; \Delta \vdash_\Sigma M\, N \downarrow B}\,\mathsf{l\lambda\_iapp}$$

Figure 1: Typing for η-long $\lambda^{\to\multimap\&\top}$ Terms

in this section.

2.1 Syntax

The linear simply-typed λ-calculus $\lambda^{\to\multimap\&\top}$ extends Church's $\lambda^{\to}$ 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 is convenient in developments of this work that go beyond the scope of this paper. 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. Clearly, constants and variables can have any type.

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

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 and B for terms and types, respectively.

The notions of free and bound variables are adapted from $\lambda^{\to}$. 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.

2.2 Typing Semantics

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

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

We will also use the letter Δ, possibly subscripted, to indicate a context. Contexts and signatures are treated as multisets; we promote "," to denote their union and omit writing "$\cdot$" when unnecessary. Finally, we require variables and constants to be declared at most once in a context and in a signature, respectively.

Operating solely on well-typed terms in η-long form is particularly convenient when implementing operations such as unification since it strongly restricts the structure that a term of a given type can assume. Instead, untyped η-conversion rules are often included in the reduction semantics of a λ-calculus in order to focus on η-long representatives when needed. In the presence of a unit element, $\langle\rangle$ in $\lambda^{\rightarrow\multimap\&\top}$, this approach is unsound. We cleanly realize the above desideratum by distinguishing a pre-canonical typing judgment, which validates precisely the well-typed terms of $\lambda^{\rightarrow\multimap\&\top}$ in η-long form (*pre-canonical terms*), from a pre-atomic judgment, which handles intermediate stages of their construction (*pre-atomic terms*). These judgments are respectively denoted as follows:

$$\begin{array}{ll} \Gamma;\Delta \vdash_\Sigma M \Uparrow A & \quad M \textit{ is a pre-canonical term of type } A \textit{ in } \Gamma;\Delta \textit{ and } \Sigma \\ \Gamma;\Delta \vdash_\Sigma M \downarrow A & \quad M \textit{ is a pre-atomic term of type } A \textit{ in } \Gamma;\Delta \textit{ and } \Sigma \end{array}$$

where Γ and Δ are called the *intuitionistic* and the *linear* context, respectively. Whenever a property holds uniformly for the pre-canonical and pre-atomic judgments above, we will write $\Gamma;\Delta \vdash_\Sigma M \Updownarrow A$ and then refer to the term M and the type A if needed. Moreover, if two or more such expressions occur in a statement, we assume that the arrows of the actual judgments match, unless explicitly stated otherwise.

The rules displayed in the upper part of Figure 1 validate pre-canonical terms M by deriving judgments of the form $\Gamma;\Delta \vdash_\Sigma M \Uparrow A$. Rules **l$\lambda$_unit**, **l$\lambda$_pair**, **l$\lambda$_llam** and **l$\lambda$_ilam** allow the construction of terms of the form $\langle\rangle$, $\langle M_1, M_2\rangle$, $\hat{\lambda}x:A.\,M$, and $\lambda x:A.\,M$, respectively. The manner they handle their context is familiar from linear logic. Notice in particular that **lλ_unit** is applicable with any linear context and that the premisses of rule **lλ_pair** share the same context, which also appears in its conclusion. Rules **lλ_llam** and **lλ_ilam** differ only by the nature of the assumption they add to the context in their premiss: linear in the case of the former, intuitionistic for the latter. The remaining rule defining the pre-canonical judgment, **lλ_atm**, is particularly interesting since it is the reason why all terms derivable in the pre-canonical system are in η-long form. Notice that this rule can be applied only at base types.

The rules defining the pre-atomic judgment, $\Gamma;\Delta \vdash_\Sigma M \downarrow A$, are displayed in the lower part of Figure 1. They validate constants (rule **lλ_con**) and linear and intuitionistic variables (rules **lλ_lvar** and **lλ_ivar**, respectively). They also allow the formation of the terms $\textsc{fst}\ M$, $\textsc{snd}\ M$, $M\,\hat{}\,N$ and $M\ N$ (rules **lλ_fst**, **lλ_snd**, **lλ_lapp** and **lλ_iapp**, respectively). The role played by linear assumptions in $\lambda^{\rightarrow\multimap\&\top}$ is particularly evident in these rules. Indeed, an axiom rule (**lλ_con**, **lλ_lvar** and **lλ_ivar**) can be applied only if the linear part of its context is empty, or contains just the variable to be validated, with the proper type. Linearity appears also in the elimination rule for $\multimap$, where the linear context in the conclusion of rule **lλ_lapp** is split and distributed among its premisses. Observe also that the linear context of the argument part of an intuitionistic application, in rule **lλ_iapp**, is constrained to be empty. The presence of rule **lλ_redex** accounts for the possibility of validating terms containing β-redices, as defined below. If we remove it, only η-long β-normal (or more succinctly *canonical*) terms can be derived.

This formulation of the typing semantics of $\lambda^{\rightarrow\multimap\&\top}$ is the simply-typed variant of the pre-canonical system which defines the semantics of the linear type theory underlying *LLF* [Cer96, CP96]. We direct the interested reader to these references for the proofs of the statements in this section.

If we ignore the terms and the distinction between the pre-canonical and the pre-atomic judgments, the rules in Figure 1 correspond to the specification of the familiar inference rules for the $(\rightarrow\multimap\&\top)$ fragment of intuitionistic linear logic, $ILL^{\rightarrow\multimap\&\top}$ [HM94], presented in a *natural deduction* style. It is easy to prove the equivalence to the usual sequent formulation. $\lambda^{\rightarrow\multimap\&\top}$ and $ILL^{\rightarrow\multimap\&\top}$ are related by a form of the Curry-Howard isomorphism: the terms that appear on the left of the types in the above judgments record the structure of a natural deduction proof for the corresponding linear formulas. Note that the interactions of rules **λ_unit** and **λ_lapp** can flatten distinct proofs to the same $\lambda^{\rightarrow\multimap\&\top}$ term.

Extensionality, i.e., the property of validating only η-long terms, contributes to achieving the simple and elegant formulation of the pre-unification algorithm for $\lambda^{\rightarrow\multimap\&\top}$ described in [CP97]. More importantly, this property and the subject reduction lemma below account for the possibility of omitting

Congruences

$$\frac{M \longrightarrow M'}{\langle M, N\rangle \longrightarrow \langle M', N\rangle}\ \textbf{lr\_pair1} \qquad \frac{N \longrightarrow N'}{\langle M, N\rangle \longrightarrow \langle M, N'\rangle}\ \textbf{lr\_pair2}$$

$$\frac{M \longrightarrow M'}{\textsc{fst}\ M \longrightarrow \textsc{fst}\ M'}\ \textbf{lr\_fst} \qquad \frac{M \longrightarrow M'}{\textsc{snd}\ M \longrightarrow \textsc{snd}\ M'}\ \textbf{lr\_snd}$$

$$\frac{M \longrightarrow M'}{\hat{\lambda} x\!:\!A.\, M \longrightarrow \hat{\lambda} x\!:\!A.\, M'}\ \textbf{lr\_llam} \qquad \frac{M \longrightarrow M'}{M\,\hat{}\,N \longrightarrow M'\,\hat{}\,N}\ \textbf{lr\_lapp1} \qquad \frac{N \longrightarrow N'}{M\,\hat{}\,N \longrightarrow M\,\hat{}\,N'}\ \textbf{lr\_lapp2}$$

$$\frac{M \longrightarrow M'}{\lambda x\!:\!A.\, M \longrightarrow \lambda x\!:\!A.\, M'}\ \textbf{lr\_ilam} \qquad \frac{M \longrightarrow M'}{M\ N \longrightarrow M'\ N}\ \textbf{lr\_iapp1} \qquad \frac{N \longrightarrow N'}{M\ N \longrightarrow M\ N'}\ \textbf{lr\_iapp2}$$

β−reductions

$$\frac{}{\textsc{fst}\ \langle M, N\rangle \longrightarrow M}\ \textbf{lr\_beta\_fst} \qquad \frac{}{\textsc{snd}\ \langle M, N\rangle \longrightarrow N}\ \textbf{lr\_beta\_snd}$$

$$\frac{}{(\hat{\lambda} x\!:\!A.\, M)\,\hat{}\,N \longrightarrow [N/x]M}\ \textbf{lr\_beta\_lin} \qquad \frac{}{(\lambda x\!:\!A.\, M)\ N \longrightarrow [N/x]M}\ \textbf{lr\_beta\_int}$$

Figure 2: Reduction Semantics for $\lambda^{\rightarrow\multimap\&\top}$

type information in an implementation of this procedure, an essential efficiency gain. Extensionality is formalized in the following lemma, which proof can be easily adapted from [Cer96].

Lemma 2.1 (*Extensionality*)

i. *If* $\Gamma;\Delta \vdash_\Sigma M \Uparrow a$, *then* M *is one of* c, x, $\textsc{fst}\ N$, $\textsc{snd}\ N$, $N_1\,\hat{}\,N_2$, $N_1\ N_2$;

ii. *If* $\Gamma;\Delta \vdash_\Sigma M \Uparrow \top$, *then* $M = \langle\rangle$;

iii. *If* $\Gamma;\Delta \vdash_\Sigma M \Uparrow A \,\&\, B$, *then* $M = \langle N_1, N_2\rangle$;

iv. *If* $\Gamma;\Delta \vdash_\Sigma M \Uparrow A \multimap B$, *then* $M = \hat{\lambda} x\!:\!A.\, N$;

v. *If* $\Gamma;\Delta \vdash_\Sigma M \Uparrow A \rightarrow B$, *then* $M = \lambda x\!:\!A.\, N$. □

2.3 Reduction Semantics

The reduction semantics of $\lambda^{\rightarrow\multimap\&\top}$ is given by the congruence relation on terms $\longrightarrow$ based on the following β-reduction rules:

$$\beta_{fst}: \quad \textsc{fst}\ \langle M, N\rangle \longrightarrow M \qquad\qquad \beta_{lapp}: \quad (\hat{\lambda} x\!:\!A.\, M)\,\hat{}\,N \longrightarrow [N/x]M$$
$$\beta_{snd}: \quad \textsc{snd}\ \langle M, N\rangle \longrightarrow N \qquad\qquad \beta_{iapp}: \quad (\lambda x\!:\!A.\, M)\ N \longrightarrow [N/x]M.$$

The complete definition of $\longrightarrow$ is displayed in Figure 2. If $M \longrightarrow N$ is derivable, then N differs from M by the reduction of exactly one redex. We denote its reflexive and transitive closure as $\longrightarrow^*$, and use $\equiv$ for the corresponding equivalence relation. It is easy to show that the rules obtained from Figure 2 by replacing $\longrightarrow$ with $\longrightarrow^*$ (or even with $\equiv$) are admissible. We adopt the standard terminology and call a term M that does not contain β-redices *normal*, or *β-normal*. When emphasizing the fact that our well-typed terms are η-long, we will instead use the term *canonical*.

Similarly to $\lambda^{\rightarrow}$, $\lambda^{\rightarrow\multimap\&\top}$ enjoys a number of highly desirable properties [Cer96]. In particular, confluence and the Church-Rosser property hold for this language, as expressed by the following lemma:

Theorem 2.2 (*Church-Rosser*)

Confluence: *If* $M \longrightarrow^* M'$ *and* $M \longrightarrow^* M''$, *then there is a term* N *such that* $M' \longrightarrow^* N$ *and* $M'' \longrightarrow^* N$.

Church-Rosser: *If* $M' \equiv M''$, *then there is a term* N *such that* $M' \longrightarrow^* N$ *and* $M'' \longrightarrow^* N$. □

Moreover, $\lambda^{\rightarrow \multimap \& \top}$ enjoys the following substitution principle (also known as *transitivity lemma*), that, among many interpretations, permits viewing variables as unspecified hypothetical derivations to be instantiated with actual derivations. Notice the different treatment of intuitionistic and linear variables.

Lemma 2.3 (*Transitivity*)

i. If $\Gamma; \Delta, x : B \vdash_\Sigma M \Uparrow\!\!\Downarrow A$ *and* $\Gamma; \Delta' \vdash_\Sigma N \Uparrow B$*, then* $\Gamma; \Delta, \Delta' \vdash_\Sigma [N/x]M \Uparrow\!\!\Downarrow A$.

ii. If $\Gamma, x : B; \Delta \vdash_\Sigma M \Uparrow\!\!\Downarrow A$ *and* $\Gamma; \cdot \vdash_\Sigma N \Uparrow B$*, then* $\Gamma; \Delta \vdash_\Sigma [N/x]M \Uparrow\!\!\Downarrow A$. □

An important computational property of a typed λ-calculus is subject reduction: it states that reductions do not alter the typability (and the type) of a term. The lemma below also implies that β-reductions do not interfere with extensionality: reducing a redex rewrites η-long terms to η-long terms.

Lemma 2.4 (*Subject reduction*)

If $\Gamma; \Delta \vdash_\Sigma M \Uparrow\!\!\Downarrow A$ *and* $M \longrightarrow^* N$*, then* $\Gamma; \Delta \vdash_\Sigma N \Uparrow\!\!\Downarrow A$. □

Our calculus also enjoys strong normalization, i.e., a well-typed term cannot undergo an infinite sequence of β-reductions. Said in another way, a normal form will eventually be reached no matter which β-redex we choose to reduce first.

Theorem 2.5 (*Strong normalization*)

If $\Gamma; \Delta \vdash_\Sigma M \Uparrow\!\!\Downarrow A$*, then M is strongly normalizing.* □

Finally, well-typed terms have unique normal forms, up to the renaming of bound variables. Since every extension of $\lambda^{\rightarrow \multimap \& \top}$ (for example with $\otimes$ and multiplicative pairs) introduces commutative conversions, this language is the largest linear λ-calculus for which strong normalization holds and yields unique normal forms.

Corollary 2.6 (*Uniqueness of normal forms*)

If $\Gamma; \Delta \vdash_\Sigma M \Uparrow\!\!\Downarrow A$*, then there is a unique normal term N such that* $M \longrightarrow^* N$. □

We write $\mathrm{Can}(M)$ for the *canonical form* of the term M, defined as the η-expansion of its β-normal form. A calculus that validates only canonical terms can easily be obtained from the system in Figure 1 by removing rule **lλ_redex**.

To achieve better efficiency in an implementation of this calculus, we sometimes refer to the *weak head-normal form* of a term M, written $\overline{M}$, that differs from $\mathrm{Can}(M)$ for the possible presence of redices in the arguments of applications. Notice that $\overline{x}$ corresponds to the η-long form of the variable x.

3 The Spine Calculus $S^{\rightarrow \multimap \& \top}$

In this section, we present an alternative formulation of $\lambda^{\rightarrow \multimap \& \top}$, the spine calculus $S^{\rightarrow \multimap \& \top}$, that we suspect permits achieving more efficient implementation of critical procedures such as unification [CP97]. We describe the syntax, typing and reduction semantics of $S^{\rightarrow \multimap \& \top}$ in Sections 3.1, 3.2 and 3.3, respectively. We will formally state the equivalence of $\lambda^{\rightarrow \multimap \& \top}$ and $S^{\rightarrow \multimap \& \top}$ in Section 4 and prove major properties of the spine calculus in Section 5.

3.1 Syntax

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 or weak head-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 $(t\ M_1\ \ldots M_m)$ has base type. 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 term of composite type can have several heads (e.g. $\langle c_1\,\hat{}\,x, c_2\,\hat{}\,x\rangle$), 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* $S^{\rightarrow\multimap\&\top}$ 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 $U = t \cdot (U_1; \ldots U_m; \textsc{nil})$ 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 [Bar80, Her95]; its practical merits in our setting are currently assessed in an experimental implementation of a unification algorithm for *LLF* [Cer96, CP96] and a complete system for an extension of *LF*. 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.

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

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

3.2 Typing Semantics

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

$$\begin{array}{ll}
\Gamma; \Delta \vdash_\Sigma U : A & U \textit{ is a term of type } A \textit{ in } \Gamma; \Delta \textit{ and } \Sigma \\
\Gamma; \Delta \vdash_\Sigma S : A > a & S \textit{ is a spine from heads of type } A \textit{ to terms of type } a \textit{ in } \Gamma; \Delta \textit{ and } \Sigma
\end{array}$$

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: allowing arbitrary types on the right-hand side of the spine typing judgment corresponds to dropping this property. Abstract Böhm trees [Bar80, Her95] are obtained in this manner.

The mutual definition of the two typing judgments of $S^{\rightarrow\multimap\&\top}$ is given in Figure 3. The rules concerning terms resemble very closely the definition of the pre-canonical judgment of $\lambda^{\rightarrow\multimap\&\top}$, except for the treatment of heads. The rules for the spine typing judgment are instead related to pre-atomic typing in $\lambda^{\rightarrow\multimap\&\top}$. 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 Figure 3.

We conclude this section by showing that, as for $\lambda^{\rightarrow\multimap\&\top}$, the typing relation of $S^{\rightarrow\multimap\&\top}$ validates only terms in η-long form, as expressed by the lemma below.

Terms

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

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

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

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

Spines

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

(No spine rule for $\top$) $$\dfrac{\Gamma;\Delta \vdash_\Sigma S : A_1 > a}{\Gamma;\Delta \vdash_\Sigma \pi_1 S : A_1 \,\&\, A_2 > a}\,\textsf{lS\_fst} \qquad \dfrac{\Gamma;\Delta \vdash_\Sigma S : A_2 > a}{\Gamma;\Delta \vdash_\Sigma \pi_2 S : A_1 \,\&\, A_2 > a}\,\textsf{lS\_snd}$$

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

Figure 3: Typing for η-long $S^{\to\multimap\&\top}$ Terms

Lemma 3.1 (*Extensionality*)

i. *If* $\Gamma;\Delta \vdash_\Sigma U : a$, *then* $U = H \cdot S$;

ii. *If* $\Gamma;\Delta \vdash_\Sigma U : \top$, *then* $U = \langle\rangle$;

iii. *If* $\Gamma;\Delta \vdash_\Sigma U : A \,\&\, B$, *then* $U = \langle V_1, V_2\rangle$;

iv. *If* $\Gamma;\Delta \vdash_\Sigma U : A \multimap B$, *then* $U = \hat{\lambda} x{:}A.\,V$;

v. *If* $\Gamma;\Delta \vdash_\Sigma U : A \to B$, *then* $U = \lambda x{:}A.\,V$.

Proof.

By inversion on the given derivations. ☑

Notice how the structure of $S^{\to\multimap\&\top}$ terms, in particular the availability of roots, permits a leaner statement of extensionality as compared with the traditional formulation in Lemma 2.1.

3.3 Reduction Semantics

We will now concentrate on the reduction semantics of $S^{\to\multimap\&\top}$. The natural translation of the β-rules of $\lambda^{\to\multimap\&\top}$ (right) yields the *β-reductions* displayed on the left-hand side of the following table:

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

The trailing spine in the reductions for $S^{\to\multimap\&\top}$ is a consequence of the fact that this language reverses the nesting order of $\lambda^{\to\multimap\&\top}$ destructors. We call the expression patterns on the left-hand side of the arrow *β-redices*. We write $\xrightarrow{S}_\beta$ for the congruence relation based on these rules and overload this notation to

Congruences *terms*

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

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

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

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

spines

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

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

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

NIL−**reduction**

$$\frac{}{(H \cdot S) \cdot \textsc{nil} \xrightarrow{S} H \cdot S}\ \textbf{Sr\_nil}$$

β−**reductions**

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

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

Figure 4: Reduction Semantics for $S^{\rightarrow\multimap\&\top}$

apply to both terms and spines. We denote the reflexive and transitive closure of this relation as $\xrightarrow{S}{}^{*}_{\beta}$. Formal inference rules for $\xrightarrow{S}_{\beta}$ are obtained by considering the two upper and the lower segments of Figure 4.

The structure of roots in the spine calculus makes one more reduction rule necessary, namely:

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

We call this rule NIL-*reduction*, its left-hand side a NIL-*redex* and write $\xrightarrow{S}_{\textsc{nil}}$ for the congruence relation, for both terms and spines, built on top of it. It is formally defined by the topmost three parts of Figure 4. We denote its reflexive and transitive closure as $\xrightarrow{S}{}^{*}_{\textsc{nil}}$ and the corresponding equivalence relation as $\stackrel{S}{\equiv}_{\textsc{nil}}$.

We write $\xrightarrow{S}$ for the union of $\xrightarrow{S}_{\beta}$ and $\xrightarrow{S}_{\textsc{nil}}$. It is the congruence relation obtained by allowing the use of both β-reductions and the NIL-reduction. This is the relation we will use as the basis of the reduction semantics of $S^{\rightarrow\multimap\&\top}$. We reserve $\xrightarrow{S}{}^{*}$ for its reflexive and transitive closure, and $\stackrel{S}{\equiv}$ for the corresponding equivalence relation. The complete definition of $\xrightarrow{S}$ is displayed in Figure 4. As for $\lambda^{\rightarrow\multimap\&\top}$, the rules obtained from this figure by replacing $\xrightarrow{S}$ with $\xrightarrow{S}{}^{*}$ are admissible. This fact will enable us to lift every result below mentioning $\xrightarrow{S}$ (possibly as $\xrightarrow{S}_\beta$ or $\xrightarrow{S}_{\text{NIL}}$) to corresponding properties of $\xrightarrow{S}{}^*$ ($\xrightarrow{S}{}^*_\beta$ or $\xrightarrow{S}{}^*_{\text{NIL}}$, respectively).

Finally, a $S^{\rightarrow -\circ \& \top}$ term or spine that does not contain any β- or NIL-redex is called *normal*. We use instead the adjective *canonical* when this object is also in η-long form. By the above extensionality property, every well-typed normal term is canonical.

NIL-reduction appears as an omnipresent nuisance when investigating the meta-theory of $S^{\rightarrow -\circ \& \top}$ in Section 5. Fortunately, we can isolate the main properties of $\xrightarrow{S}_{\text{NIL}}$ and, by the very nature of the NIL-reduction, achieve simple proofs of these results. We will therefore dedicate the remainder of this section to this task.

The analysis of the interplay between typing and NIL-reduction reveals that this relation enjoys the subject reduction property, as stated by the following lemma. Here and below, we abbreviate the phrases "the judgment J has derivation $\mathcal{J}$" and "there is a derivation $\mathcal{J}$ of the judgment J" as $\mathcal{J} :: J$.

Lemma 3.2 (NIL-*reduction preserves typing*)

i. If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$ *and* $\mathcal{R} :: U \xrightarrow{S}_{\text{NIL}} U'$, *then* $\mathcal{U}' :: \Gamma; \Delta \vdash_\Sigma U' : A$.

ii. If $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$ *and* $\mathcal{R} :: S \xrightarrow{S}_{\text{NIL}} S'$, *then* $\mathcal{S}' :: \Gamma; \Delta \vdash_\Sigma S' : A > a$.

Proof.

By induction on the structure of $\mathcal{R}$ and inversion on $\mathcal{U}$ and $\mathcal{S}$. ☑

A further property, that we will use in Section 5 is that the use of NIL-reduction in the reverse direction, i.e., as an expansion rule, preserves typing too.

Lemma 3.3 (NIL-*expansion preserves typing*)

i. If $\mathcal{R} :: U \xrightarrow{S}_{\text{NIL}} U'$ *and* $\mathcal{U}' :: \Gamma; \Delta \vdash_\Sigma U' : A$, *then* $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$.

ii. If $\mathcal{R} :: S \xrightarrow{S}_{\text{NIL}} S'$ *and* $\mathcal{S}' :: \Gamma; \Delta \vdash_\Sigma S' : A > a$, *then* $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$.

Proof.

By induction on the structure of $\mathcal{R}$ and inversion on $\mathcal{U}'$ and $\mathcal{S}'$. ☑

We now concentrate on the properties of $S^{\rightarrow -\circ \& \top}$ and $\xrightarrow{S}_{\text{NIL}}$ as a rewriting system. An application of rule **Sr_nil** reduces a NIL-redex by eliminating a trailing NIL spine. Therefore, only as many NIL-reductions can be chained starting from a given term as the number of NIL-redices present in it. This implies that any sequence of NIL-reductions is terminating in $S^{\rightarrow -\circ \& \top}$.

Lemma 3.4 (*Strong* NIL-*normalization*)

Every maximal sequence of NIL-*reductions starting at a term U (spine S) is finite.*

Proof.

A formal proof goes by induction on the structure of U and S. ☑

This property entails also that, given a term U, there is only a finite number of terms V such that $U \xrightarrow{S}{}^*_{\text{NIL}} V$ is derivable. Therefore checking whether $U \xrightarrow{S}{}^*_{\text{NIL}} V$ has a derivation is decidable. Clearly, these results hold also for spines.

If the NIL-reduction rule is applicable in two positions in a term, the resulting terms can be reported to a common reduct by a further application (unless they are already identical). This property is formalized in the following local confluence lemma, that applies equally to terms and spines.

Lemma 3.5 (*Local confluence*)

If $\mathcal{R}' :: U \xrightarrow{S}_{\text{NIL}} U'$ *and* $\mathcal{R}'' :: U \xrightarrow{S}_{\text{NIL}} U''$, *then either* $U' = U''$ *or there is a term* V *such that* $\mathcal{R}^* :: U' \xrightarrow{S}_{\text{NIL}} V$ *and* $\mathcal{R}^{**} :: U'' \xrightarrow{S}_{\text{NIL}} V$, *and similarly for spines.*

Proof.

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

Well-known results in term rewriting theory [DJ90] allow lifting this property, in the presence of termination, to the reflexive and transitive closure of $\xrightarrow{S}_{\text{NIL}}$.

Corollary 3.6 (*Confluence*)

If $\mathcal{R}' :: U \xrightarrow{S}{}^{*}_{\text{NIL}} U'$ *and* $\mathcal{R}'' :: U \xrightarrow{S}{}^{*}_{\text{NIL}} U''$, *then there is a term* V *such that* $\mathcal{R}^* :: U' \xrightarrow{S}{}^{*}_{\text{NIL}} V$ *and* $\mathcal{R}^{**} :: U'' \xrightarrow{S}{}^{*}_{\text{NIL}} V$, *and similarly for spines.* □

We say that a term or a spine is in NIL-*normal form* if it does not contain any NIL-redex. Since $\xrightarrow{S}_{\text{NIL}}$ eliminates a NIL-redex, an exhaustive application to a term U (a spine S) yields a NIL-normal term (spine, respectively). A combination of the results above ensures that a NIL-normal form is eventually found (by the termination lemma), and that it is unique (by confluence). This is the essence of the uniqueness lemma below.

Lemma 3.7 (*Uniqueness of* NIL-*normal forms*)

For every term U *(spine* S*) there is a unique* NIL-*normal term* V *(spine* S'*) such that* $U \xrightarrow{S}{}^{*}_{\text{NIL}} V$ *(*$S \xrightarrow{S}{}^{*}_{\text{NIL}} S'$*, respectively).*

Proof.

Since $\xrightarrow{S}{}^{*}_{\text{NIL}}$ is terminating, there is at least one term V such that $U \xrightarrow{S}{}^{*}_{\text{NIL}} V$ is derivable and such that V does not admit further NIL-reductions. Then V cannot contain any NIL-redex.

Assume that there are two such term, V' and V'' say. Then by confluence, they must have a common NIL-reduct $\overline{V}$. However, since neither V' nor V'' admit NIL-reductions, it must be the case that $V' = V'' = \overline{V}$.

A similar analysis applies to spines. ☑

We denote *the* NIL-normal form of a term U and a spine S as $\text{NF}_{\text{NIL}}(U)$ and $\text{NF}_{\text{NIL}}(S)$, respectively. Furthermore, we write $S^{\rightarrow\multimap\&\top}_{\text{NIL}}$ for the sublanguage of $S^{\rightarrow\multimap\&\top}$ that consists only of NIL-normal terms.

In Section 5, we will take advantage of the following technical result that states that substitution preserves NIL-reducibility.

Lemma 3.8 (*Substitution*)

i. If $\mathcal{R} :: U \xrightarrow{S}{}^{*}_{\text{NIL}} U'$ *and* $\mathcal{R}_V :: V \xrightarrow{S}{}^{*}_{\text{NIL}} V'$, *then* $\mathcal{R}' :: [V/x]U \xrightarrow{S}{}^{*}_{\text{NIL}} [V'/x]U'$.

ii. If $\mathcal{R} :: S \xrightarrow{S}{}^{*}_{\text{NIL}} S'$ *and* $\mathcal{R}_V :: V \xrightarrow{S}{}^{*}_{\text{NIL}} V'$, *then* $\mathcal{R}' :: [V/x]S \xrightarrow{S}{}^{*}_{\text{NIL}} [V'/x]S'$.

Proof.

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

We conclude this section by analyzing how NIL-reduction interacts with the β-reduction rules of $S^{\rightarrow\multimap\&\top}$. The interesting result is that NIL-reductions can always be pushed past β-reductions, as expressed by the following lemma. We render this property graphically by means of the diagram on the right: derivations given as assumptions are represented with full lines, while derivations whose existence needs to be shown are displayed using dotted edges. For typographic reasons, we use a double arrow rather than a star (*) in order to denote the reflexive and transitive closure of a relation

Lemma 3.9 (*Postponing* NIL-*reductions*)

If $\mathcal{R}_{\text{NIL}} :: U \xrightarrow{S}_{\text{NIL}} V'$ *and* $\mathcal{R}_\beta :: V' \xrightarrow{S}_\beta V$, *then there is a term* U' *such that* $\mathcal{R}'_\beta :: U \xrightarrow{S}_\beta U'$ *and* $\mathcal{R}'_{\text{NIL}} :: U' \xrightarrow{S}{}^{*}_{\text{NIL}} V$, *and similarly for spines.*

$$\begin{tikzcd} U \arrow[r, "S", "\text{NIL}"'] \arrow[d, dotted, "S"', "\beta"] & V' \arrow[d, "S"', "\beta"] \\ U' \arrow[r, dotted, twoheadrightarrow, "S", "\text{NIL}"'] & V \end{tikzcd}$$

Proof.

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

Notice that postponing a NIL-reduction can lead to any number of instances of it, possibly zero. They should all be reduced to have the above diagram commute. Indeed, on the one hand, the application of intuitionistic β-reduction (rule **Sr_beta_int**) can result in several copies of an argument containing a NIL-redex, possibly none. On the other hand, rules **Sr_beta_fst** and **Sr_beta_snd** can project away a term containing a NIL-redex.

Observe that the dual property of pushing β-reductions past NIL-reductions does not hold in general. Consider for example the following sequence of reductions:

$$\langle c \cdot \text{NIL}, d \cdot \text{NIL}\rangle \cdot (\pi_1 \text{NIL}) \xrightarrow{S}_\beta (c \cdot \text{NIL}) \cdot \text{NIL} \xrightarrow{S}_{\text{NIL}} c \cdot \text{NIL}$$

The two reduction cannot be interchanged since the original term, FST $\langle c, d\rangle$ in the traditional notation, does not contain a NIL-redex. The problem is that, while performing a NIL-reduction does not introduce β-redices, carrying out a β-reduction can create new NIL-redices.

4 Relationship between $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$

There exists a structural translation of terms in $\lambda^{\rightarrow\multimap\&\top}$ to terms in $S^{\rightarrow\multimap\&\top}$ and vice versa. As we will see in this section, this translation preserves typing and β-reductions, so that $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$ share the same properties on well-typed (η-long) terms, and are therefore equivalent for practical purposes. However, $S^{\rightarrow\multimap\&\top}$ is structurally richer than $\lambda^{\rightarrow\multimap\&\top}$ in the sense that it permits terms containing NIL-redices, which are indistinguishable from their NIL-normal form in the more traditional formulation. Therefore, we will treat the two directions of the translation separately. In Section 4.1 we will introduce a mapping of $\lambda^{\rightarrow\multimap\&\top}$ to $S^{\rightarrow\multimap\&\top}$ and prove its soundness with respect to typing. In Section 4.2, we will instead develop the machinery to prove the soundness of this translation with respect to the reduction semantics of the two languages. We introduce the reverse translation in Section 4.3 and establish its soundness with respect to reduction in Section 4.4. Sections 4.2 and 4.4 are rather technical; the casual reader should be able to skip them and still follow the overall discussion.

4.1 λS: A Translation from $\lambda^{\rightarrow\multimap\&\top}$ to $S^{\rightarrow\multimap\&\top}$

The translation from $\lambda^{\rightarrow\multimap\&\top}$ to $S^{\rightarrow\multimap\&\top}$, abbreviated λS, maps the uniform notion of $\lambda^{\rightarrow\multimap\&\top}$ term to the roots, terms and spines of $S^{\rightarrow\multimap\&\top}$, depending on the structure of the original term. λS is specified by means of the following judgments:

$M \xrightarrow{\lambda S} U$	*M translates to U*
$M \setminus S \xrightarrow{\lambda S} U$	*M translates to U, given spine S*

The rules defining them are displayed in Figure 5. When translating a pre-atomic $\lambda^{\rightarrow\multimap\&\top}$ term M by means of the second judgment, the spine S acts as an accumulator for the destructors appearing in M. This indirection is needed to cope with the opposite associativity of spines in $S^{\rightarrow\multimap\&\top}$ and destructor nesting in $\lambda^{\rightarrow\multimap\&\top}$. The side conditions in rules λ**S_atm** and λ**S_redex** specify the admissible structure of their first argument (M); they could be avoided by specializing these rules to take into account the different possibilities they encompass. Notice that, for each of the two judgments of λS, the structure of the first argument determines uniquely which rule can be used in the translation process.

Pre-canonical terms

$$\dfrac{M \setminus \textsc{nil} \xrightarrow{\lambda S} H \cdot S}{M \xrightarrow{\lambda S} H \cdot S}\,\text{λS\_atm} \quad (\text{for } M = c,\ x,\ \textsc{fst}\ M',\ \textsc{snd}\ M',\ M' \hat{}\, M'',\ M'\ M'')$$

$$\dfrac{}{\langle\rangle \xrightarrow{\lambda S} \langle\rangle}\,\text{λS\_unit} \qquad \dfrac{M_1 \xrightarrow{\lambda S} U_1 \quad M_2 \xrightarrow{\lambda S} U_2}{\langle M_1, M_2\rangle \xrightarrow{\lambda S} \langle U_1, U_2\rangle}\,\text{λS\_pair}$$

$$\dfrac{M \xrightarrow{\lambda S} U}{\hat{\lambda} x\!:\!A.\, M \xrightarrow{\lambda S} \hat{\lambda} x\!:\!A.\, U}\,\text{λS\_llam} \qquad \dfrac{M \xrightarrow{\lambda S} U}{\lambda x\!:\!A.\, M \xrightarrow{\lambda S} \lambda x\!:\!A.\, U}\,\text{λS\_ilam}$$

Pre-atomic terms

$$\dfrac{M \xrightarrow{\lambda S} U}{M \setminus S \xrightarrow{\lambda S} U \cdot S}\,\text{λS\_redex} \quad (\text{for } M = \langle M', M''\rangle,\ \hat{\lambda} x\!:\!A.\, M',\ \lambda x\!:\!A.\, M')$$

$$\dfrac{}{c \setminus S \xrightarrow{\lambda S} c \cdot S}\,\text{λS\_con} \qquad \dfrac{}{x \setminus S \xrightarrow{\lambda S} x \cdot S}\,\text{λS\_var}$$

$$\dfrac{M \setminus \pi_1 S \xrightarrow{\lambda S} V}{\textsc{fst}\ M \setminus S \xrightarrow{\lambda S} V}\,\text{λS\_fst} \qquad \dfrac{M \setminus \pi_2 S \xrightarrow{\lambda S} V}{\textsc{snd}\ M \setminus S \xrightarrow{\lambda S} V}\,\text{λS\_snd}$$

$$\dfrac{N \xrightarrow{\lambda S} U \quad M \setminus U \,\hat{;}\, S \xrightarrow{\lambda S} V}{M \hat{}\, N \setminus S \xrightarrow{\lambda S} V}\,\text{λS\_lapp} \qquad \dfrac{N \xrightarrow{\lambda S} U \quad M \setminus U ; S \xrightarrow{\lambda S} V}{M\ N \setminus S \xrightarrow{\lambda S} V}\,\text{λS\_iapp}$$

Figure 5: Translation of $\lambda^{\rightarrow\multimap\&\top}$ into $S^{\rightarrow\multimap\&\top}$

We can immediately prove the faithfulness of this translation with respect to typing. This result expresses the adequacy of the system in Figure 3 as an emulation of the typing semantics of $\lambda^{\rightarrow\multimap\&\top}$. We will take advantage of this fact below.

Theorem 4.1 (*Soundness of λS for typing*)

i. If $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \Uparrow A$, then $\mathcal{D} :: M \xrightarrow{\lambda S} U$ and $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$;

ii. if $\mathcal{A} :: \Gamma; \Delta_1 \vdash_\Sigma M \downarrow A$ and $\mathcal{S} :: \Gamma; \Delta_2 \vdash_\Sigma S : A > a$, then $\mathcal{D} :: M \setminus S \xrightarrow{\lambda S} V$ and $\mathcal{U} :: \Gamma; \Delta_1, \Delta_2 \vdash_\Sigma V : a$.

Proof.

By simultaneous induction on the structure of $\mathcal{C}$ and $\mathcal{A}$. The cases concerning rules **lλ_atm** and **lλ_redex** require some care in order to satisfy the side conditions in rules **λS_atm** and **λS_redex**, respectively. ☑

Notice that this statement implies not only that types are preserved during the translation process, but also, by virtue of extensionality, that η-long objects of $\lambda^{\rightarrow\multimap\&\top}$ are mapped to η-long terms in the spine calculus.

We will obtain an indirect proof of the completeness of λS with respect to typing in Section 4.3. As a preparatory step, we dedicate the remainder of this section to getting some insight in the manner λS operates.

We first show that λS is a function, i.e., that every term has a unique translation. It is interesting to observe that it is not defined over all of $\lambda^{\rightarrow\multimap\&\top}$. For example, the term $\textsc{fst}\ \langle\rangle$ has no translation since no rule can derive a judgment of the form $\langle\rangle \setminus S \xrightarrow{\lambda S} U$, whichever S and U are. Although it suffices to add the possibility of having $M = \langle\rangle$ in the side condition of rule **λS_redex** to eliminate this apparent anomaly, we are not interested in such a term since it is ill-typed. Instead, we fix the domain of λS to be the set of typable (i.e., either pre-canonical or pre-atomic) terms in $\lambda^{\rightarrow\multimap\&\top}$. The following lemma establishes the functionality of the translation.

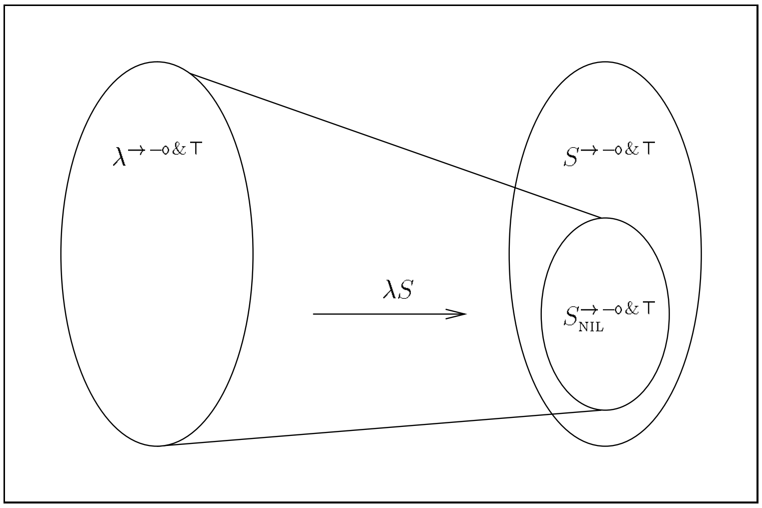

Figure 6: λS

Lemma 4.2 (*Functionality of λS*)

i. *If* $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \Uparrow A$, *then there is a unique term* U *such that* $\mathcal{T} :: M \xrightarrow{\lambda S} U$.

ii. *If* $\mathcal{A} :: \Gamma; \Delta \vdash_\Sigma M \downarrow A$ *and* $\mathcal{S} :: \Gamma; \Delta' \vdash_\Sigma S : A > a$, *then, there is a unique term* U *such that* $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} U$.

Proof.

The proof proceeds by induction on the structure of M, or equivalently on the structure of $\mathcal{C}$ and $\mathcal{A}$. The typing judgments, in particular $\mathcal{S}$, serve the only purpose of preventing considering the case $M = \langle\rangle$ in (ii), for which no rule of λS is applicable. ☑

λS translates every term in $\lambda^{\rightarrow\multimap\&\top}$ to an object in NIL-normal form. Therefore, the range of this function is the set of well-typed terms in $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$, as depicted in Figure 6, and formally stated below.

Lemma 4.3 (*Range of λS*)

i. *If* $\mathcal{T} :: M \xrightarrow{\lambda S} U$, *then* U *is in* NIL*-normal form.*

ii. *If* $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} U$ *and* S *is in* NIL*-normal form, then* U *is in* NIL*-normal form.*

Proof.

The proof proceeds by induction on the structure of $\mathcal{T}$. ☑

λS is actually a bijection between the set of well-typed terms in $\lambda^{\rightarrow\multimap\&\top}$ and the set of well-typed objects in $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$. We delay proving this property till Section 4.3, when discussing its inverse.

4.2 Soundness of λS with respect to Reduction

We have seen in the previous section that λS is sound with respect to the typing semantics of $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$. We dedicate the present section to proving that it preserves also reductions. This task is surprisingly complex for a number of reasons.

- Firstly, β-reductions in $\lambda^{\rightarrow\multimap\&\top}$ do not correspond to β-reductions in $S^{\rightarrow\multimap\&\top}$, but in general to β-reductions followed by zero or more NIL-reductions. Therefore, most statements below will mention unexpected of NIL-reductions.

- Secondly, the reduction semantics of $S^{\rightarrow -\circ \& \top}$ is specialized to η-long forms. Consider for example the $\lambda^{\rightarrow -\circ \& \top}$ term
$$M = (\lambda x\!:\!a.\, f\ x)\ c\ d$$
for appropriate declarations of f, c and d (the latter two of base type). This term is not in η-long form (its η-expansion is $(\lambda x\!:\!a.\, \lambda y\!:\!a'.\, f\ x\ y)\ c\ d$). M reduces to the canonical form
$$N = f\ c\ d$$
which translation in $S^{\rightarrow -\circ \& \top}$ is
$$V = f \cdot ((c \cdot \mathrm{NIL}); (d \cdot \mathrm{NIL}); \mathrm{NIL})$$
On the other hand, λS would translate M to the $S^{\rightarrow -\circ \& \top}$ term
$$U = (\lambda x\!:\!a.\, f \cdot (x \cdot \mathrm{NIL}); \mathrm{NIL}) \cdot ((c \cdot \mathrm{NIL}); (d \cdot \mathrm{NIL}); \mathrm{NIL})$$
which cannot be reduced further than
$$(f \cdot (c \cdot \mathrm{NIL}); \mathrm{NIL}) \cdot ((d \cdot \mathrm{NIL}); \mathrm{NIL}),$$
a different term from V. V can however be recovered by appending the spines. Such a step proved compulsory in the implementation of *LF* as the new programming language *Twelf*. Indeed, types left implicit by the user are reconstructed through unification, but since not all typing information is available at this stage, η-long forms cannot be achieved. Therefore, this preprocessing phase cannot take advantage of the strong invariants that derive from extensionality. In particular, spines occasionally need to be appended.

  As we can see from this example, λS does not commute with reduction in the general case. We can track the problem to the fact that, while β-reduction and extensionality are orthogonal concepts in $\lambda^{\rightarrow -\circ \& \top}$, they are intimately related in $S^{\rightarrow -\circ \& \top}$. Indeed, analyzing the spine calculus in the absence of extensionality requirements reveals the NIL-reduction rule as the degenerated form of a general η-expansion rule.

  However, as long as we are interested only in η-long terms, the definitions given in the previous section ensure the λS is sound with respect to the reduction semantics of our calculi. Therefore, we need to pay particular attention to operating only on η-long terms. We achieve this purpose indirectly by requiring explicitly that all the $\lambda^{\rightarrow -\circ \& \top}$ terms we consider be well-typed. This is stricter than needed, but typing is the only way we can enforce extensionality.

Rules **lr_beta_lin** and **lr_beta_int** generate their reduct means of a meta-level substitution. The corresponding reduction in $S^{\rightarrow -\circ \& \top}$ operate in a similar way. Therefore, we need to show that λS commutes reasonably well with substitution. This is achieved in the following lemma, where "reasonably well" means modulo NIL-reductions.

Lemma 4.4 (*Substitution in λS*)

i. *Assume that* $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \Uparrow A$, $\mathcal{C}_N :: \Gamma; \Delta'' \vdash_\Sigma N \Uparrow B$ *and* $x\!:\!B$ *occurs in either* Γ *or* Δ.

   If $\mathcal{T} :: M \xrightarrow{\lambda S} U$ *and* $\mathcal{T}_N :: N \xrightarrow{\lambda S} V$, *then* $\mathcal{T}' :: [N/x]M \xrightarrow{\lambda S} V'$ *where* $\mathcal{R} :: [V/x]U \xrightarrow{S}{}^{*}_{\mathrm{NIL}} V'$.

ii. *Assume that* $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \downarrow A$, $\mathcal{S} :: \Gamma; \Delta'' \vdash_\Sigma S : A > a$, $\mathcal{C}_N :: \Gamma; \Delta' \vdash_\Sigma N \Uparrow B$ *and* $x\!:\!B$ *occurs in either* Γ, Δ *or* Δ''.

   If $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} U$ *and* $\mathcal{T}_N :: N \xrightarrow{\lambda S} V$, *then* $\mathcal{T}' :: [N/x]M \setminus [V/x]S \xrightarrow{\lambda S} V'$ *where* $\mathcal{R} :: [V/x]U \xrightarrow{S}{}^{*}_{\mathrm{NIL}} V'$.

Proof.

The proof proceeds by induction on the structure of $\mathcal{T}$ and then by case distinction on the structure of $\mathcal{C}_N$. All cases are quite simple except for the situation where M is precisely x (subcase of rule **λS_var** in part (ii) of this lemma). We will analyze this situation in detail since a similar proof pattern will appear again further.

Assume therefore that

$$\mathcal{T} = \frac{}{x \setminus S \xrightarrow{\lambda S} x \cdot S}\,\lambda\mathbf{S\_var}$$

Thus, $M = x$, $U = x \cdot S$, $B = A$, $\mathcal{C}$ is either $\mathbf{l\lambda\_lvar}$, $\mathbf{l\lambda\_ivar}$, or exposes one of them after traversing alternations of instances of $\mathbf{l\lambda\_redex}$ and $\mathbf{l\lambda\_atm}$, and $\mathcal{C}_N :: \Gamma; \Delta' \vdash_\Sigma N \Uparrow A$.

We make a case distinction on the last rule applied in $\mathcal{C}_N$:

Subcase $\mathbf{l\lambda\_atm}$

Then, $A = a'$ for some base type a'. By inversion on $\mathcal{S}$, we deduce that $a' = a$ and $S = \text{NIL}$. By extensionality, we further obtain that $N = c$, $N = y$, $N = \text{FST}\, N'$, $N = \text{SND}\, N'$, $N = N'\,\hat{}\,N''$ or $N = N'\, N''$.

By inversion on rule $\lambda\mathbf{S\_atm}$ for $\mathcal{T}_N$, we have that $V = H_V \cdot S_V$ and that there is a derivation $\mathcal{T}'$ of $N \setminus \text{NIL} \xrightarrow{\lambda S} (H_V \cdot S_V)$, i.e., of $[N/x]x \setminus [V/x]\text{NIL} \xrightarrow{\lambda S} (H_V \cdot S_V)$. Now simply set $\mathcal{R}$ to

$$\frac{}{(H_V \cdot S_V) \cdot \text{NIL} \xrightarrow{S}_{\text{NIL}} H_V \cdot S_V}\,\mathbf{Sr\_beta\_nil}$$

as a derivation of $[V/x](x \cdot \text{NIL}) \xrightarrow{S}{}^{*}_{\text{NIL}} V$.

Subcase $\mathbf{l\lambda\_unit}$

Then, $A = \top$, but no rule can start a derivation of $\Gamma; \Delta'' \vdash_\Sigma S : \top > a$. Therefore, this case cannot possibly arise.

Other subcases

By inversion, $N = \langle N_1, N_2 \rangle$, $N = \hat{\lambda} y : A'. N'$ or $N = \hat{\lambda} y : A'. N'$. We apply rule $\lambda\mathbf{S\_redex}$ to $\mathcal{T}_N :: N \xrightarrow{\lambda S} V$ to obtain a derivation $\mathcal{T}'$ of $N \setminus [V/x]S \xrightarrow{\lambda S} V \cdot [V/x]S$, i.e., of $[N/x]x \setminus [V/x]S \xrightarrow{\lambda S} [V/x](x \cdot S)$. Simply take the identity as $\mathcal{R}$. ☑

We need one more technical result prior to tackling the main theorem of this section. More precisely, we need to show that, when translating a pre-atomic term, reductions to the accessory spine are mapped directly to reductions in the resulting $S^{\rightarrow -\circ \& \top}$ term, as expressed by the diagram on the right. In particular, β-reductions are mapped to β-reductions and NIL-reductions yield NIL-reductions. Notice that the statement below does not mention typing derivations. Indeed it applies to generic terms, possibly ill-typed or not in η-long form.

Lemma 4.5 (*Spine reduction*)

i. *If* $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} V$ *and* $\mathcal{R} :: S \xrightarrow{S}_\beta S'$, *then there is a term* V' *such that* $\mathcal{T}' :: M \setminus S' \xrightarrow{\lambda S} V'$ *and* $\mathcal{R}' :: V \xrightarrow{S}_\beta V'$.

ii. *If* $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} V$ *and* $\mathcal{R} :: S \xrightarrow{S}_{\text{NIL}} S'$, *then there is a term* V' *such that* $\mathcal{T}' :: M \setminus S' \xrightarrow{\lambda S} V'$ *and* $\mathcal{R}' :: V \xrightarrow{S}_{\text{NIL}} V'$.

$$\begin{tikzcd} M \setminus S \arrow[r, "\lambda S"] \arrow[d, "S"] & V \arrow[d, dotted, "S"] \\ M \setminus S' \arrow[r, dotted, "\lambda S"] & V' \end{tikzcd}$$

Proof.

This straightforward proof proceeds by induction on the structure of $\mathcal{R}$. ☑

It is easy to show that this result remains valid when considering the transitive and reflexive closure of the involved relations, or even $\xrightarrow{S}{}^{*}$.

At this point, we are in a position to prove that λS is sound with respect to the reduction semantics of $\lambda^{\rightarrow -\circ \& \top}$ and $S^{\rightarrow -\circ \& \top}$. This property is schematized by the diagram on the right.

Theorem 4.6 (*Soundness of λS for reducibility*)

$$\begin{tikzcd} M \arrow[r] \arrow[d, "\lambda S"'] & N \arrow[drr, dotted, "\lambda S"] & & \\ U \arrow[r, dotted, "S", "\beta"'] & \overline{V} \arrow[rr, dotted, two heads, "S", "\mathrm{NIL}"'] & & V \end{tikzcd}$$

i. Assume that $\mathcal{C} :: \Gamma;\Delta \vdash_\Sigma M \Uparrow A$.

If $\mathcal{R} :: M \longrightarrow N$ *and* $\mathcal{T} :: M \xrightarrow{\lambda S} U$, *then there are terms* $\overline{V}$ *and* V *such that* $\mathcal{R}_\beta :: U \xrightarrow{S}_\beta \overline{V}$, $\mathcal{R}_{\mathrm{NIL}} :: \overline{V} \xrightarrow{S}{}^{*}_{\mathrm{NIL}} V$ *and* $\mathcal{T}' :: N \xrightarrow{\lambda S} V$.

ii. Assume that $\mathcal{A} :: \Gamma;\Delta_1 \vdash_\Sigma M \downarrow A$ *and* $\mathcal{S} :: \Gamma;\Delta_2 \vdash_\Sigma S : A > a$.

If $\mathcal{R} :: M \longrightarrow N$ *and* $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} U$, *then there are terms* $\overline{V}$ *and* V *such that* $\mathcal{R}_\beta :: U \xrightarrow{S}_\beta \overline{V}$, $\mathcal{R}_{\mathrm{NIL}} :: \overline{V} \xrightarrow{S}{}^{*}_{\mathrm{NIL}} V$ *and* $\mathcal{T}' :: N \setminus S \xrightarrow{\lambda S} V$.

Proof.

The proof proceeds by induction on the structure of $\mathcal{R}$ and inversion on $\mathcal{A}$, $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{S}$. All cases are straightforward with the exception of the treatment of the β-reduction steps of $\lambda^{\rightarrow\multimap\&\top}$ (rules **lr_beta_fst**, **lr_beta_snd**, **lr_beta_lin** and **lr_beta_int**) and rules **lr_lapp2** and **lr_iapp2**, that require some care. We develop in full the cases where the last rule applied in $\mathcal{R}$ is either **lr_beta_lin** or **lr_lapp2**.

Case lr_beta_lin (i)

Then

$$\mathcal{R} = \frac{}{(\hat\lambda x:A'.\,M')\,\hat{}\,M'' \;\longrightarrow\; [M''/x]M'}$$

where $M = (\hat\lambda x:A'.\,M')\,\hat{}\,M''$ and $N = [M''/x]M'$.

By inversion over $\mathcal{C}$ (glossing over possible alternations of rules **lλ_redex** and **lλ_atm**), we have that

$$\mathcal{C} = \dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{C}_1\\ \Gamma;\Delta',x:A' \vdash_\Sigma M' \Uparrow a\end{array}}{\Gamma;\Delta' \vdash_\Sigma \hat\lambda x:A'.\,M' \Uparrow A' \multimap a}\,\mathbf{l\lambda\_llam}}{\Gamma;\Delta' \vdash_\Sigma \hat\lambda x:A'.\,M' \downarrow A' \multimap a}\,\mathbf{l\lambda\_redex} \qquad \begin{array}{c}\mathcal{C}_2\\ \Gamma;\Delta'' \vdash_\Sigma M'' \Uparrow A'\end{array}}{\Gamma;\Delta',\Delta'' \vdash_\Sigma (\hat\lambda x:A'.\,M')\,\hat{}\,M'' \downarrow a}\,\mathbf{l\lambda\_lapp}}{\Gamma;\Delta',\Delta'' \vdash_\Sigma (\hat\lambda x:A'.\,M')\,\hat{}\,M'' \Uparrow a}\,\mathbf{l\lambda\_atm}$$

Notice in particular that A is an atomic type a, and that the subterm M' has precisely this type.

We can similarly invert $\mathcal{T}$ obtaining the following partially expanded tree:

$$\mathcal{T} = \dfrac{\dfrac{\begin{array}{c}\mathcal{T}_2\\ M'' \xrightarrow{\lambda S} U''\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{T}_1\\ M' \xrightarrow{\lambda S} U'\end{array}}{\hat\lambda x:A'.\,M' \xrightarrow{\lambda S} \hat\lambda x:A'.\,U'}\,\mathbf{\lambda S\_llam}}{\hat\lambda x:A'.\,M' \setminus U''\,\hat{;}\,\mathrm{NIL} \xrightarrow{\lambda S} (\hat\lambda x:A'.\,U') \cdot (U''\,\hat{;}\,\mathrm{NIL})}\,\mathbf{\lambda S\_redex}}{(\hat\lambda x:A'.\,M')\,\hat{}\,M'' \setminus \mathrm{NIL} \xrightarrow{\lambda S} (\hat\lambda x:A'.\,U') \cdot (U''\,\hat{;}\,\mathrm{NIL})}\,\mathbf{\lambda S\_lapp}}{(\hat\lambda x:A'.\,M')\,\hat{}\,M'' \xrightarrow{\lambda S} (\hat\lambda x:A'.\,U') \cdot (U''\,\hat{;}\,\mathrm{NIL})}\,\mathbf{\lambda S\_atm}$$

By extensionality relative to $\mathcal{C}_1$, M' is either a constant, a variable or a destructor applied to some subterm. Therefore, by inversion on $\mathcal{T}_1$, we have that $U' = H' \cdot S'$ for some head H' and spine S'. Now, by definition of substitution,

$$[U''/x](H' \cdot S') = ([U''/x]H') \cdot ([U''/x]S')$$

so that NIL-reduction can be applied to $([U''/x]U') \cdot \text{NIL}$. By chaining rules **Sr_beta_lin** and **Sr_nil**, we get

$$(\hat{\lambda} x : A'.\, U') \cdot (U'' \,\hat{;}\, \text{NIL}) \xrightarrow{S}_{\beta} ([U''/x]U') \cdot \text{NIL} \xrightarrow{S}_{\text{NIL}} [U''/x]U'$$

By the substitution lemma 4.4 on $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{T}_1$ and $\mathcal{T}_2$, we know that there is a term V such that $[M''/x]M' \xrightarrow{\lambda S} V$ where $[U''/x]U' \xrightarrow{S}^{*}_{\text{NIL}} V$. It now suffices to take $([U''/x]U') \cdot \text{NIL}$ as $\overline{V}$.

Case lr_beta_lin (ii)

Inversion on $\mathcal{A}$ and $\mathcal{T}$ yields the fact that $U = (\hat{\lambda} x : A'.\, U') \cdot (U'' \,\hat{;}\, S)$ for some terms U' and U'', and derivations

$$\begin{aligned}
&\mathcal{C}_1 :: \Gamma; \Delta_1', x : A' \vdash_{\Sigma} M' \Uparrow A \\
&\mathcal{C}_2 :: \Gamma; \Delta_1'' \vdash_{\Sigma} M'' \Uparrow A' \\
&\mathcal{T}_1 :: M' \xrightarrow{\lambda S} U' \\
&\mathcal{T}_2 :: M'' \xrightarrow{\lambda S} U''
\end{aligned}$$

where $\Delta_1 = \Delta_1', \Delta_1''$.

By applying the substitution lemma 4.4 on these derivations, there is a term V' such that both $[M''/x]M' \xrightarrow{\lambda S} V'$ and $[U''/x]U' \xrightarrow{S}^{*}_{\text{NIL}} V'$ are derivable with derivations $\mathcal{T}''$ and $\mathcal{R}'$, respectively. At this point we proceed by cases on the structure of $\mathcal{T}''$ using the same technique already employed in the proof fragment we showed earlier for the substitution lemma itself.

Subcase lλ_atm

Then $[M''/x]M'$ is one of c, y, $\text{FST}\ N'$, $\text{SND}\ N'$, $N' \hat{\ } N''$ or $N'\ N''$. By the transitivity lemma 2.3 on $\mathcal{C}_1$ and $\mathcal{C}_2$, there is a derivation of $\Gamma; \Delta_1', \Delta_1'' \vdash_{\Sigma} [M''/x]M' \Uparrow A$. By inversion, it must be the case that $A = a$, $S = \text{NIL}$ and $U' = H' \cdot S'$ for some head H' and spine S'. Therefore, by inversion on rule **λS_atm** for $\mathcal{T}''$ we obtain the desired derivation $\mathcal{T}'$ of $[M''/x]M' \setminus \text{NIL} \xrightarrow{\lambda S} V'$. On the other hand, we can build the following chain of reductions:

$$(\hat{\lambda} x : A'.\, U') \cdot (U'' \,\hat{;}\, \text{NIL}) \xrightarrow{S}_{\beta} ([U''/x]U') \cdot \text{NIL} \xrightarrow{S}_{\text{NIL}} [U''/x]U' \xrightarrow{S}^{*}_{\text{NIL}} V'$$

Here, we need to take $([U''/x]U') \cdot \text{NIL}$ as $\overline{V}$ and V' as V.

Subcase lλ_unit

This case does not apply.

Other subcases

Then $[M''/x]M'$ has the form $\langle N', N'' \rangle$, $\hat{\lambda} y : A''.\, N'$ or $\hat{\lambda} y : A''.\, N'$. We can therefore apply rule **λS_redex** to $\mathcal{T}''$, obtaining the desired translation $\mathcal{T}'$ of $[M''/x]M' \setminus S \xrightarrow{\lambda S} V' \cdot S$. Moreover, by chaining **Sr_beta_lin** to $\mathcal{R}'$ (modulo embedded applications of **Sr_redex1**), we obtained the required reduction $\mathcal{R}$:

$$(\hat{\lambda} x : A'.\, U') \cdot (U'' \,\hat{;}\, S) \xrightarrow{S}_{\beta} ([U''/x]U') \cdot S \xrightarrow{S}_{\text{NIL}} V' \cdot S$$

In this case, $V = V' \cdot S$ and $\overline{V} = ([U''/x]U') \cdot S$.

Case lr_lapp2

We will focus on proving part (ii) of the theorem. Part (i) combines the technique to be shown with the reasoning pattern used above.

We have therefore that $M = M' \hat{\ } M''$ and

$$\mathcal{R} = \dfrac{\begin{matrix}\mathcal{R}_1 \\ M'' \xrightarrow{\lambda S} N''\end{matrix}}{M' \hat{\ } M'' \xrightarrow{\lambda S} M' \hat{\ } N''}\ \textbf{lr\_lapp2}$$

where $N = M' \hat{\ } N''$.

By inversion on $\mathcal{A}$ and $\mathcal{T}$, we obtain that

$$\mathcal{A} = \dfrac{\begin{array}{cc}\mathcal{A}_1 & \mathcal{C}_2 \\ \Gamma; \Delta_1' \vdash_\Sigma M' \downarrow A' \multimap A & \Gamma; \Delta_1'' \vdash_\Sigma M'' \Uparrow A'\end{array}}{\Gamma; \Delta_1', \Delta_1'' \vdash_\Sigma M' \hat{\ } M'' \downarrow A} \, \mathbf{l\lambda\_lapp}$$

and

$$\mathcal{T} = \dfrac{\begin{array}{cc}\mathcal{T}_2 & \mathcal{T}_1 \\ M'' \xrightarrow{\lambda S} U'' & M' \setminus U'' \,\hat{;}\, S \xrightarrow{\lambda S} U\end{array}}{M' \hat{\ } M'' \setminus S \xrightarrow{\lambda S} U} \, \mathbf{l\lambda\_lapp}$$

By induction hypothesis (i) on $\mathcal{R}_1$, $\mathcal{C}_2$ and $\mathcal{T}_2$, there are terms V'' and $\overline{V}''$ and derivations $\mathcal{R}''_\beta ::$ $U'' \xrightarrow{S}_\beta \overline{V}''$, $\mathcal{R}''_{\text{NIL}} :: \overline{V}'' \xrightarrow{S}^*_{\text{NIL}} V''$ and $\mathcal{T}'' :: N'' \xrightarrow{\lambda S} V''$.

Applying rule **Sr_lapp1** to $\mathcal{R}''_\beta$ and $\mathcal{R}''_{\text{NIL}}$, we obtain derivations $\mathcal{R}'''_\beta$ and $\mathcal{R}'''_{\text{NIL}}$ of $U'' \cdot S \xrightarrow{S}_\beta \overline{V}'' \cdot S$ and $\overline{V}'' \cdot S \xrightarrow{S}^*_{\text{NIL}} V'' \cdot S$, respectively.

By the spine reduction lemma 4.5 on $\mathcal{T}_1$ and $\mathcal{R}'''_\beta$, there is a term $\overline{V}$ such that $\mathcal{T}''' :: M'' \setminus \overline{V}'' \cdot S \xrightarrow{\lambda S} \overline{V}$ and $\mathcal{R}_\beta :: U \xrightarrow{S}_\beta \overline{V}$ are derivable. By a further application of the spine reduction lemma on $\mathcal{T}'''$ and $\mathcal{R}'''_{\text{NIL}}$, we find a term V such that $\mathcal{T}^* :: M'' \setminus V'' \cdot S \xrightarrow{\lambda S} V$ and $\mathcal{R}_{\text{NIL}} :: \overline{V} \xrightarrow{S}^*_{\text{NIL}} V$ are derivable.

Having the desired terms and reduction derivations, we obtain the required translation derivation $\mathcal{T}' :: M' \hat{\ } M'' \setminus S \xrightarrow{\lambda S} V$ by applying rule **λS_lapp** to $\mathcal{T}''$ and $\mathcal{T}^*$. ☑

The postponement lemma allows us to lift this result to the reflexive and transitive closures of the mentioned reduction relations.

The notion of soundness we adopted relatively to the reduction semantics of our calculi requires that every reduction in the source language correspond to one (or more) reductions in the target language. We define completeness dually: every reduction in the target language should correspond to some reduction in the source language, possibly none. We will give an indirect proof of the completeness of λS with respect to the reduction semantics of our calculi in Section 4.4, when considering the inverse of our translation.

4.3 $S\lambda$: A Translation from $S^{\rightarrow\multimap\&\top}$ to $\lambda^{\rightarrow\multimap\&\top}$

In this section and in the next, we consider the problem of translating terms from $S^{\rightarrow\multimap\&\top}$ back to $\lambda^{\rightarrow\multimap\&\top}$, an essential operation to interpret $S^{\rightarrow\multimap\&\top}$ objects in the usual notation. λS cannot be used for this purpose since its codomain is $S^{\rightarrow\multimap\&\top}_{\text{NIL}}$, the subset of $S^{\rightarrow\multimap\&\top}$ consisting only of NIL-normal forms. Moreover, it would be impractical even for NIL-normal terms since the rules in Figure 5 are not syntax-directed with respect to the $S^{\rightarrow\multimap\&\top}$ objects they mention.

The approach we take is instead to define an independent translation, $S\lambda$, that maps entities in $S^{\rightarrow\multimap\&\top}$ to terms in $\lambda^{\rightarrow\multimap\&\top}$. We will prove later that it is precisely the inverse of λS, modulo details. $S\lambda$ is specified by means of the judgments

$U \xrightarrow{S\lambda} M$	*U translates to M*
$S \setminus M \xrightarrow{S\lambda} N$	*S translates to N, given seed M*

and defined in Figure 7. The notion of spine does not have a proper equivalent in $\lambda^{\rightarrow\multimap\&\top}$: it corresponds indeed to a term with a *hole* as its head. Therefore, when translating a spine, we need to supply a head in order to generate a meaningful $\lambda^{\rightarrow\multimap\&\top}$ term. This is achieved by the judgment $S \setminus M \xrightarrow{S\lambda} N$: the auxiliary term M (the *seed*) is initialized to the translation of some head for the spine S (rules **Sλ_con**, **Sλ_var** and **Sλ_redex**); it is successively used as an accumulator for the translation of the operators

Terms

$$\frac{S \setminus c \xrightarrow{S\lambda} M}{c \cdot S \xrightarrow{S\lambda} M}\ \textbf{S}\lambda\textbf{\_con} \qquad \frac{S \setminus x \xrightarrow{S\lambda} M}{x \cdot S \xrightarrow{S\lambda} M}\ \textbf{S}\lambda\textbf{\_var} \qquad \frac{U \xrightarrow{S\lambda} M' \quad S \setminus M' \xrightarrow{S\lambda} M}{U \cdot S \xrightarrow{S\lambda} M}\ \textbf{S}\lambda\textbf{\_redex}$$

$$\frac{}{\langle\rangle \xrightarrow{S\lambda} \langle\rangle}\ \textbf{S}\lambda\textbf{\_unit} \qquad \frac{U_1 \xrightarrow{S\lambda} M_1 \quad U_2 \xrightarrow{S\lambda} M_2}{\langle U_1, U_2\rangle \xrightarrow{S\lambda} \langle M_1, M_2\rangle}\ \textbf{S}\lambda\textbf{\_pair}$$

$$\frac{U \xrightarrow{S\lambda} M}{\hat{\lambda} x{:}A.\,U \xrightarrow{S\lambda} \hat{\lambda} x{:}A.\,M}\ \textbf{S}\lambda\textbf{\_llam} \qquad \frac{U \xrightarrow{S\lambda} M}{\lambda x{:}A.\,U \xrightarrow{S\lambda} \lambda x{:}A.\,M}\ \textbf{S}\lambda\textbf{\_ilam}$$

Spines

$$\frac{}{\textsc{nil} \setminus M \xrightarrow{S\lambda} M}\ \textbf{S}\lambda\textbf{\_nil}$$

$$\frac{S \setminus \textsc{fst}\ M \xrightarrow{S\lambda} N}{\pi_1 S \setminus M \xrightarrow{S\lambda} N}\ \textbf{S}\lambda\textbf{\_fst} \qquad \frac{S \setminus \textsc{snd}\ M \xrightarrow{S\lambda} N}{\pi_2 S \setminus M \xrightarrow{S\lambda} N}\ \textbf{S}\lambda\textbf{\_snd}$$

$$\frac{U \xrightarrow{S\lambda} M' \quad S \setminus M\,\hat{}\,M' \xrightarrow{S\lambda} N}{U \mathbin{\hat{;}} S \setminus M \xrightarrow{S\lambda} N}\ \textbf{S}\lambda\textbf{\_lapp} \qquad \frac{U \xrightarrow{S\lambda} M' \quad S \setminus M\ M' \xrightarrow{S\lambda} N}{U ; S \setminus M \xrightarrow{S\lambda} N}\ \textbf{S}\lambda\textbf{\_iapp}$$

Figure 7: Translation of $S^{\to\multimap\&\top}$ into $\lambda^{\to\multimap\&\top}$

appearing in S (rules **Sλ_fst**, **Sλ_snd**, **Sλ_lapp** and **Sλ_iapp**); when the empty spine is eventually reached (rule **Sλ_nil**), the overall translation has been completed and M is returned. As in λS, the use of an accumulator handles the opposite associativity of $S^{\to\multimap\&\top}$ and $\lambda^{\to\multimap\&\top}$.

The faithfulness of $S\lambda$ with respect to typing is formally expressed by the following theorem. Again, we shall stress the fact that the translation process preserves not only types, but also extensionality.

Theorem 4.7 (*Soundness of $S\lambda$ for typing*)

i. If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$, then $\mathcal{Q} :: U \xrightarrow{S\lambda} M$ and $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \Uparrow A$.

ii. If $\mathcal{S} :: \Gamma; \Delta_1 \vdash_\Sigma S : A > a$ and $\mathcal{A} :: \Gamma; \Delta_2 \vdash_\Sigma M \downarrow A$, then $\mathcal{Q} :: S \setminus M \xrightarrow{S\lambda} N$ and $\mathcal{U} :: \Gamma; \Delta_1, \Delta_2 \vdash_\Sigma N \Uparrow a$.

Proof.

By simultaneous induction on the structure of $\mathcal{U}$ and $\mathcal{S}$. ☑

We dedicate the remainder of this section to proving that $S\lambda$ is the inverse of λS. Besides getting the comforting formal acknowledgment that our two translations do behave as expected, we will take advantage of this result to obtain straightforward proofs of the completeness of λS and $S\lambda$ with respect to typing and reduction.

We begin our endeavor by proving that $S\lambda$ is actually a function from $S^{\to\multimap\&\top}$ to $\lambda^{\to\multimap\&\top}$. Notice that the statement of the lemma below does not mention any typing information (compare it with Lemma 4.2). Indeed, $S\lambda$ operates properly also on aberrant terms such as $\langle\rangle \cdot \pi_1\textsc{nil}$, which is mapped to $\textsc{fst}\ \langle\rangle$ (remember that λS was ineffective on this term). It has $S^{\to\multimap\&\top}$ as a whole as its domain.

Lemma 4.8 (*Functionality of $S\lambda$*)

i. For every $S^{\to\multimap\&\top}$ term U, there is a unique $\lambda^{\to\multimap\&\top}$ term M such that $U \xrightarrow{S\lambda} M$.

ii. For every spine S and seed M, there is a unique $\lambda^{\to\multimap\&\top}$ term N such that $S \setminus M \xrightarrow{S\lambda} N$.

Proof.

By induction on the structure of U and S. ☑

We wish $S\lambda$ to be the inverse of λS. Although this property does not hold in its full strength, it is "true enough" so that we can take practical advantage of it. The problem is that these two functions have different domains and ranges. Indeed, not only does λS operate exclusively on well-typed $\lambda^{\rightarrow\multimap\&\top}$ terms, but it produces elements in $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$, a strict subset or $S^{\rightarrow\multimap\&\top}$. On the other hand, $S\lambda$ accepts arbitrary terms in $S^{\rightarrow\multimap\&\top}$. We bridge these differences in the lemma below by insisting on well-typed terms and relying on NIL-reduction.

Lemma 4.9 (*Invertibility*)

$$\begin{tikzcd} U \arrow[rr, dotted, "S" above, "\text{NIL}" below] \arrow[dr, "S\lambda"'] & & V \\ & M \arrow[ur, dotted, "\lambda S"'] & \end{tikzcd}$$

i. Assume that $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$.

If $\mathcal{Q} :: U \xrightarrow{S\lambda} M$, *then* $\mathcal{T} :: M \xrightarrow{\lambda S} V$ *where* $\mathcal{R} :: U \xrightarrow{S}{}^{*}_{\text{NIL}} V$.

ii. Assume that $\mathcal{S} :: \Gamma; \Delta_1 \vdash_\Sigma S : A > a$ *and* $\mathcal{A} :: \Gamma; \Delta_2 \vdash_\Sigma N \downarrow A$.

If $\mathcal{Q} :: S \setminus N \xrightarrow{S\lambda} M$ *and* $\mathcal{T}_N :: N \setminus S \xrightarrow{\lambda S} U$, *then* $\mathcal{T} :: M \xrightarrow{\lambda S} V$ *where* $\mathcal{R} :: U \xrightarrow{S}{}^{*}_{\text{NIL}} V$.

Proof.

The proof proceeds by induction on the structure of $\mathcal{Q}$ and inversion on the other given derivations. We rely on the same reasoning pattern already used in the proofs of the substitution lemma for λS (Lemma 4.4) and in the soundness theorem 4.6. The most complex cases involve rules **Sλ_redex** and **Sλ_nil** and the application rules. ☑

The reverse of this property holds in a much stronger sense: not only no typing information is needed, but translating a $\lambda^{\rightarrow\multimap\&\top}$ term to $S^{\rightarrow\multimap\&\top}$ and then back yields the very same original term. We have the following untyped invertibility lemma.

Lemma 4.10 (*Untyped invertibility*)

i. If $\mathcal{T} :: M \xrightarrow{\lambda S} U$, *then* $\mathcal{Q} :: U \xrightarrow{S\lambda} M$.

ii. If $\mathcal{T} :: M \setminus S \xrightarrow{\lambda S} V$ *and* $\mathcal{Q}_S :: S \setminus M \xrightarrow{S\lambda} N$, *then* $\mathcal{Q} :: V \xrightarrow{S\lambda} N$.

Proof.

By inversion on the structure of $\mathcal{T}$. ☑

The untyped invertibility lemma states that composing $S\lambda$ with λS transforms a $\lambda^{\rightarrow\multimap\&\top}$ term to itself; therefore it corresponds to the identity function on $\lambda^{\rightarrow\multimap\&\top}$. On the other hand, the invertibility lemma 4.9 states that $S\lambda$ is the left inverse of λS on well-typed $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$ terms. On the basis of this observation and of previously proved properties, we easily deduce that they form a pair of inverse functions between the well-typed fragments of $\lambda^{\rightarrow\multimap\&\top}$ and $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$.

Corollary 4.11 (*Bijectivity*)

λS and $S\lambda$ are bijections between the set of well-typed $\lambda^{\rightarrow\multimap\&\top}$ terms and the set of well-typed $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$ terms. Moreover, they are each other's inverse.

Proof.

It is an easy exercise in abstract algebra to show that, given two functions $f : X \rightarrow Y$ and $g : Y \rightarrow X$, if $f \circ g = \text{Id}_Y$ and $g \circ f = \text{Id}_X$, then f and g are bijections and moreover $g = f^{-1}$.

By Lemmas 4.2, 4.3 and theorem 4.1, we know that λS is a function from the well-typed portion of $\lambda^{\rightarrow\multimap\&\top}$ to the well-typed subset of $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$. By the functionality lemma 4.8, $S\lambda$ maps $S^{\rightarrow\multimap\&\top}$ terms to $\lambda^{\rightarrow\multimap\&\top}$ objects; in particular, by typing soundness, it associates well-typed NIL-normal $S^{\rightarrow\multimap\&\top}$ terms to well-typed $\lambda^{\rightarrow\multimap\&\top}$ terms. Moreover, since terms that are already NIL-normal cannot be further NIL-reduced, the invertibility lemma states that $S\lambda$ is the left inverse of λS on well-typed $S_{\text{NIL}}^{\rightarrow\multimap\&\top}$ terms. Finally, by the untyped invertibility lemma, λS is the left inverse of $S\lambda$ on $\lambda^{\rightarrow\multimap\&\top}$, and in particular on its well-typed fragment.

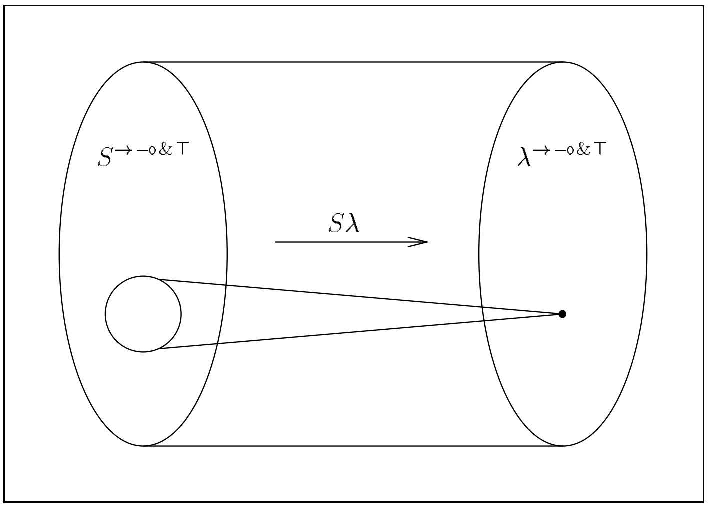

Figure 8: $S\lambda$

On the basis of these hypotheses, the previous algebraic observation allows us to conclude that λS and $S\lambda$ are indeed bijections between well-typed objects in $\lambda^{\to\multimap\&\top}$ and well-typed terms in $S_{\text{NIL}}^{\to\multimap\&\top}$, and that they are one another's inverse. ☑

This property opens the door to easy proofs of the completeness direction of every soundness theorem so painfully achieved so far. We first consider the completeness of λS with respect to typing. In this and other results below, we do not need to present any auxiliary part related to pre-atomic terms.

Corollary 4.12 (*Completeness of λS for typing*)

If $M \xrightarrow{\lambda S} U$ *and* $\Gamma; \Delta \vdash_\Sigma U : A$, *then* $\Gamma; \Delta \vdash_\Sigma M \Uparrow A$.

Proof.

By the untyped invertibility lemma, $U \xrightarrow{S\lambda} M$. Then, the soundness of $S\lambda$ for typing yields a derivation of $\Gamma; \Delta \vdash_\Sigma M \Uparrow A$. ☑

An implementation that relies on $S^{\to\multimap\&\top}$ as its internal representation of $\lambda^{\to\multimap\&\top}$ terms would translate these terms as it parses them and only then check that they are well typed. The novel *Twelf* implementation of *LF* [HHP93] takes precisely these steps. The above corollary decrees that this way of proceeding is correct since if λS produces a well-typed term, then the original $\lambda^{\to\multimap\&\top}$ object is well-typed.

In a similar fashion, we prove the completeness of $S\lambda$ with respect to typing.

Corollary 4.13 (*Completeness of $S\lambda$ for typing*)

If $U \xrightarrow{S\lambda} M$ *and* $\Gamma; \Delta \vdash_\Sigma M \Uparrow A$, *then* $\Gamma; \Delta \vdash_\Sigma U : A$.

Proof.

By the invertibility lemma, $M \xrightarrow{\lambda S} V$ where $U \xrightarrow{S}{}^{*}_{\text{NIL}} V$. By the soundness of λS for typing, we obtain that $\Gamma; \Delta \vdash_\Sigma V : A$. Finally, since NIL-expansion preserves typing (Lemma 3.3), we have that $\Gamma; \Delta \vdash_\Sigma U : A$. ☑

4.4 Soundness of $S\lambda$ with respect to Reduction

We will now analyze the interaction between $S\lambda$ as a translation from $S^{\to\multimap\&\top}$ and $\lambda^{\to\multimap\&\top}$, and the notion of reduction inherent to these two languages. The main results of our investigation will be that

$S\lambda$ preserves β-reductions, but identifies NIL-convertible terms. We will also take advantage of the fact that this translation is the inverse of λS to prove the completeness counterpart of these statements.

It will be convenient to start by getting a deeper understanding of how NIL-reducibility relates to $S\lambda$. Consider the equivalence relation $\overset{S}{\equiv}_{\text{NIL}}$ induced by the NIL-reduction congruence $\xrightarrow{S}_{\text{NIL}}$. Its equivalence classes consist of all the terms of $S^{\rightarrow\multimap\&\top}$ that NIL-reduce to the same NIL-normal form. $S\lambda$ uniformly maps every object in such an equivalence class to the same $\lambda^{\rightarrow\multimap\&\top}$ term, as depicted in Figure 8. In order to prove this fact, we first show that NIL-reducing a term does not affect its translation.

Lemma 4.14 (*Invariance of $S\lambda$ under* NIL*-reduction*)

i. If $\mathcal{R} :: U \xrightarrow{S}_{\text{NIL}} U'$ *and* $\mathcal{Q} :: U \xrightarrow{\text{S}\lambda} M$*, then* $\mathcal{Q}' :: U' \xrightarrow{\text{S}\lambda} M$.

ii. If $\mathcal{R} :: S \xrightarrow{S}_{\text{NIL}} S'$ *and* $\mathcal{Q} :: S \setminus N \xrightarrow{\text{S}\lambda} M$*, then* $\mathcal{Q}' :: S' \setminus N \xrightarrow{\text{S}\lambda} M$.

$$\begin{tikzcd} U \arrow[rr, "S"] \arrow[rr, "\text{NIL}"'] \arrow[dr, "S\lambda"'] & & U' \arrow[dl, dotted, "S\lambda"] \\ & M & \end{tikzcd}$$

Proof.

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

This lemma can also be interpreted as stating that $S\lambda$ is sound with respect to NIL-reducibility. Therefore, in the following discussion, we will concentrate on the interaction between this translation and the proper β-reductions of $S^{\rightarrow\multimap\&\top}$.

The converse of the above property holds also: $S\lambda$ maps a term and all of its NIL-expansions to the same $\lambda^{\rightarrow\multimap\&\top}$ object. This is formally stated as follows.

Lemma 4.15 (*Invariance of $S\lambda$ under* NIL*-expansion*)

i. If $\mathcal{R} :: U \xrightarrow{S}_{\text{NIL}} U'$ *and* $\mathcal{Q}' :: U' \xrightarrow{\text{S}\lambda} M$*, then* $\mathcal{Q} :: U \xrightarrow{\text{S}\lambda} M$.

ii. If $\mathcal{R} :: S \xrightarrow{S}_{\text{NIL}} S'$ *and* $\mathcal{Q}' :: S' \setminus N \xrightarrow{\text{S}\lambda} M$*, then* $\mathcal{Q} :: S \setminus N \xrightarrow{\text{S}\lambda} M$.

$$\begin{tikzcd} U \arrow[rr, "S"] \arrow[rr, "\text{NIL}"'] \arrow[dr, dotted, "S\lambda"'] & & U' \arrow[dl, "S\lambda"] \\ & M & \end{tikzcd}$$

Proof.

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

Strong NIL-normalization (Lemma 3.7) enables to easily shift these properties to the reflexive and transitive closure of $\xrightarrow{S}_{\text{NIL}}$, and to the corresponding equivalence relation.

The NIL-invariance properties we just achieved together with the discovery in the previous section that λS and $S\lambda$ are weakly bijective account for a simple proof of the completeness of the latter translation with respect to the reduction semantics of the involved calculi.

Corollary 4.16 (*Completeness of $S\lambda$ for reduction*)

Assume that $\Gamma; \Delta \vdash_\Sigma U : A$.

If $U \xrightarrow{\text{S}\lambda} M$ *and* $M \longrightarrow N$*, then there is a term* V *such that* $U \xrightarrow{S}_\beta V$ *and* $V \xrightarrow{\text{S}\lambda} N$.

$$\begin{tikzcd} U \arrow[r, dotted, "S"] \arrow[r, dotted, "\beta"'] \arrow[d, "S\lambda"'] & V \arrow[d, dotted, "S\lambda"] \\ M \arrow[r] & N \end{tikzcd}$$

Proof.

By the invertibility lemma 4.9, there is a NIL-normal term U' such that $M \xrightarrow{\lambda S} U'$ and $U \xrightarrow{S}{}^*_{\text{NIL}} U'$ are derivable. By the soundness of λS with respect to reduction, there are terms $\overline{V}'$ and V' such that

$$U' \xrightarrow{S}_\beta \overline{V}' \xrightarrow{S}{}^*_{\text{NIL}} V' \qquad \text{and} \qquad N \xrightarrow{\lambda S} V'.$$

By virtue of Lemma 3.9, we can postpone the NIL-reductions that lead from U to U', obtaining a term V such that

$$U \xrightarrow{S}_\beta V \xrightarrow{S}{}^*_{\text{NIL}} V'$$

On the other hand, by untyped invertibility, there is a derivation of $V' \xrightarrow{S\lambda} N$. At this point, an iterated use of the invariance of $S\lambda$ under NIL-expansion (Lemma 4.15) allows us to obtain the desired derivation of $V \xrightarrow{S\lambda} N$. ☑

We conclude this section by showing that $S\lambda$ is sound with respect to the reduction semantics of $S^{\rightarrow\multimap\&\top}$. The above invariance lemmas capture this property in the case of NIL-reduction. Therefore, we focus the discussion on β-reductions.

The required steps in order to achieve this result are reminiscent of the path we followed when proving the analogous statement for λS. There are however three important differences. First, the proofs are much simpler in the present case. Second, the statements below do not need to mention any typing information. Third, NIL-reductions do not appear in these statements. This overall simplification derives from the fact that, because of the presence of NIL-reduction, $S^{\rightarrow\multimap\&\top}$ has more structure than $\lambda^{\rightarrow\multimap\&\top}$. Therefore, while λS needed to extract the additional information from a typing derivation, $S\lambda$ can simply forget about the extra structure of the $S^{\rightarrow\multimap\&\top}$ terms it acts upon.

The first step towards the soundness of $S\lambda$ with respect to (β-)reduction is given by the following substitution lemma, needed to cope with functional objects, both linear and intuitionistic.

Lemma 4.17 (*Substitution in $S\lambda$*)

i. If $\mathcal{Q} :: U \xrightarrow{S\lambda} M$ and $\mathcal{Q}_V :: V \xrightarrow{S\lambda} N$, then $\mathcal{Q}' :: [V/x]U \xrightarrow{S\lambda} [N/x]M$.

ii. If $\mathcal{Q} :: S \setminus M \xrightarrow{S\lambda} M'$ and $\mathcal{Q}_V :: V \xrightarrow{S\lambda} N$, then $\mathcal{Q}' :: [V/x]S \setminus [N/x]M \xrightarrow{S\lambda} [N/x]M'$.

Proof.

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

In order to handle the translation rules for the two forms of application of $S^{\rightarrow\multimap\&\top}$, we need the following technical result, akin to the spine reduction lemma presented in Section 4.2.

Lemma 4.18 (*Seed reduction*)

If $\mathcal{Q} :: S \setminus M \xrightarrow{S\lambda} N$ and $\mathcal{R} :: M \longrightarrow M'$, then there is a term N' such that $\mathcal{Q}' :: S \setminus M' \xrightarrow{S\lambda} N'$ and $\mathcal{R}' :: N \longrightarrow N'$.

$$\begin{tikzcd} S \setminus M \arrow[r, "S\lambda"] \arrow[d] & N \arrow[d, dotted] \\ S \setminus M' \arrow[r, dotted, "S\lambda"] & N' \end{tikzcd}$$

Proof.

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

Finally, we have the following soundness theorem, that states that $S\lambda$ preserves β-reduction.

Theorem 4.19 (*Soundness of $S\lambda$ for β-reducibility*)

i. If $\mathcal{R} :: U \xrightarrow{S}_{\beta} U'$ and $\mathcal{Q} :: U \xrightarrow{S\lambda} M$, then there is a term M' such that $\mathcal{R}' :: M \longrightarrow M'$ and $\mathcal{Q}' :: U' \xrightarrow{S\lambda} M'$.

ii. If $\mathcal{R} :: S \xrightarrow{S}_{\beta} S'$ and $\mathcal{Q} :: S \setminus N \xrightarrow{S\lambda} M$, then there is a term M' such that $\mathcal{R}' :: M \longrightarrow M'$ and $\mathcal{Q}' :: S' \setminus N \xrightarrow{S\lambda} M'$.

$$\begin{tikzcd} U \arrow[r, "S"', "\beta"] \arrow[d, "S\lambda"'] & U' \arrow[d, dotted, "S\lambda"] \\ M \arrow[r, dotted] & M' \end{tikzcd}$$

Proof.

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

We can summarize the previous theorem, stating the soundness of $S\lambda$ for β-reducibility, and the invariance lemma 4.14, expressing the soundness of $S\lambda$ for NIL-reducibility in a single statement mentioning the generic notion of reduction of $S^{\rightarrow\multimap\&\top}$.

Corollary 4.20 (*Soundness of $S\lambda$ for reducibility*)

If $U \xrightarrow{S} V$ and $U \xrightarrow{S\lambda} M$, then there is a term N such that $M \longrightarrow^{*} N$ and $V \xrightarrow{S\lambda} N$.

$$
\begin{tikzcd}
U \arrow[r, "S"] \arrow[d, "S\lambda"'] & V \arrow[d, dotted, "S\lambda"] \\
M \arrow[r, dotted, two heads] & N
\end{tikzcd}
$$

Proof.

Depending on whether $\xrightarrow{S}$ is $\xrightarrow{S}_{\text{NIL}}$ or $\xrightarrow{S}_{\beta}$, this statement corresponds to Lemma 4.14 or to theorem 4.19, respectively. In the former case, $N = M$ and $\longrightarrow^{*}$ is instantiated to the identity. ☑

Clearly, the above result holds also relatively to the reflexive and transitive closure of $\xrightarrow{S}$.

The previous theorem, together with the fact that $S\lambda$ and λS form a pair of inverse functions, allows us to achieve a simple proof of the completeness of λS with respect to the reduction semantics of $S^{\rightarrow\multimap\&\top}$. Notice that this corollary mentions both β- and NIL-reductions.

Corollary 4.21 (*Completeness of λS for reduction*)

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

If $M \xrightarrow{\lambda S} U$ and $U \xrightarrow{S}_{\beta} \overline{V} \xrightarrow{S}{}^{*}_{\text{NIL}} V$ with V in NIL-normal form, then there is a term N such that $M \longrightarrow N$ and $N \xrightarrow{\lambda S} V$.

$$
\begin{tikzcd}
M \arrow[r, dotted] \arrow[d, "\lambda S"'] & N \arrow[drr, dotted, "\lambda S"] & & \\
U \arrow[r, "S"', "\beta"] & \overline{V} \arrow[rr, two heads, "S", "\text{NIL}"'] & & V
\end{tikzcd}
$$

Proof.

By the untyped invertibility lemma 4.10, there is a derivation of $U \xrightarrow{S\lambda} M$. By the soundness of $S\lambda$ with respect to β-reduction, there is a term N such that $M \longrightarrow N$ and $\overline{V} \xrightarrow{S\lambda} N$. By the invariance of $S\lambda$ under NIL-reduction, there is a derivation of $V \xrightarrow{S\lambda} N$. By composing various typing soundness results, we obtain that $\Gamma; \Delta \vdash_{\Sigma} V : A$, so that we can apply the invertibility lemma, obtaining that $N \xrightarrow{\lambda S} V$ is derivable. ☑

5 Properties of $S^{\rightarrow\multimap\&\top}$

We will now present the main properties of $S^{\rightarrow\multimap\&\top}$, ultimately strong normalization and the uniqueness of normal forms. In order to do so, we will take advantage of the facts that similar results hold for $\lambda^{\rightarrow\multimap\&\top}$, and that we have reasonably well-behaved translations to and from this calculus. An alternative would have been to give direct proofs of these properties.

We begin by showing that $S^{\rightarrow\multimap\&\top}$ admits confluence and the Church-Rosser property. Differently from $\lambda^{\rightarrow\multimap\&\top}$, the statement of this properties must include typing assumptions in order to express certain extensionality requirements. For typographic reasons, we model the equivalence relation $\overset{S}{\equiv}$ with a double arrow.

Theorem 5.1 (*Church-Rosser*)

Confluence: *Assume that $\mathcal{U} :: \Gamma; \Delta \vdash_{\Sigma} U : A$.*
If $\mathcal{R}' :: U \xrightarrow{S}{}^{*} U'$ and $\mathcal{R}'' :: U \xrightarrow{S}{}^{*} U''$, then there is a term V such that $\mathcal{R}^{*} :: U' \xrightarrow{S}{}^{*} V$ and $\mathcal{R}^{**} :: U'' \xrightarrow{S}{}^{*} V$.
Similarly for spines

$$
\begin{tikzcd}
 & U \arrow[dl, "S"'] \arrow[dr, "S"] & \\
U' \arrow[dr, dotted, "S"'] & & U'' \arrow[dl, dotted, "S"] \\
 & V &
\end{tikzcd}
$$

Church-Rosser: *Assume that $\mathcal{U}' :: \Gamma; \Delta \vdash_{\Sigma} U' : A$ and $\mathcal{U}'' :: \Gamma; \Delta \vdash_{\Sigma} U'' : A$.*
If $\mathcal{R} :: U' \overset{S}{\equiv} U''$, then there is a term V such that $\mathcal{R}^{*} :: U' \xrightarrow{S}{}^{*} V$ and $\mathcal{R}^{**} :: U'' \xrightarrow{S}{}^{*} V$.
Similarly for spines

$$
\begin{tikzcd}
U' \arrow[rr, leftrightarrow, "S"] \arrow[dr, dotted, "S"'] & & U'' \arrow[dl, dotted, "S"] \\
 & V &
\end{tikzcd}
$$

Proof.

We will carry out the proof in the case of confluence only. The Church-Rosser property is handled similarly.

Since, by Lemma 4.8, $S\lambda$ is a total function over $S^{\rightarrow -\circ \& \top}$, there is a unique term M such that $U \xrightarrow{S\lambda} M$ is derivable. By typing soundness, we obtain that $\Gamma; \Delta \vdash_\Sigma M \Uparrow A$. By iterated applications of the soundness of $S\lambda$ over reduction, we deduce that there are terms M' and M'' such that $M \longrightarrow^* M'$ and $U' \xrightarrow{S\lambda} M'$, and similarly $M \longrightarrow^* M''$ and $U'' \xrightarrow{S\lambda} M''$. By subject reduction, we have that $\Gamma; \Delta \vdash_\Sigma M' \Uparrow A$ and $\Gamma; \Delta \vdash_\Sigma M'' \Uparrow A$. By the confluence property of $\lambda^{\rightarrow -\circ \& \top}$, we know that there exists a term N such that $M' \longrightarrow^* N$ and $M'' \longrightarrow^* N$ are derivable.

By the invertibility lemma, there are $S^{\rightarrow -\circ \& \top}$ terms U^* and U^{**} such that $M' \xrightarrow{\lambda S} U^*$ with $U' \xrightarrow{S}{}^*_{\text{NIL}} U^*$ and $M'' \xrightarrow{\lambda S} U^{**}$ with $U'' \xrightarrow{S}{}^*_{\text{NIL}} U^{**}$. By the soundness of λS with respect to reductions, there are terms V' and V'' such that $U^* \xrightarrow{S}{}^* V'$ and $N \xrightarrow{\lambda S} V'$, and similarly $U^{**} \xrightarrow{S}{}^* V''$ and $N \xrightarrow{\lambda S} V''$. However, since λS is a function, $V' = V''$; let us call this term V. By composing the various reductions above, we obtain the desired derivations of $U' \xrightarrow{S}{}^* V$ and $U'' \xrightarrow{S}{}^* V$. ☑

Next, we consider the $S^{\rightarrow -\circ \& \top}$ equivalent of the transitivity lemma 2.3 discussed in Section 2. As in $\lambda^{\rightarrow -\circ \& \top}$, we must distinguish the linear and the intuitionistic cases, but we have no convenient notation that spans uniformly over terms and spines. Therefore, the lemma below has four parts.

Lemma 5.2 (*Transitivity*)

i. If $\mathcal{U} :: \Gamma; \Delta, x{:}B \vdash_\Sigma U : A$ *and* $\mathcal{U}_V :: \Gamma; \Delta' \vdash_\Sigma V : B$, *then* $\mathcal{U}' :: \Gamma; \Delta, \Delta' \vdash_\Sigma [V/x]U : A$.

ii. If $\mathcal{S} :: \Gamma; \Delta, x{:}B \vdash_\Sigma S : A > a$ *and* $\mathcal{U}_V :: \Gamma; \Delta' \vdash_\Sigma V : B$, *then* $\mathcal{S}' :: \Gamma; \Delta, \Delta' \vdash_\Sigma [V/x]S : A > a$.

iii. If $\mathcal{U} :: \Gamma, x{:}B; \Delta \vdash_\Sigma U : A$ *and* $\mathcal{U}_V :: \Gamma; \cdot \vdash_\Sigma V : B$, *then* $\mathcal{U}' :: \Gamma; \Delta \vdash_\Sigma [V/x]U : A$.

iv. If $\mathcal{S} :: \Gamma, x{:}B; \Delta \vdash_\Sigma S : A > a$ *and* $\mathcal{U}_V :: \Gamma; \cdot \vdash_\Sigma V : B$, *then* $\mathcal{U}' :: \Gamma; \Delta \vdash_\Sigma [V/x]S : A > a$.

Proof.

We prove this lemma by means of a technique similar to the one we just sketched in the case of the Church-Rosser property. We illustrate the manner spines are handled by presenting the full treatment of case (ii). The treatment of the other parts is similar or simpler.

Let z be a variable that does not appear in neither Γ, Δ, nor Δ', and that is different from x. We will use it as a generic head for S. By rule **lλ_lvar**, there is a (trivial) typing derivation of $\Gamma; z{:}A \vdash_\Sigma z \downarrow A$. On the basis of this fact, by the soundness of $S\lambda$ for typing (Theorem 4.7), there is a term M such that $S \setminus z \xrightarrow{S\lambda} M$ and $\Gamma; \Delta, x : B, z : A \vdash_\Sigma M \Uparrow a$ are derivable. By the same theorem, there is a term N and derivations of $V \xrightarrow{S\lambda} N$ and $\Gamma; \Delta' \vdash_\Sigma N \Uparrow B$. By the transitivity lemma 2.3 for $\lambda^{\rightarrow -\circ \& \top}$, $\Gamma; \Delta, \Delta', z{:}A \vdash_\Sigma [N/x]M \Uparrow a$ is derivable.

By rule **λS_var**, there is a derivation of $z \setminus S \xrightarrow{\lambda S} z \cdot S$. By the invertibility lemma 4.9, there is a term U' and derivations of $M \xrightarrow{\lambda S} U'$ and $z \cdot S \xrightarrow{S}{}^*_{\text{NIL}} U'$. By inversion on the reduction rules for $S^{\rightarrow -\circ \& \top}$, $U' = z \cdot S'$ for some spine S'. Therefore, by rule **Sr_var**, there must be a derivation of $S \xrightarrow{S}{}^*_{\text{NIL}} S'$. Again by the invertibility lemma 4.9, there is a term V' and a derivation $N \xrightarrow{\lambda S} V'$, where $V \xrightarrow{S}{}^*_{\text{NIL}} V'$ is derivable. By the substitution lemma 4.4, there is a term $\overline{V}$ and derivations of $[N/x]M \xrightarrow{\lambda S} \overline{V}$ and $z \cdot ([V'/x]S') \xrightarrow{S}{}^*_{\text{NIL}} \overline{V}$ (remember that $x \neq z$). Again by invertibility on the reduction rules for $S^{\rightarrow -\circ \& \top}$, $\overline{V} = z \cdot \overline{S}$ for some spine $\overline{S}$, and $[V'/x]S' \xrightarrow{S}{}^*_{\text{NIL}} \overline{S}$. By iterated applications of the substitution lemma 3.8, there is a derivation of $[V/x]S \xrightarrow{S}{}^*_{\text{NIL}} \overline{S}$.

By the soundness of λS with respect to typing, $\Gamma; \Delta, \Delta', z{:}A \vdash_\Sigma z \cdot \overline{S} : a$ is derivable. By inversion on rule **lS_lvar**, $\Gamma; \Delta, \Delta' \vdash_\Sigma \overline{S} : A > a$ is derivable as well. Now, since NIL-expansion preserves typing, there is a derivation of $\Gamma; \Delta, \Delta' \vdash_\Sigma [V/x]S : A > a$. ☑

The next property we are interested in proving for $S^{\rightarrow -\circ \& \top}$ is subject reduction. Again, we must deal separately with terms and with spines. Remember that we have already proved this property in the subcase of NIL-reduction.

Lemma 5.3 (*Subject reduction*)

i. If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$ *and* $\mathcal{Q} :: U \xrightarrow{S} V$, *then* $\mathcal{U}' :: \Gamma; \Delta \vdash_\Sigma V : A$.

ii. If $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$ *and* $\mathcal{Q} :: S \xrightarrow{S} S'$, *then* $\mathcal{S}' :: \Gamma; \Delta \vdash_\Sigma S' : A > a$.

Proof.

We will prove only part (i) of this statement. Part (ii) adapts the technique we just applied in the transitivity lemma.

By the soundness of $S\lambda$ with respect to typing, there are a term M and derivations of $U \xrightarrow{s\lambda} M$ and $\Gamma; \Delta \vdash_\Sigma M \Uparrow A$. By the soundness of $S\lambda$ with respect to reductions, there are a term N and derivations of $V \xrightarrow{s\lambda} N$ and $M \longrightarrow^* N$. By the subject reduction property of $\lambda^{\rightarrow\multimap\&\top}$, $\Gamma; \Delta \vdash_\Sigma N \Uparrow A$ is derivable.

Now, by the soundness of λS with respect to typing, there is a term V' such that $\Gamma; \Delta \vdash_\Sigma V' : A$ and $N \xrightarrow{\lambda s} V'$ are derivable. On the other hand, by the invertibility lemma 4.9, there is a term V'' such that $N \xrightarrow{\lambda s} V''$ and $V \xrightarrow{S}^*_{\text{NIL}} V''$ are derivable. However, since, by Lemma 4.2, λS is a function, we have that $V' = V''$. Then, in order to conclude this proof, we simply take advantage of the fact that NIL-expansion preserves typing (Lemma 3.3) to obtain the desired derivation of $\Gamma; \Delta \vdash_\Sigma V : A$. ☑

We now tackle strong normalization which, as in the case of $\lambda^{\rightarrow\multimap\&\top}$, states that no infinite chain of (either NIL- or β-) reductions can start from a well-typed $S^{\rightarrow\multimap\&\top}$ term. Therefore, we can reduce a well-typed term to normal (actually canonical) form by exhaustively reducing randomly selected redices.

Theorem 5.4 (*Strong normalization*)

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

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

Proof.

We will prove only part (i) of this theorem. Part (ii) is handled similarly.

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 the following reductions:

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

By the soundness of $S\lambda$ with respect to reducibility, every β-reduction in σ corresponds to a reduction in $\lambda^{\rightarrow\multimap\&\top}$ (Theorems 4.19) while every NIL-reduction disappears (Lemma 4.14). This entails that there is a sequence of $\lambda^{\rightarrow\multimap\&\top}$ term $M_0, M_1, M_2, \ldots$ such that on the one hand there are derivations of $U_i \xrightarrow{s\lambda} M_{\varphi(i)}$ where φ maps maximal subsequences of σ linked by NIL-reductions to the same $\lambda^{\rightarrow\multimap\&\top}$ term, and on the other hand the following reduction sequence is derivable

$$\sigma' = M_0 \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \ldots$$

Notice in particular that there is a derivation of $U \xrightarrow{s\lambda} M_0$. Therefore, by the soundness of $S\lambda$ with respect to typing, the judgment $\Gamma; \Delta \vdash_\Sigma M_0 \Uparrow A$ is derivable. By the strong normalization theorem for $\lambda^{\rightarrow\multimap\&\top}$, σ' is finite. Then, also σ must be finite since, by the strong normalization of NIL-reduction (Lemma 3.7), the maximal subsequences of NIL-reducts collapsed by φ are finite. ☑

Strong normalization ensures that exhaustive reductions of a well-typed $S^{\rightarrow\multimap\&\top}$ term (or spine) will eventually produce an object in normal form. Depending on which redex is selected at each step, this procedure might yield different normal objects. The uniqueness corollary below guarantees that every reduction path will lead to the same normal term (or spine), up to the renaming of bound variables.

Corollary 5.5 (*Uniqueness of normal forms*)

i. If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$, *then there is a unique normal term* V *such that* $\mathcal{Q}' :: U \longrightarrow^* V$.

ii. If $\mathcal{S} :: \Gamma; \Delta \vdash_\Sigma S : A > a$*, then there is a unique normal spine* S' *such that* $\mathcal{Q}' :: S \longrightarrow^* S'$.

Proof.

By the strong normalization theorem, we know that every sequence of reductions starting at U leads to a term in normal form. Let consider two reduction sequences validating $U \longrightarrow^* V'$ and $U \longrightarrow^* V''$, for terms V' and V'' in normal form. By confluence, there is a term V to which both reduce. However, since V' and V'' do not contain redices, the only way to close the diamond is to have that $V' = V'' = V$, and use the identical reduction.

We proceed similarly in order to prove the second part of this statement. ☑

As in the case of $\lambda^{\rightarrow\multimap\&\top}$, the above results entitle speaking about *the* normal form (or equivalently *the* canonical form) of a term U or a spine S, whenever these objects are well-typed. We denote this term and spine $\mathrm{Can}(U)$ and $\mathrm{Can}(S)$, respectively. A calculus that accepts only canonical objects can be obtained from the typing system displayed in Figure 3 by simply removing rule **lS_redex**.

A term (spine) in which redices appear at most in the argument of an application is said to be in *weak head-normal form*. Any well-typed term can be converted to weak-head normal form by repeatedly selecting a redex that violates this property and reducing it. A similar property holds for spines. We use $\overline{U}$ and $\overline{S}$ to denote the weak-head normal form of a term U and a spine S, respectively. Weak head-normalization is not as computationally expensive as full normalization since it operates on shallow redices only. However, it exposes enough of a normal form to work comfortably in many circumstances. Therefore, the implementation of procedures that, by their very nature, need to perform reductions, unification for example, often rely on weak head-normalization rather than on full normalization.

6 Further Remarks

In this section, we briefly report on important relationships between our spine calculus and other formal systems in the literature. More precisely, we hint at an alternative development of the results obtained in this paper (Section 6.1), point at a relationship between the spine calculus and the logic programming notion of uniform derivability (Section 6.2), and discuss related work (Section 6.3).

6.1 Alternative Development

We observed that the spine calculus $S^{\rightarrow\multimap\&\top}$ has more structure than the corresponding traditional formulation, as manifested by the presence of the NIL-reduction rule, which has no equivalent in $\lambda^{\rightarrow\multimap\&\top}$. When analyzing the translation function λS, we had to recover this additional structure by accompanying most statements with $\lambda^{\rightarrow\multimap\&\top}$ typing derivations. This step was over-restrictive, but acceptable because of our interest in well-typed terms only.

We can achieve a better correspondence by enriching the syntax of $\lambda^{\rightarrow\multimap\&\top}$ with the two new operators AC and CA. Let us call the enriched language $\lambda_{ac}^{\rightarrow\multimap\&\top}$. We use these new constructs to annotate the *coercion* rules **lλ_atm** and **lλ_redex**, which become:

$$\frac{\Gamma; \Delta \vdash_\Sigma M \downarrow a}{\Gamma; \Delta \vdash_\Sigma \textsc{ac}\, M \Uparrow a}\ \mathbf{l\lambda\_atm} \qquad\qquad \frac{\Gamma; \Delta \vdash_\Sigma M \Uparrow A}{\Gamma; \Delta \vdash_\Sigma \textsc{ca}\, M \downarrow A}\ \mathbf{l\lambda\_redex}$$

Notice that every typing derivation $\Gamma; \Delta \vdash_\Sigma M \Uparrow\!\!\Downarrow A$ can now be uniquely reconstructed on the basis of the term M, except for the composition of the context $\Gamma; \Delta$ (in particular, the manner Δ is split in certain applications of multiplicative rules).

We augment the reduction semantics of $\lambda^{\rightarrow\multimap\&\top}$ with the rule

$$\beta_{ac}: \quad \textsc{ca}\,(\textsc{ac}\, M) \longrightarrow M$$

which corresponds to eliminating alternations of rules **lλ_atm** and **lλ_redex**. We call the expression on the left-hand side of the arrow a *coercion redex*. The other reduction rules are upgraded as follows:

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

Then, the absence of any occurrence of $\textsc{ca}$ in a well-typed term implies that a term does not contain redices, and therefore that it is in normal form.

We modify the translation rules λ**S_atm** and λ**S_redex** as follows:

$$\frac{M \setminus \textsc{nil} \xrightarrow{\lambda S} H \cdot S}{\textsc{ac}\,M \xrightarrow{\lambda S} H \cdot S}\ \lambda\mathbf{S\_atm} \qquad\qquad \frac{M \xrightarrow{\lambda S} U}{\textsc{ca}\,M \setminus S \xrightarrow{\lambda S} U \cdot S}\ \lambda\mathbf{S\_redex}$$

Notice the absence of side-conditions. On the basis of these upgraded definitions, it is easy to see that λS maps coercion redices to $\textsc{nil}$ redices, and therefore applications of the new reduction rule β_{ac} are emulated in $S^{\rightarrow -\circ \& \top}$ by $\textsc{nil}$-reductions.

Rules **Sλ_con**, **Sλ_var** and **Sλ_redex** of $S\lambda$ are modified as follows:

$$\frac{S \setminus c \xrightarrow{S\lambda} M}{c \cdot S \xrightarrow{S\lambda} \textsc{ac}\,M}\ \mathbf{S\lambda\_con} \qquad \frac{S \setminus x \xrightarrow{S\lambda} M}{x \cdot S \xrightarrow{S\lambda} \textsc{ac}\,M}\ \mathbf{S\lambda\_var} \qquad \frac{U \xrightarrow{S\lambda} M' \quad S \setminus \textsc{ca}\,M' \xrightarrow{S\lambda} M}{U \cdot S \xrightarrow{S\lambda} \textsc{ac}\,M}\ \mathbf{S\lambda\_redex}$$

The new and improved $S\lambda$ translates $\textsc{nil}$-redices to coercion redices and puts $\textsc{nil}$-reductions and β_{ac} in one-to-one correspondence. As we suggested, this treatment eliminates the mismatch between $\lambda^{\rightarrow -\circ \& \top}$ and $S^{\rightarrow -\circ \& \top}$ that led to many of the complications in Section 4.

Using $\lambda_{ac}^{\rightarrow -\circ \& \top}$ as an external language is impractical for most applications. Indeed, a more reasonable approach is to give the user the simpler $\lambda^{\rightarrow -\circ \& \top}$ and have coercions filled in during parsing; this is essentially what λS did in the previous sections. Coercions need to be inserted at every alternation of constructors and destructors in a term. Notice that the resulting object does not contain coercion redices but, similarly to $\textsc{nil}$-redices, they may be exposed by the application of β-reductions. Further flexibility can be achieved by relieving the user from the requirement of writing terms in η-long form only: subterms can be expanded as soon as their type has been inferred as a result of type checking.

6.2 Relationship to Uniform Provability

An *abstract logic programming language* [MNPS91] is a fragment of a logic such that every derivable sequent has a *uniform derivation*. An intuitionistic cut-free sequent derivation is uniform if it is constructed in the following way, from the bottom up: right introduction rules are applied until the formula on the right-hand side of the sequent (the *goal formula*) is atomic, then a formula on the left-hand side (the *program*) of the sequent is selected (the *focus* or *stoup*) and left introduction rules are applied to it until the same atomic formula is exposed, possibly spawning subgoals that are to have uniform proofs.

The fragment of linear logic obtained by considering the types of $\lambda^{\rightarrow -\circ \& \top}$ and $S^{\rightarrow -\circ \& \top}$ as logic formulas is known as the language of (propositional) linear hereditary Harrop formulas [HM94, Cer96]. We denoted it $ILL^{\rightarrow -\circ \& \top}$ in Section 2. This formalism is an abstract logic programming language and a uniform proof system for it, adapted from [Cer96], is reported in Figure 9. The uniform provability judgment

$$\Gamma; \Delta \xrightarrow{u} A$$

is subject to the application of the right introduction rules of a sequent calculus presentation of $ILL^{\rightarrow -\circ \& \top}$. When an atomic formula a is exposed (rules **u_lin** and **u_int**), a program formula A is selected and isolated in the central part of the *immediate entailment judgment*

$$\Gamma; \Delta \xrightarrow{u} A \gg a$$

and left introduction rules are applied to it.

Uniform provability	
$$\frac{\Gamma;\Delta \xrightarrow{u} A \gg a}{\Gamma;\Delta,A \xrightarrow{u} a}\ \texttt{u\_lin}$$	$$\frac{\Gamma,A;\Delta \xrightarrow{u} A \gg a}{\Gamma,A;\Delta \xrightarrow{u} a}\ \texttt{u\_int}$$
$$\frac{}{\Gamma;\Delta \xrightarrow{u} \top}\ \texttt{u\_top}$$	$$\frac{\Gamma;\Delta \xrightarrow{u} A_1 \quad \Gamma;\Delta \xrightarrow{u} A_2}{\Gamma;\Delta \xrightarrow{u} A_1 \mathbin{\&} A_2}\ \texttt{u\_with}$$
$$\frac{\Gamma;\Delta,A \xrightarrow{u} B}{\Gamma;\Delta \xrightarrow{u} A \multimap B}\ \texttt{u\_lolli}$$	$$\frac{\Gamma,A;\Delta \xrightarrow{u} B}{\Gamma;\Delta \xrightarrow{u} A \to B}\ \texttt{u\_imp}$$
Immediate entailment	
$$\frac{}{\Gamma;\cdot \xrightarrow{u} a \gg a}\ \texttt{i\_atm}$$	
(No rule for $\top$) $$\frac{\Gamma;\Delta \xrightarrow{u} A_1 \gg a}{\Gamma;\Delta \xrightarrow{u} A_1 \mathbin{\&} A_2 \gg a}\ \texttt{i\_with1}$$	$$\frac{\Gamma;\Delta \xrightarrow{u} A_2 \gg a}{\Gamma;\Delta \xrightarrow{u} A_1 \mathbin{\&} A_2 \gg a}\ \texttt{i\_with2}$$
$$\frac{\Gamma;\Delta' \xrightarrow{u} B \gg a \quad \Gamma;\Delta'' \xrightarrow{u} A}{\Gamma;\Delta',\Delta'' \xrightarrow{u} A \multimap B \gg a}\ \texttt{i\_lolli}$$	$$\frac{\Gamma;\Delta \xrightarrow{u} B \gg a \quad \Gamma;\cdot \xrightarrow{u} A}{\Gamma;\Delta \xrightarrow{u} A \to B \gg a}\ \texttt{i\_imp}$$

Figure 9: Uniform Derivability

There is a striking correspondence between the proof system displayed in Figure 9 and the typing inference system for $S^{\to\multimap\&\top}$ given in Figure 1. Indeed, deleting every trace of terms from the typing rules of our spine calculus yields precisely the above derivability rules for $ILL^{\to\multimap\&\top}$, except for rules **lS_con** and **lS_redex** that do not have any match. A uniform provability equivalent of rule **lS_con** can be obtained by partitioning the left-hand side of a sequent into an intuitionistic *program*, corresponding to the concept of signature, and a collection of *dynamic assumptions*, corresponding to the notion of context in $S^{\to\multimap\&\top}$. If we ignore the terms in rule **lS_redex**, we recognize an analogue of the cut rule. Clearly, since uniform derivations are cut-free, the system in Figure 9 is not supposed to contain such an inference figure.

The similarity between the inference rules of uniform provability and the typing rules of $S^{\to\multimap\&\top}$ indicates that our spine calculus is a natural term assignment system for uniform derivations. This sets the basis for a form of the Curry-Howard isomorphism [How69] between normal, well-typed $S^{\to\multimap\&\top}$ terms and valid uniform derivations in $ILL^{\to\multimap\&\top}$.

6.3 Related Work

The uniform derivation system given in Figure 9 is a presentation of the sequent calculus for $ILL^{\to\multimap\&\top}$ that embeds restrictions on the applicability of inference rules. The strong relationship between intuitionistic fragment of sequent calculi (not necessarily linear) and term languages akin to our spine calculus has been already noticed in the literature. A first indirect reference appears in the seminal work of Howard on the types-as-formulas correspondence [How69], although a formal spine-like calculus is not defined.

In [Her95], Herbelin presents a systematic account of the relationship between the system LJT and the term language $\bar{\lambda}$, which extends the $\lambda^{\to}$ restriction of our spine calculus with a spine concatenation operator and explicit substitutions. LJT is a slightly massaged variant of the implicational fragment of Gentzen's intuitionistic sequent calculus with ideas similar to the uniform provability system from the previous section: in particular the left-hand side of a sequent contains a stoup and left rules are restricted to operate only to the formula currently on focus. Since no extensionality requirement is made on $\bar{\lambda}$ terms, the calculus relies on concatenation to append fragmented spines. The presence of explicit substitutions provides a direct handling of the two cut-rules of this calculus. $\bar{\lambda}$ is defined for foundational reasons, as the target λ-calculus of a derivations-as-terms correspondence for LJT. Indeed, its reduction rules correspond to the steps in a cut-elimination procedure for LJT, so that strong normalization theorem for $\bar{\lambda}$ subsumes the cut-elimination property for this logic.

Schwichtenberg [Sch97] adopts a similar approach relatively to a richer logic consisting of implication, conjunction and universal quantification. He starts from a more traditional presentation of the sequent calculus. In particular the absence of a stoup forces him to consider permutative conversions. The term calculus he proposes differs from Herbelin's by the absence of explicit concatenation operators and substitutions. It is therefore more similar to our spine calculus.

Barendregt [Bar80] relies on an term language akin to our spine calculus to study the notion of normalization in the untyped λ-calculus. Terms in this language are called Böhm trees.

7 Conclusion and Future Work

In this paper, we have formalized an alternative presentation of the linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ which we believe can be used to improve the efficiency of critical procedures such as unification in the implementation of languages based on (linear) λ-calculi. The resulting language, the spine calculus $S^{\rightarrow\multimap\&\top}$, strengthens the natural adaptation of the notion of abstract Böhm tree [Bar80, Her95] to encompass extensional products ($\&$), a unit type ($\top$) and linearity ($\multimap$), with the further requirement that well-typed terms be in η-long form. $S^{\rightarrow\multimap\&\top}$ terms of base type are structured similarly to the objects found in first-order term languages. In particular, their head is immediately available, an important benefit for procedures such as unification that base a number of choices on the nature of the heads of the terms they operate upon. Having extensionality built-in permits avoiding the overhead, both in terms of bookkeeping and execution time, of performing η-conversions at run time.

The intended applications of this work lie in proof search, logic programming, and the implementation of logical frameworks based on linear type theories. In particular, the spine calculus $S^{\rightarrow\multimap\&\top}$ has been designed as a first approximation of an internal representation for the type theory $\lambda^{\Pi\multimap\&\top}$ underlying the linear logical framework *LLF* [Cer96, CP96]. An extension to the full language, which includes dependent types, does not appear to be problematic. The adoption of a spine calculus as an internal representation device appears to integrate well with the simultaneous use of explicit substitutions [ACCL91]. However, the details of the amalgamation of these two techniques in the presence of linearity still need to be worked out.

A variant of the spine calculus deprived of linear constructs, but featuring dependent types and explicit substitutions is currently tested in a new implementation of the linear framework *LF* [HHP93] as a higher-order constraint logic programming language. This system, called *Twelf*, is expected to supersede the *Elf* implementation of *LF* currently in use [Pfe91, Pfe94]. It will be available later this year.

Acknowledgments

We would like to thank Carsten Schürmann for the insight he provided as a co-implementor of *Twelf* and through numerous discussions. We are also deeply indebted to Helmut Schwichtenberg for the fruitful discussions and for pointing out literature relevant to the topics treated in this paper.

References

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, October 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, 1996.

[Cer96] Iliano Cervesato. *A Linear Logical Framework.* PhD thesis, Dipartimento di Informatica, Università di Torino, February 1996.

[Chu40] Alonzo Church. A formulation of a simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[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. Linear higher-order pre-unification. Technical report, Department of Computer Science, Carnegie Mellon University, 1997. Forthcoming.

[dB72] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. MIT Press, 1990.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Her95] 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.

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

[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.

[How69] W. A. Howard. The formulae-as-types notion of construction. Unpublished manuscript, 1969. Reprinted in To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, 1980.

[IP96] Samin Ishtiaq and David Pym. A relevant analysis of natural deduction, December 1996. Manuscript.

[JG95] C. Barry Jay and Neil Ghani. The virtues of eta-expansion. *Journal of Functional Programming*, 2(5):135–154, 1995.

[Mil94] Dale Miller. A multiple-conclusion meta-logic. In S. Abramsky, editor, *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 272–281, Paris, France, July 1994.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[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.

[Pfe91] 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.

[Pfe94] Frank Pfenning. Elf: A meta-language for deductive systems. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 811–815, Nancy, France, June 1994. Springer-Verlag LNAI 814. System abstract.

[Sch97] Helmut Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. Manuscript, March 1997.