A Linear Spine Calculus*

Iliano Cervesato
Advanced Engineering and Sciences Division
ITT Industries, Inc.
Alexandria, VA 22303
iliano@itd.nrl.navy.mil

Frank Pfenning
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
fp@cs.cmu.edu

December 16, 2003

Abstract

We present the spine calculus $S^{\rightarrow -\circ \& \top}$ as an efficient representation for the linear λ-calculus $\lambda^{\rightarrow -\circ \& \top}$ which includes unrestricted functions ($\rightarrow$), linear functions ($-\circ$), additive pairing ($\&$), and additive unit ($\top$). $S^{\rightarrow -\circ \& \top}$ enhances the representation of Church's simply typed λ-calculus 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. Applications lie in proof search, logic programming, and logical frameworks based on linear type theories. It is also related to foundational work on term assignment calculi for presentations of the sequent calculus. We define the spine calculus, give translations of $\lambda^{\rightarrow -\circ \& \top}$ into $S^{\rightarrow -\circ \& \top}$ and vice-versa, prove their soundness and completeness with respect to typing and reductions, and show that the typable fragment of the spine calculus is strongly normalizing and admits unique canonical, *i.e.* $\beta\eta$-normal, forms.

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

*This work was sponsored by NSF Grants CCR-9303383 and CCR-9988281, and partially supported by NRL under contract N00173-00-C-2086.

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 such as theorem provers and logic programming languages. 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. Early implementations of systems embedding a higher-order term language, such as the logic programming languages *Elf* [Pfe94] and *λProlog* [Mil89, Mil01], typically represented terms in a way that mimics the traditional definition of a λ-calculus. Ignoring common orthogonal optimizations such as the use of de Bruijn 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 significantly improve the efficiency of higher-order unification algorithms. This approach has been studied extensively for different purposes [Bar80, Her95, DP98, DP99, Sch99, San00, DU01]. 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, such a representation may 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 this idea 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 a Church-Rosser theorem unless all terms are η-expanded [JG95]: this means that typing information must be stored and maintained in otherwise type-free procedures such as pattern unification [Mil91, Pfe91].

The benefits of the spine calculus representation, in conjunction with explicit substitutions, have been exploited in a new implementation of the logical framework *LF* [HHP93] as the higher-order logic programming language *Twelf* [PS99]. *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 [How80]. 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 [Sch99], Dyckhoff and colleagues [DP98, DP99, DU01], and Espírito Santo [San00] study a version of the intuitionistic spine representation and ordinary λ-calculi in a single system which incorpo-

rates commutative conversions, instead of the wholesale translation investigated here (which is closer to an efficient implementation).

$\lambda^{\rightarrow -\circ \& \top}$ corresponds, via a natural extension of the Curry-Howard isomorphism, to the $(\rightarrow -\circ \, \& \top)$ 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, CP02]. $\lambda^{\rightarrow -\circ \& \top}$ 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. $\lambda^{\rightarrow -\circ \& \top}$ shares similarities with the calculus proposed in [Bar96] and with the term language of the system *RLF* [IP98].

The implementation of a language based on linear type theories such as *LLF* and *RLF* raises new challenges that emerge neither for non-linear languages such as *Twelf* [PS99], nor for linear logic programming languages featuring plain (non-linear) 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 [CP97a]. The spine calculus $S^{\rightarrow -\circ \& \top}$ 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 [IP98] or sequent derivations [CP02] in substructural logics. In the realm of programming languages, linear terms naturally model *computations* in imperative languages [CP02] 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 [CP02]).

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 $\lambda^{\rightarrow -\circ \& \top}$ and present its main properties. We introduce the syntax and the typing and reduction semantics of $S^{\rightarrow -\circ \& \top}$ 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 $S^{\rightarrow -\circ \& \top}$. 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]. Appendix A studies the intermediate *coercion calculus* used in Section 4.

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

In this section, we introduce the linear simply-typed λ-calculus $\lambda^{\rightarrow -\circ \& \top}$, 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 and some properties in Section 2.3. It is the simply-typed variant of the linear type theory $\lambda^{\Pi -\circ \& \top}$, thoroughly analyzed in [Cer96]. We refer the interested reader to this work for the proofs of the properties of $\lambda^{\rightarrow -\circ \& \top}$ stated in this section.

2.1 Syntax

The simply-typed linear λ-calculus $\lambda^{\rightarrow -\circ \& \top}$ extends Church's $\lambda^{\rightarrow}$ with the three type constructors $-\circ$ (*linear function*), $\&$ (*additive product*) and $\top$ (*additive unit*), derived from the identically denoted connectives of linear logic [Gir87]. 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 unrestricted 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

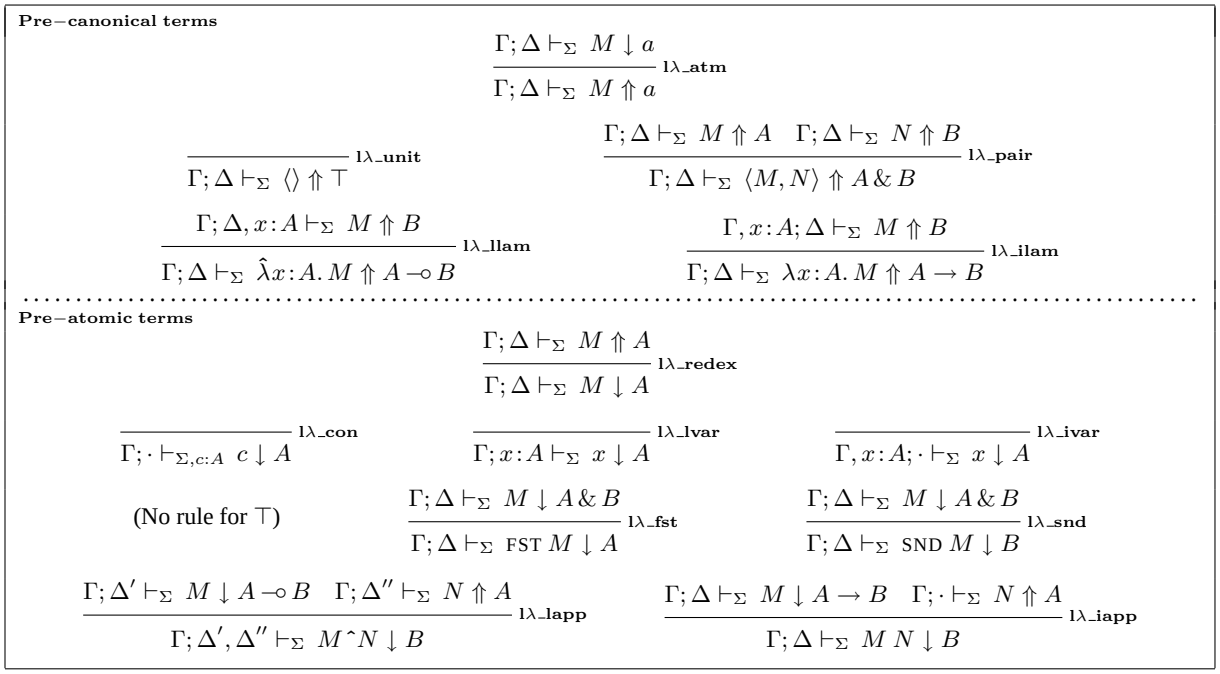


Figure 2.1: Typing for η-long $\lambda^{\to\multimap\&\top}$ Terms

can have any type.

Types:	$A ::= a$	*Terms:*	$M ::= c \mid x$		
	$\mid A_1 \to A_2$		$\mid \lambda x{:}A.\, M$	$\mid M_1\ M_2$	(*unrestricted functions*)
	$\mid A_1 \multimap A_2$		$\mid \hat{\lambda} x{:}A.\, M$	$\mid M_1 \hat{}\, M_2$	(*linear functions*)
	$\mid A_1 \,\&\, A_2$		$\mid \langle M_1, M_2\rangle$	$\mid \mathrm{FST}\ M \mid \mathrm{SND}\ M$	(*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 by the name of their bound variables and write $[M/x]N$ for the capture-avoiding substitution of M for x in the term N.

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. We require variables and constants to be declared at most once in a context and in a signature, respectively. Contexts and signatures are treated as sets; we promote "," to denote their disjoint union and omit writing "$\cdot$" when unnecessary.

Recall that, in $\lambda^{\to}$, a term M is in *η-long form* only if every function appearing in M (either as a symbol or as a redex) occurs applied to as many arguments as dictated by its type. For example, if $f : a \to a \to a$ and $c : a$, the term $\lambda y{:}a.\, f\ c\ y$ is η-long, but $f\ c$ is not. This idea extends naturally to $\lambda^{\to\multimap\&\top}$ by requiring that every symbol be enclosed by as many destructors as necessary to expose a base type. For example, given the constant $p : a \,\&\, a$, the term $\langle \mathrm{FST}\ p, \mathrm{SND}\ p\rangle$ is in η-long form, but p by itself is not. It also follows that $\langle\rangle$ is the only η-long term of type $\top$.

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 (see the Surjectivity Lemma 2.1 later in this section). Instead, untyped η-conversion rules are often included in the reduction semantics of a λ-calculus in order to expand or contract terms as 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*) such as $f\ c$ and p above. These judgments are respectively denoted as follows:

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

where Γ and Δ are called the *unrestricted* and the *linear* context, respectively. Whenever a property holds uniformly for the pre-canonical and pre-atomic judgments above, we will write $\Gamma;\Delta \vdash_\Sigma\ M \Uparrow\!\Downarrow A$ 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. We write $\mathcal{C}$ and $\mathcal{A}$, possibly super- and/or sub-scripted, for derivations of the above pre-canonical and pre-atomic typing judgments, respectively.

The rules displayed in the upper part of Figure 2.1 validate pre-canonical terms M by deriving judgments of the form $\Gamma;\Delta \vdash_\Sigma\ M \Uparrow A$. Rules $\mathbf{l\lambda\_unit}$, $\mathbf{l\lambda\_pair}$, $\mathbf{l\lambda\_llam}$ and $\mathbf{l\lambda\_ilam}$ 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 constructors of our language). The manner they handle their context is familiar from linear logic. Notice in particular that $\mathbf{l\lambda\_unit}$ is applicable with any linear context and that the premises of rule $\mathbf{l\lambda\_pair}$ share the same context, which also appears in its conclusion. Rules $\mathbf{l\lambda\_llam}$ and $\mathbf{l\lambda\_ilam}$ differ only by the nature of the assumption they add to the context in their premise: linear in the case of the former, unrestricted for the latter. The remaining rule defining the pre-canonical judgment, $\mathbf{l\lambda\_atm}$, is particularly interesting since it is the reason why all terms derivable in the pre-canonical system are in η-long form: a pre-atomic term must be fully applied (*i.e.*, have base type) before it can be considered pre-canonical. Observe that the rules defining the pre-canonical judgment are type-directed.

The rules defining the pre-atomic judgment, $\Gamma;\Delta \vdash_\Sigma\ M \downarrow A$, are displayed in the lower part of Figure 2.1. They validate constants (rule $\mathbf{l\lambda\_con}$) and linear and unrestricted variables (rules $\mathbf{l\lambda\_lvar}$ and $\mathbf{l\lambda\_ivar}$, respectively). They also allow the formation of the terms $\textsc{fst}\ M$, $\textsc{snd}\ M$, $M\,\hat{}\,N$ and $M\ N$ that start with one of the destructors of $\lambda^{\rightarrow\multimap\&\top}$ (rules $\mathbf{l\lambda\_fst}$, $\mathbf{l\lambda\_snd}$, $\mathbf{l\lambda\_lapp}$ and $\mathbf{l\lambda\_iapp}$, respectively). The role played by linear assumptions in $\lambda^{\rightarrow\multimap\&\top}$ is particularly evident in these rules. Indeed, an axiom rule ($\mathbf{l\lambda\_con}$, $\mathbf{l\lambda\_lvar}$ and $\mathbf{l\lambda\_ivar}$) 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 $\mathbf{l\lambda\_lapp}$ is split and distributed among its premises. Observe also that the linear context of the argument part of an unrestricted application, in rule $\mathbf{l\lambda\_iapp}$, is constrained to be empty. The presence of rule $\mathbf{l\lambda\_redex}$ accounts for the possibility of validating terms containing β-redices, as defined below: it allows arbitrary pre-canonical terms in positions where only pre-atomic objects could otherwise appear. 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, CP02]. 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 2.1 correspond to the specification of the familiar inference rules for the $(\rightarrow\ \multimap\&\top)$ fragment of intuitionistic linear logic, $\mathit{ILL}^{\rightarrow\multimap\&\top}$ [HM94], presented in a *natural deduction* style. It is easy to prove the equivalence to the usual sequent formulation. $\lambda^{\rightarrow\multimap\&\top}$ and $\mathit{ILL}^{\rightarrow\multimap\&\top}$ 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 $\mathbf{l\lambda\_unit}$ and $\mathbf{l\lambda\_lapp}$ can collapse proofs with the same structure but different linear contexts to the same $\lambda^{\rightarrow\multimap\&\top}$ term.

We say a constructor for a type is *surjective* if its image includes all canonical members of the type. For example, if every canonical object of type $A \rightarrow B$ has the form $\lambda x : A.\,M$ for some M, then λ is surjective. Since this notion refers to objects that are canonical (*i.e.* do not contain β-redices and every pre-canonical subterm appears in η-long form), surjectivity is intimately related to the notion *extensionality*: intuitively, two members of a type are extensionally equal if they cannot be distinguished by applying elimination rules to them. We will

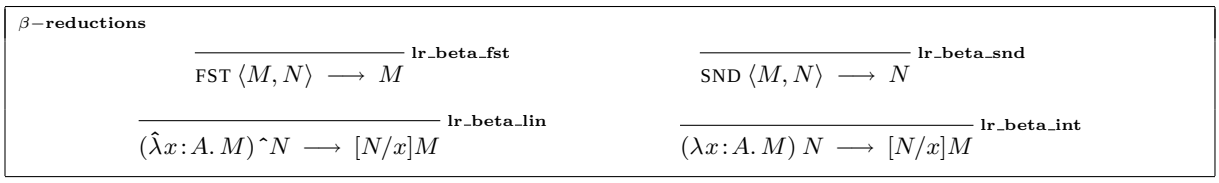


Figure 2.2: Reduction Semantics for $\lambda^{\rightarrow -\circ \& \top}$

not investigate these properties in detail, but it is worth noting that both surjectivity and extensionality are critical in applications of λ-calculi in logical frameworks and logic programming languages. In our language, we make an a priori commitment to extensionality, which is partially expressed by the surjectivity property below. Note that the constructors are surjective already for pre-canonical terms, not only canonical terms. It is this property, together with the subject reduction lemma, that allows an implementation to drop type information entirely during algorithms such as unification [CP97a]. Surjectivity is formalized in the following lemma, whose proof can be easily adapted from [Cer96].

Lemma 2.1 (*Surjectivity*)

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

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* $M = \lambda x{:}A.\, N$. □

This result, like the many to follow, adopts the convention that all meta-variables appearing in the antecedant of an if-then sentence are implicitely universally quantified, while the remaining variables (occurring only in the consequent) are existentially quantified.

2.3 Reduction Semantics

The reduction semantics of $\lambda^{\rightarrow -\circ \& \top}$ is given by the congruence relation on terms, $\longrightarrow$, based on the β-reduction rules in Figure 2.2. In addition we have straightforward congruence rules that allow a reduction to take place at an arbitrary subterm occurrence, but which we do not show explicitly. 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. We write $\mathcal{E}$, possibly variously decorated, for derivations of any of these judgments. It is easy to show that the rules obtained from Figure 2.2 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, we reserve the word *atomic* for a pre-atomic term that does not contain any β-redices.

Similarly to $\lambda^{\rightarrow}$, $\lambda^{\rightarrow -\circ \& \top}$ 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, $\lambda^{\rightarrow -\circ \& \top}$ 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 unrestricted 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 surjectivity: 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, which is well-defined by the above corollary. A calculus that validates only canonical terms can easily be obtained from the system in Figure 2.1 by removing rule $\mathbf{l\lambda\_redex}$.

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 contributes to achieving more efficient implementations of critical procedures such as unification [CP97a]. 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

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

where the head h is a constant or a variable and $(h\ M_1\ \ldots M_m)$ has base type. The usual parentheses saving conventions hide the fact that h is indeed deeply buried in the sequence of application and therefore not immediately accessible. A similar notational trick would be difficult for $\lambda^{\rightarrow\multimap\&\top}$, 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.*, $\mathrm{FST}\ ((\mathrm{SND}\ c)\,\hat{}\,x\ y)$).

The *spine calculus* $S^{\rightarrow -\circ \& \top}$ permits recovering both efficient head accesses and notational convenience. Every atomic term M of $\lambda^{\rightarrow -\circ \& \top}$ 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, $M = (h\, M_1\, \ldots M_m)$ is written $U = h \cdot (U_1; \ldots U_m; \text{NIL})$ in this language, where ";" represents application, NIL identifies the end of the spine, and U_i is the translation of M_i. Application and ";" have opposite associativity so that M_1 is the innermost subterm of M while U_1 is outermost in the spine of U. 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 has been employed in experimental implementation of a unification algorithm for *LLF* [Cer96, CP02] and *Twelf* [PS99]. A similar technique has been independently applied in the recent *Teyjus* implementation [Nad01] of *λProlog* [Mil89, Mil01].

The following grammar describes the syntax of $S^{\rightarrow -\circ \& \top}$: we write constructors as in $\lambda^{\rightarrow -\circ \& \top}$, but use new symbols to distinguish a spine operator from the corresponding term destructor.

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

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

We conclude this section by giving a few examples of how $\lambda^{\rightarrow -\circ \& \top}$ terms (left) appear, once rendered in the syntax of $S^{\rightarrow -\circ \& \top}$ (right), with a few parentheses added for clarity:

$$
\begin{array}{lcl}
c & \leftrightsquigarrow & c \cdot \text{NIL} \\
\langle c, d \rangle & \leftrightsquigarrow & \langle c \cdot \text{NIL}, d \cdot \text{NIL} \rangle \\
\lambda y{:}a.\, (f\; c\; y) & \leftrightsquigarrow & \lambda y{:}a.\, f \cdot ((c \cdot \text{NIL}); (y \cdot \text{NIL}); \text{NIL}) \\
\text{FST}\, ((\text{SND}\, c) \mathbin{\hat{}} x\; y) & \leftrightsquigarrow & c \cdot (\pi_2\, (x \cdot \text{NIL}) \mathbin{\hat{;}} (y \cdot \text{NIL}); \pi_1 \text{NIL})
\end{array}
$$

Admittedly, it takes some practice to familiarize oneself to the syntax of $S^{\rightarrow -\circ \& \top}$. However, we do not promote it as a replacement for $\lambda^{\rightarrow -\circ \& \top}$, but as an internal syntax aimed at expediting execution. We will describe translations from $\lambda^{\rightarrow -\circ \& \top}$ and $S^{\rightarrow -\circ \& \top}$, and vice versa, in Section 4.

3.2 Typing Semantics

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

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

The latter expresses the fact that given a head H of type A, the root $H \cdot S$ has type a. Notice that the target type of a well-typed spine is a base type. This has the desirable effect of permitting only η-long terms to be derivable in this calculus: 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 since more destructors could legitimately be applied to it. We will further comment on this point later.

The mutual definition of the two typing judgments of $S^{\rightarrow -\circ \& \top}$ is given in Figure 3.1. The rules concerning terms resemble very closely the definition of the pre-canonical judgment of $\lambda^{\rightarrow -\circ \& \top}$, except for the treatment of heads. The rules for the spine typing judgment are instead related to pre-atomic typing in $\lambda^{\rightarrow -\circ \& \top}$. 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 3.1. We write $\mathcal{U}$ and $\mathcal{S}$, possibly super-/sub-scripted, for derivations of the typing judgments for terms and spines respectively.

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

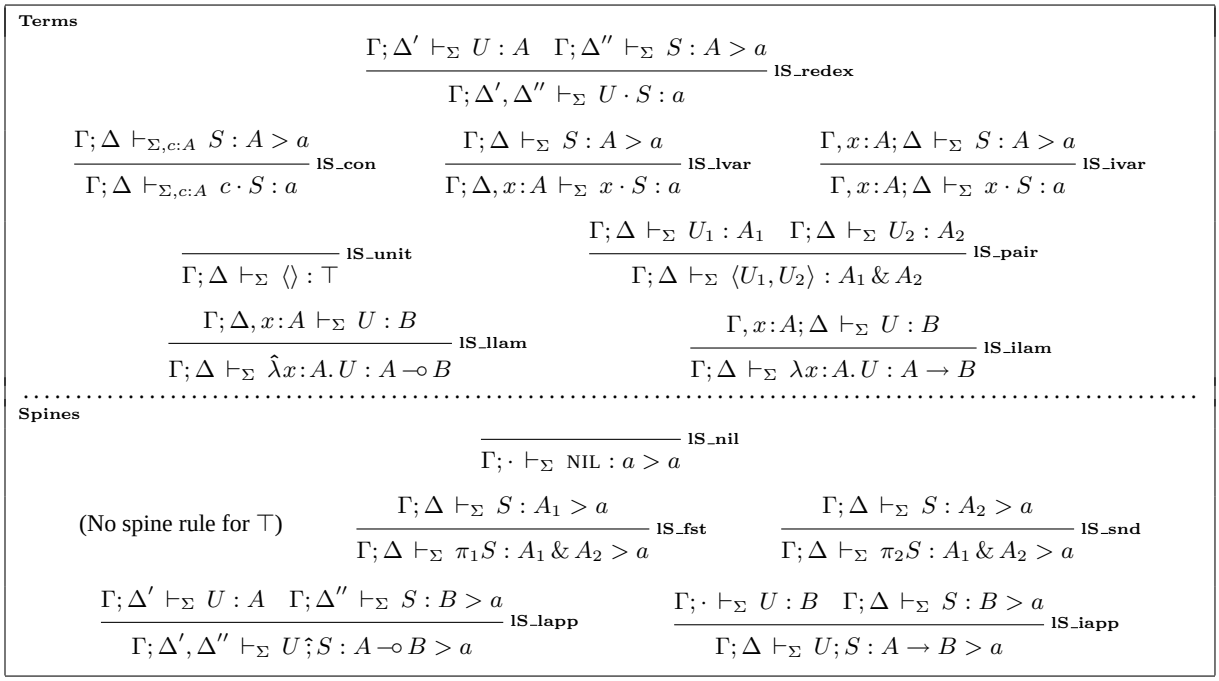


Figure 3.1: Typing for η-long $S^{\to\multimap\&\top}$ Terms

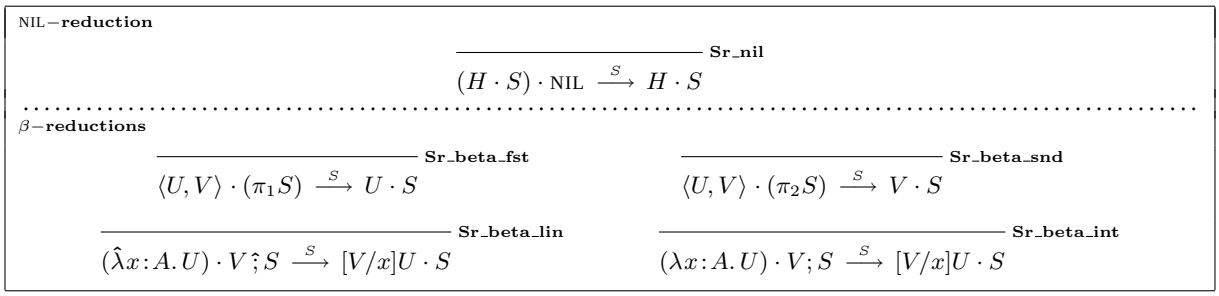


Figure 3.2: Reduction Semantics for $S^{\to\multimap\&\top}$

Lemma 3.1 (*Surjectivity*)

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 first rule applied in the given derivations. ☑

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

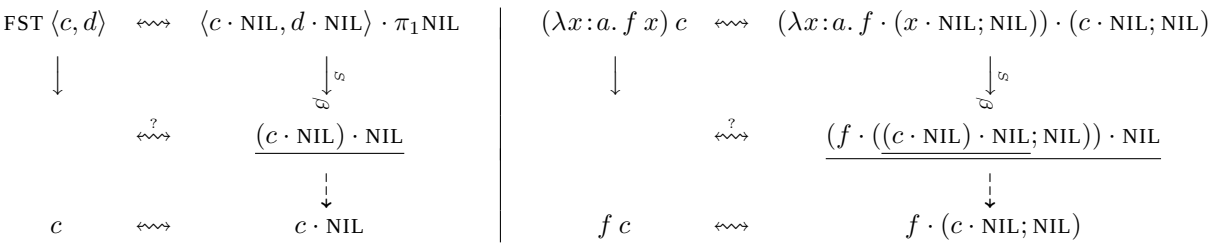
3.3 Reduction Semantics

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

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

The trailing spine in the reductions for $S^{\rightarrow\multimap\&\top}$ is a consequence of the fact that this language reverses the nesting order of $\lambda^{\rightarrow\multimap\&\top}$ destructors: S accounts for the operators that possibly *enclose* the corresponding $\lambda^{\rightarrow\multimap\&\top}$ object. 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 apply to both terms and spines. We denote the reflexive and transitive closure of this relation as $\xrightarrow{S}{}^{*}_{\beta}$. Finally, we write $\overset{S}{\equiv}_{\beta}$ for the associated equivalence relation. Formal inference rules for $\xrightarrow{S}_{\beta}$ are obtained by considering the lower segment of Figure 3.2 and the straightforward congruence rules that allow a reduction to be applied to an arbitrary subterm. We write $\mathcal{F}_{\beta}$, possibly superscripted, for derivations of these judgments.

It takes little experimentation to realize that the above β-reduction rules do not produce exactly the same effects as the notion of reducibility of $\lambda^{\rightarrow\multimap\&\top}$. Consider for example the simple projection redex $\mathrm{FST}\,\langle c, d\rangle$, which reduces to c in just one step in $\lambda^{\rightarrow\multimap\&\top}$. Applying rule $\mathbf{Sr\_beta\_fst}$ to the corresponding $S^{\rightarrow\multimap\&\top}$ term, $\langle c \cdot \mathrm{NIL}, d \cdot \mathrm{NIL}\rangle \cdot \pi_1 \mathrm{NIL}$, yields $(c \cdot \mathrm{NIL}) \cdot \mathrm{NIL}$ rather than the expected $c \cdot \mathrm{NIL}$. A similar phenomenon arises with functional redices, as schematized on the right-hand side of the following figure.



In both cases, the gap represented by the dashed arrow ($\dashrightarrow$) can be bridged if we consider terms of the form $(H \cdot S) \cdot \mathrm{NIL}$ as additional redices that reduce to $H \cdot S$. The projection redex on the left requires one such reduction (underlined), while the functional redex on the right would make two uses of it.

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

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

We call this rule NIL-*reduction*, its left-hand side a NIL-*redex* and write $\xrightarrow{S}_{\mathrm{NIL}}$ for the corresponding congruence relation. We denote its reflexive and transitive closure as $\xrightarrow{S}{}^{*}_{\mathrm{NIL}}$ and the corresponding equivalence relation as $\overset{S}{\equiv}_{\mathrm{NIL}}$. We write $\mathcal{F}_{\mathrm{NIL}}$, possibly decorated, for derivations of any of these judgments.

We will investigate the nature and properties of NIL-reductions in Sections 4 and 5. Meanwhile, we shall provide an informal explanation of its origin. The roots and spines of $S^{\rightarrow\multimap\&\top}$ act as a syntactic discriminant between pre-canonical and pre-atomic objects. Instead, $\lambda^{\rightarrow\multimap\&\top}$ requires a typing derivation to distinguish them: rules $\mathbf{l\lambda\_atm}$ and $\mathbf{l\lambda\_redex}$ play an essential role in this process, but are not syntactically accounted for within $\lambda^{\rightarrow\multimap\&\top}$ terms. We will see in Section 4 that observing $\lambda^{\rightarrow\multimap\&\top}$ reductions at the level of typing derivations reveals the formation of alternations of rules $\mathbf{l\lambda\_atm}$ and $\mathbf{l\lambda\_redex}$ which correspond precisely to occurrences of NIL-redices.

We write $\xrightarrow{S}$ for the union of $\xrightarrow{S}_{\beta}$ and $\xrightarrow{S}_{\mathrm{NIL}}$. 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 $\overset{S}{\equiv}$ for the corresponding equivalence relation. We denote derivations of these judgments as $\mathcal{F}$, possibly decorated. The definition of $\xrightarrow{S}$ is displayed

in Figure 3.2, except for the obvious congruence rules. As for $\lambda^{\rightarrow-\circ\&\top}$, 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 emphasizing that this object is in η-long form. By the above surjectivity property, every well-typed normal term is canonical.

It is interesting to observe that rule $\mathbf{Sr\_nil}$ is not sufficient in the presence of terms that are not in η-long form. Consider for example the redex $(\lambda x : a.\, f\ x)\ c\ d$ where f has type $a \rightarrow a \rightarrow a$. This term is not in η-long form (its η-expansion is $(\lambda x : a.\, \lambda y : a'.\, f\ x\ y)\ c\ d$), but still reduces to $f\ c\ d$ in $\lambda^{\rightarrow-\circ\&\top}$. The equivalent $S^{\rightarrow-\circ\&\top}$ expression reduces to $(f \cdot (c \cdot \text{NIL}); \text{NIL})) \cdot (d \cdot \text{NIL}; \text{NIL})$ after one β- and one NIL-reduction, but cannot be further reduced to the expected $f \cdot (c \cdot \text{NIL}; d \cdot \text{NIL}; \text{NIL})$. Such a step would require a reduction of the form:

$$(H \cdot S) \cdot S' \quad \xrightarrow{S} \quad H \cdot (S \mathbin{@} S')$$

where the meta-level operation $S@S'$ has the effect of *concatenating* spines S and S', *i.e.*, it replaces the trailing NIL of S with S'. Notice that $\mathbf{Sr\_nil}$ is an instance of this rule where $S' = \text{NIL}$. Observe also that the more general rule is not needed when operating exclusively with η-long terms: $(\lambda x{:}a.\, \lambda y{:}a'.\, f \cdot (x \cdot \text{NIL}; y \cdot \text{NIL}; \text{NIL})) \cdot (c \cdot \text{NIL}; d \cdot \text{NIL}; \text{NIL})$ reduces to $f \cdot (c \cdot \text{NIL}; d \cdot \text{NIL}; \text{NIL})$ by two applications of rule $\mathbf{Sr\_beta\_int}$ and three NIL-reductions. Since we are primarily interested in η-long terms in this paper, we will not pursue the meta-theory of this more general setting any further, except incidentally. We refer the interested reader to [CP97a] for a precise development of this observation.

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

There exists a structural translation of terms in $\lambda^{\rightarrow-\circ\&\top}$ to terms in $S^{\rightarrow-\circ\&\top}$ and vice versa. As we will see in this section, this translation preserves typing and β-reductions, so that $\lambda^{\rightarrow-\circ\&\top}$ and $S^{\rightarrow-\circ\&\top}$ share the same properties on well-typed (η-long) terms, and are therefore equivalent for practical purposes. Although a direct translation between the two languages is possible (see [CP97b]), a more elegant and instructive approach breaks this endeavor at the intermediate *coercion calculus* $C^{\rightarrow-\circ\&\top}$, which differs from $\lambda^{\rightarrow-\circ\&\top}$ by its syntactic account of the application of the rules that bridge the pre-canonical and pre-atomic terms. The translation between $\lambda^{\rightarrow-\circ\&\top}$ and $C^{\rightarrow-\circ\&\top}$ captures in full the dynamics of what appears as NIL-reductions in the spine calculus. $C^{\rightarrow-\circ\&\top}$ is defined and its main properties are summarized in Section 4.1 (a detailed study is the object of Appendix A). The rest of this section relates the coercion and the spine calculi. More precisely, in Section 4.2 we introduce a mapping of $C^{\rightarrow-\circ\&\top}$ to $S^{\rightarrow-\circ\&\top}$ and prove its soundness with respect to typing. In Section 4.3, 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 4.4 and establish its soundness with respect to reduction in Section 4.5. Sections 4.3 and 4.5 are rather technical; the casual reader should be able to skip them and still follow the overall discussion. Finally, in Section 4.6, we adapt these results to relate $\lambda^{\rightarrow-\circ\&\top}$ and $S^{\rightarrow-\circ\&\top}$.

4.1 The Simply-Typed Lambda Calculus with Explicit Coercions $C^{\rightarrow-\circ\&\top}$

As we observed in [CP97b], NIL-reductions appear as an omnipresent nuisance when investigating the meta-theory of $S^{\rightarrow-\circ\&\top}$, and even more so when relating it to the traditional formulation. In this section, we isolate the source of these difficulties in a mismatch between the syntax of $\lambda^{\rightarrow-\circ\&\top}$ and the inference rules that define its typing semantics. The typing rules of $\lambda^{\rightarrow-\circ\&\top}$, as given in Figure 2.1, do not force a one-to-one correspondence between the structure of a well-typed term and the shape of a derivation tree for it. For example, given c of type $a \,\&\, a$, a derivation for the term FST c may start with the application of rules $\mathbf{l\lambda\_fst}$, $\mathbf{l\lambda\_atm}$, or even $\mathbf{l\lambda\_redex}$. In the first and last case, FST c would be considered pre-atomic, while in the second case it would be pre-canonical. Moreover, there are infinitely many valid typing derivations of either kinds for this term. We can easily trace the source of this multiplicity to rules $\mathbf{l\lambda\_atm}$ and $\mathbf{l\lambda\_redex}$, the only ones that are not bound to a specific construct in the language, making the rules in Figure 2.1 not syntax-directed. In this section, we will slightly alter the definition of $\lambda^{\rightarrow-\circ\&\top}$ to obtain a perfect mapping.

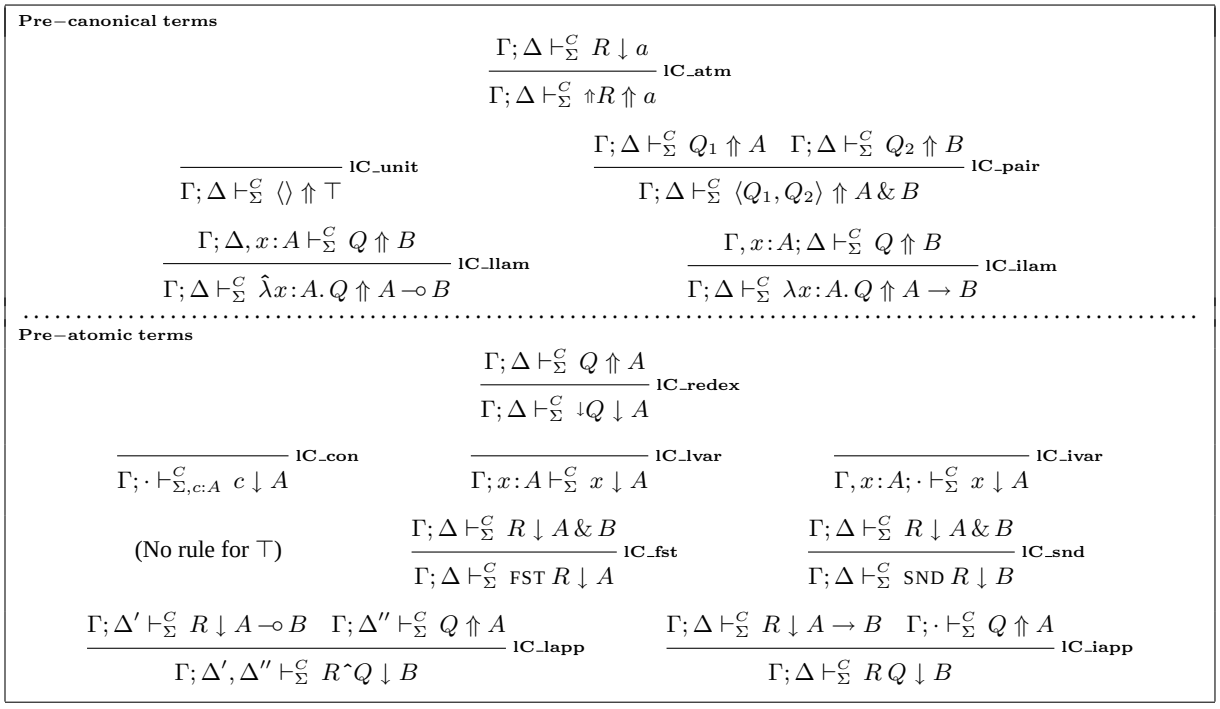


Figure 4.1: Typing for η-long $C^{\to\multimap\&\top}$ Terms

The *coercion calculus* $C^{\to\multimap\&\top}$ extends the syntax of $\lambda^{\to\multimap\&\top}$ with the two *coercion operators* $\Uparrow\_$ and $\downarrow\_$, intended to witness the application of rules $\mathbf{l\lambda\_atm}$ and $\mathbf{l\lambda\_redex}$, respectively. This allows syntactically discriminating terms that are either pre-canonical or pre-atomic (when at all typable). In order to avoid confusion, we use the letter Q, variously annotated, for the former and the letter R for the latter. We will write the other constructs as in $\lambda^{\to\multimap\&\top}$. We define *pre-canonical* and *pre-atomic terms* mutually recursively by means of the following grammar:

Pre-canonical terms:		*Pre-atomic terms:*		
$Q ::=$		$R ::= c \mid x$		
$\mid$	$\lambda x{:}A.\, Q$	$\mid$	$R\,Q$	(*unrestricted functions*)
$\mid$	$\hat{\lambda} x{:}A.\, Q$	$\mid$	$R\,\hat{}\,Q$	(*linear functions*)
$\mid$	$\langle Q_1, Q_2\rangle$	$\mid$	$\mathrm{FST}\ R \mid \mathrm{SND}\ R$	(*additive pairs*)
$\mid$	$\langle\rangle$			(*additive unit*)
$\mid$	$\downarrow R$	$\mid$	$\Uparrow Q$	(*coercions*)

We adopt the same syntactic conventions as for $\lambda^{\to\multimap\&\top}$, but write T for a term that can be either pre-canonical or pre-atomic. As for $\lambda^{\to\multimap\&\top}$ and $S^{\to\multimap\&\top}$, we write $[R/x]T$ for the substitution of R for x in T. Observe however that, since x is pre-atomic, so must be the substituting term R.

The typing semantics of $C^{\to\multimap\&\top}$ is expressed by the judgments

$$\Gamma;\Delta \vdash^C_\Sigma\ Q \Uparrow A \qquad Q \textit{ is a pre-canonical term of type } A \textit{ in } \Gamma;\Delta \textit{ and } \Sigma$$
$$\Gamma;\Delta \vdash^C_\Sigma\ R \downarrow A \qquad R \textit{ is a pre-atomic term of type } A \textit{ in } \Gamma;\Delta \textit{ and } \Sigma$$

and is given in Figure 4.1. It differs from the typing rule set of $\lambda^{\to\multimap\&\top}$ only by the fact that rules $\mathbf{lC\_atm}$ and $\mathbf{lC\_redex}$ are only applicable to terms of the form $\Uparrow R$ and $\downarrow Q$, respectively. Whenever a property holds uniformly for the pre-canonical and pre-atomic judgments above, we will write $\Gamma;\Delta \vdash^C_\Sigma\ T \Uparrow\!\downarrow A$ and then refer to the term T 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. Observe that the typing rules of our

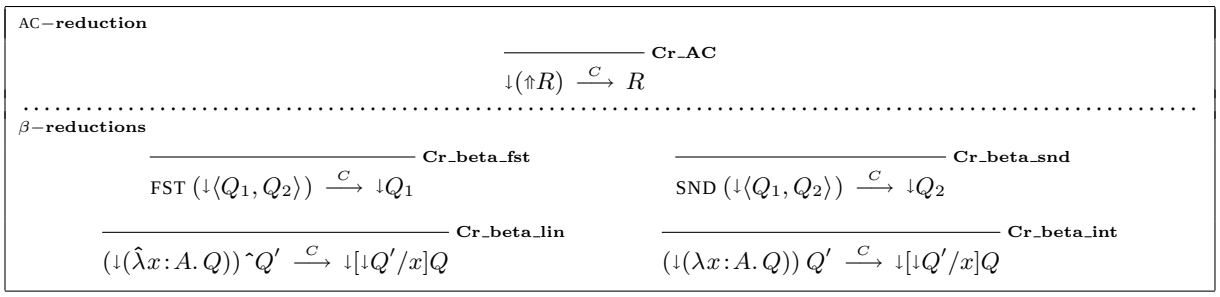


Figure 4.2: Reduction Semantics for $C^{\rightarrow\multimap\&\top}$

intermediate language are syntax-directed. We write $\mathcal{Q}$ and $\mathcal{R}$ for pre-canonical and pre-atomic $C^{\rightarrow\multimap\&\top}$ typing derivations, respectively.

Not surprisingly, $C^{\rightarrow\multimap\&\top}$ satisfies an surjectivity principle similar to $\lambda^{\rightarrow\multimap\&\top}$, but the stricter form of its typing rules limits the form of a pre-canonical term of atomic type to just $\Uparrow R$:

Lemma 4.1 (*Surjectivity*)

i. If $\Gamma;\Delta \vdash^{C}_{\Sigma} Q \Uparrow a$, *then* $Q = \Uparrow R$;

ii. If $\Gamma;\Delta \vdash^{C}_{\Sigma} Q \Uparrow \top$, *then* $Q = \langle\rangle$;

iii. If $\Gamma;\Delta \vdash^{C}_{\Sigma} Q \Uparrow A \,\&\, B$, *then* $Q = \langle Q_1, Q_2\rangle$;

iv. If $\Gamma;\Delta \vdash^{C}_{\Sigma} Q \Uparrow A \multimap B$, *then* $Q = \hat{\lambda}x{:}A.\, Q'$;

v. If $\Gamma;\Delta \vdash^{C}_{\Sigma} Q \Uparrow A \rightarrow B$, *then* $Q = \lambda x{:}A.\, Q'$.

Proof: By inversion on the given derivations. ☑

Porting the reduction semantics of $\lambda^{\rightarrow\multimap\&\top}$ to $C^{\rightarrow\multimap\&\top}$ requires some care since invisible coercions need to be made explicit in the calculus considered in this section. This process is eased by observing how β-reductions operate in $\lambda^{\rightarrow\multimap\&\top}$ at the level of derivations. Consider first a typing derivation for the two sides of rule $\mathbf{lr\_beta\_fst}$:

$$\dfrac{\dfrac{\dfrac{\mathcal{C}_1 \qquad \mathcal{C}_2}{\Gamma;\Delta \vdash_\Sigma M_1 \Uparrow A \qquad \Gamma;\Delta \vdash_\Sigma M_2 \Uparrow B}\,\mathbf{l\lambda_pair}}{\Gamma;\Delta \vdash_\Sigma \langle M_1, M_2\rangle \Uparrow A \,\&\, B}\,\mathbf{l\lambda_redex}}{\Gamma;\Delta \vdash_\Sigma \langle M_1, M_2\rangle \downarrow A \,\&\, B}\,\mathbf{l\lambda_fst}
\quad\longrightarrow\quad
\dfrac{\dfrac{\mathcal{C}_1}{\Gamma;\Delta \vdash_\Sigma M_1 \Uparrow A}\,\mathbf{l\lambda_redex}}{\Gamma;\Delta \vdash_\Sigma M_1 \downarrow A}$$

We need to account for the uses of rule $\mathbf{l\lambda\_redex}$ in devising the corresponding $C^{\rightarrow\multimap\&\top}$ reduction rule by including the $\downarrow\_$ coercion in the appropriate places. So, if M_i correspond to R_i, for $i = 1, 2$, we obtain the following rule:

$$\textsc{fst}\,(\downarrow\langle Q_1, Q_2\rangle) \;\xrightarrow{C}_{\beta}\; \downarrow Q_1$$

We similarly adapt rule $\mathbf{lr\_beta\_snd}$ as follows:

$$\textsc{snd}\,(\downarrow\langle Q_1, Q_2\rangle) \;\xrightarrow{C}_{\beta}\; \downarrow Q_2$$

We construct $C^{\rightarrow\multimap\&\top}$ reduction rules for the functional forms of β-reduction in a similar way and obtain the following $C^{\rightarrow\multimap\&\top}$ reduction rules corresponding to $\mathbf{lr\_beta\_int}$ and $\mathbf{lr\_beta\_lin}$, respectively:

$$(\downarrow(\lambda x{:}A.\, Q))\, Q' \;\xrightarrow{C}_{\beta}\; \downarrow[\downarrow Q'/x]Q$$
$$(\downarrow(\hat{\lambda}x{:}A.\, Q))\hat{}Q' \;\xrightarrow{C}_{\beta}\; \downarrow[\downarrow Q'/x]Q$$

We write $\xrightarrow{C}_\beta$ for the congruence relation based on these four rules. We overload this notation to apply to both pre-atomic and pre-canonical terms. We denote the reflexive and transitive closure of this relation as $\xrightarrow{C}{}^*_\beta$ and write $\overset{C}{\equiv}_\beta$ for the associated equivalence relation. Formal inference rules for $\xrightarrow{C}_\beta$ are given in the lower segment of Figure 4.2, augmented by the straightforward congruence rules. We write $\mathcal{E}_\beta$, possibly superscripted, for derivations of these judgments.

The presence of explicit coercions in $C^{\rightarrow-\circ\&\top}$ makes one more reduction necessary. In the traditional formulation, alternations of rules $\mathbf{l\lambda\_atm}$ and $\mathbf{l\lambda\_redex}$ can be eliminated without affecting the derivability of a judgment:

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\Gamma;\Delta \vdash_\Sigma M \downarrow a}}{\Gamma;\Delta \vdash_\Sigma M \Uparrow a}\,\mathbf{l\lambda\_atm}}{\Gamma;\Delta \vdash_\Sigma M \downarrow a}\,\mathbf{l\lambda\_redex} \quad \dashrightarrow \quad \dfrac{\mathcal{A}}{\Gamma;\Delta \vdash_\Sigma M \downarrow a}$$

This transformation, invisible at the level of $\lambda^{\rightarrow-\circ\&\top}$ terms, is captured by the following AC-*reduction* in $S^{\rightarrow-\circ\&\top}$:

$$\beta_{ac}: \quad \downarrow(\Uparrow R) \xrightarrow{C}_{\text{AC}} R$$

We call the expression on the left-hand side of the arrow a *coercion redex* or an AC-*redex*. We write $\xrightarrow{C}_{\text{AC}}$ for the congruence relation induced by this rule, $\xrightarrow{C}{}^*_{\text{AC}}$ for its reflexive and transitive closure, $\overset{C}{\equiv}_{\text{AC}}$ for the corresponding equivalence relation, and $\mathcal{E}_{\text{AC}}$ for their derivations. The upper part of Figure 4.2 formalizes $\xrightarrow{C}_{\text{AC}}$.

AC-redices and the reduction they induce are closely related to the NIL-redices and the NIL-reduction rule of $S^{\rightarrow-\circ\&\top}$, as we will see in what faollows. $C^{\rightarrow-\circ\&\top}$ offers a simple setting where to study the meta-theory of this relation, which we will port to the more complex world of spines and roots in Section 5. In order to keep the size of this section reasonable, we limit the present discussion to listing the main properties of $\xrightarrow{C}_{\text{AC}}$. The interested reader can find formal statements and proofs in Appendix A.

AC-reduction enjoys many of the desirable properties of a reduction relation for a typed λ-calculus. Not only does AC-reduction preserve typing (Lemma A.1), but this property holds also when using rule $\mathbf{Cr\_AC}$ as an expansion rule (Lemma A.2). Viewed as a rewrite system, $\xrightarrow{C}_{\text{AC}}$ and its derivatives are strongly normalizing (Lemma A.3) and confluent (Lemma A.5), and therefore admit unique normal forms (Lemma A.6). A term T is AC-*normal* if it does not contain any AC-redex, *i.e.*, a subterm of the form $\downarrow\Uparrow R$. We write $C_0^{\rightarrow-\circ\&\top}$ for the sublanguage of $C^{\rightarrow-\circ\&\top}$ that consists only of AC-normal terms.

We conclude our present discussion of the semantics of $C^{\rightarrow-\circ\&\top}$ by analyzing how rule $\mathbf{Cr\_AC}$ interacts with β-reductions. First observe that performing a β-reduction has often (but not always) the effect of exposing an AC-redex:

$$\text{FST}\,(\text{FST}\downarrow\langle\langle\Uparrow c,\Uparrow d\rangle,\Uparrow e\rangle) \xrightarrow{C}_\beta \text{FST}\downarrow\langle\Uparrow c,\Uparrow d\rangle \xrightarrow{C}_\beta \downarrow(\Uparrow c) \xrightarrow{C}_{\text{AC}} c$$

The reverse property does not hold in general, but a β-reduction can be blocked by the presence of an AC-redex:

$$\text{FST}\downarrow\Uparrow\downarrow\langle\Uparrow c,\Uparrow d\rangle \xrightarrow{C}_{\text{AC}} \text{FST}\downarrow\langle\Uparrow c,\Uparrow d\rangle \xrightarrow{C}_\beta \downarrow(\Uparrow c) \xrightarrow{C}_{\text{AC}} c$$

We write $\xrightarrow{C}$ for the union of $\xrightarrow{C}_\beta$ and $\xrightarrow{C}_{\text{AC}}$. It is the congruence relation obtained by allowing the use of both β- and AC-reductions. This is the relation we will use as the basis of the reduction semantics of $C^{\rightarrow-\circ\&\top}$, which is studied in detail in Appendix A. We reserve $\xrightarrow{C}{}^*$ for its reflexive and transitive closure, and $\overset{C}{\equiv}$ for the corresponding equivalence relation. We write $\mathcal{E}$, possibly decorated, for derivations of these judgments. The definition of $\xrightarrow{C}$ is displayed in Figure 4.2, except for the straightforward congruence rules. As for $\lambda^{\rightarrow-\circ\&\top}$ and $S^{\rightarrow-\circ\&\top}$, the rules obtained from this figure by replacing $\xrightarrow{C}$ with $\xrightarrow{C}{}^*$ are admissible. A $C^{\rightarrow-\circ\&\top}$ term that does not contain any β- or AC-redex is called *normal*. By the above surjectivity property, every well-typed normal term is η-long (and as usual we will write *canonical* when emphasizing this point). It is easy to prove that a pre-canonical term Q is normal if and only if it does not contain any occurrences of the coercion $\downarrow\_$. Normal pre-atomic terms may mention this construct, but only as their outermost operator.

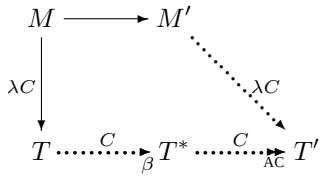
Extending $\lambda^{\rightarrow-\circ\&\top}$ with coercions to obtain $C^{\rightarrow-\circ\&\top}$ can be motivated independently from typing considerations: we insert the atomic-to-canonical coercion $\Uparrow\_$ every time the immediate subterm of a constructor starts with

a destructor (or is a constant or a variable). Dually, the canonical-to-atomic coercion $\downarrow_{\_}$ mediates the transition from destructors to constructors (except for the argument of the applications).

The type-free translation λC of $\lambda^{\to\multimap\&\top}$ into $C^{\to\multimap\&\top}$, given in Appendix A, formalizes this idea by means of the judgments

$M \xrightarrow{\lambda C} Q$ *M translates to pre-canonical term Q*
$M \xrightarrow{\lambda C} R$ *N translates to pre-atomic term R*

We prove in Appendix A that these mutually recursive judgments transform any $\lambda^{\to\multimap\&\top}$ term into an AC-normal pre-canonical and pre-atomic $C^{\to\multimap\&\top}$ object, respectively (Lemma A.10). Therefore, the range of this function is $C_0^{\to\multimap\&\top}$ rather than the full $C^{\to\multimap\&\top}$. Other important properties of λC are that it preserves typing: if $M \xrightarrow{\lambda C} T$ and either M or T is well-typed, then the other side of the arrow is typable as well (Theorem A.8 and Corollary A.19). This translation preserves reducibility (Theorem A.13 and Corollary A.27), but exposes the interplay of β- and AC-reductions in $C^{\to\multimap\&\top}$. The soundness of λC with respect to reducibility (Theorem A.13) is described by the following diagram:



where derivations given as assumptions are represented with full lines, while derivations whose existence is postulated 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. Here, T is AC-normal, but emulating the reduction $M \longrightarrow M'$ occurring in $\lambda^{\to\multimap\&\top}$ may produce a term T^* that is not in AC-normal form: it takes one or more AC-reductions to reconcile this term with the translation of M'.

The reverse translation $C\lambda$, from $C^{\to\multimap\&\top}$ to $\lambda^{\to\multimap\&\top}$, simply erases every coercion. We show in Appendix A that it is the inverse of λC modulo AC-equivalence (Corollary A.18), and that it preserves typing (Theorem A.14 and Corollary A.19) and reductions (Corollaries A.23 and A.26). In particular, AC-equivalent terms are mapped to the same $\lambda^{\to\multimap\&\top}$ term (Lemmas A.21 and A.22).

Readers interested in $C^{\to\multimap\&\top}$, its relation to $\lambda^{\to\multimap\&\top}$ and its meta-theory can find additional information in Appendix A.

4.2 CS: A Translation from $C^{\to\multimap\&\top}$ to $S^{\to\multimap\&\top}$

The translation from $C^{\to\multimap\&\top}$ to $S^{\to\multimap\&\top}$, abbreviated CS, maps the pre-canonical and pre-atomic terms of $C^{\to\multimap\&\top}$ to the roots, terms and spines of $S^{\to\multimap\&\top}$. CS is specified by means of the following judgments:

$Q \xrightarrow{CS} U$ *Pre-canonical term Q translates to U*
$R \setminus S \xrightarrow{CS} U$ *Pre-atomic term R translates to U, given spine S*

The rules defining them are displayed in Figure 4.3. We will denote derivations of either judgments with $\mathcal{CS}$, possibly annotated. When translating a pre-atomic $C^{\to\multimap\&\top}$ term R by means of the second judgment, the spine S acts as an accumulator for the destructors appearing in R. This indirection is needed to cope with the opposite associativity of spines in $S^{\to\multimap\&\top}$ and destructor nesting in $C^{\to\multimap\&\top}$. Notice that, for each of the two judgments of CS, the structure of the first argument determines uniquely which rule can be used in the translation process. This, together with the fact that every production in grammar of $C^{\to\multimap\&\top}$ is covered, ensures that these judgments implement a function, *i.e.*, that every term has a unique translation:

Lemma 4.2 (*Functionality of CS*)

i. For every pre-canonical term Q in $C^{\to\multimap\&\top}$, there is a unique term U in $S^{\to\multimap\&\top}$ such that $Q \xrightarrow{CS} U$.

ii. For every pre-atomic term R in $C^{\to\multimap\&\top}$ and every spine S there is a unique term U in $S^{\to\multimap\&\top}$ such that $R \setminus S \xrightarrow{CS} U$.

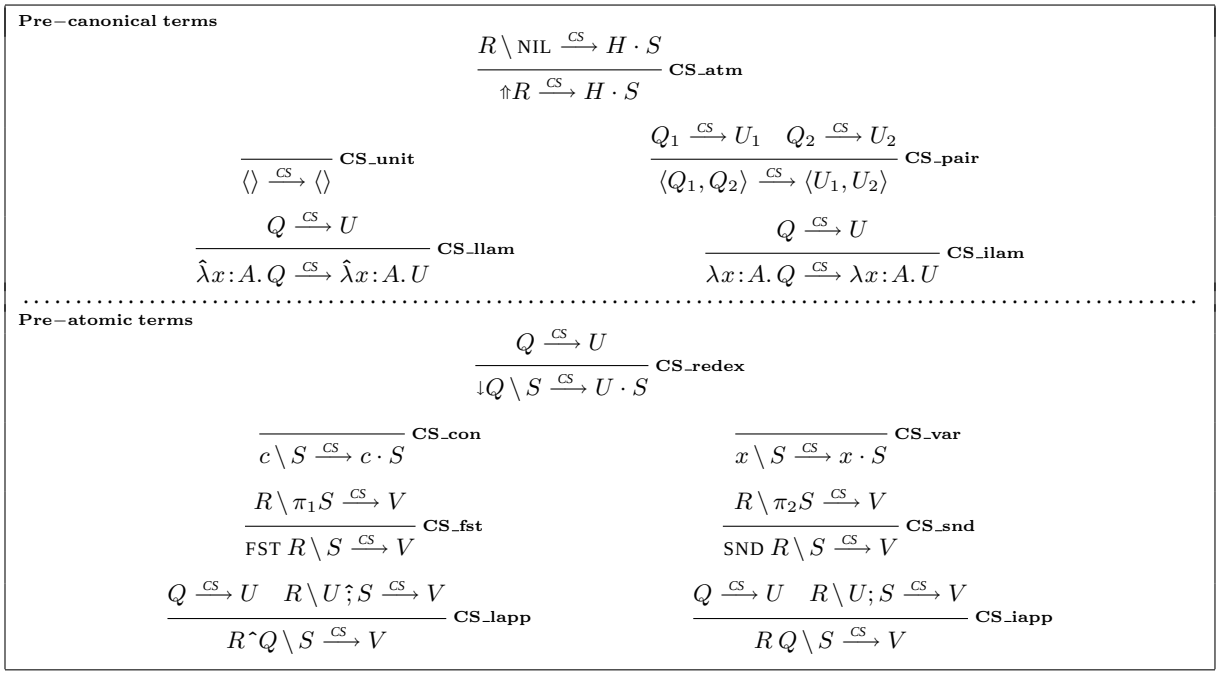


Figure 4.3: Translation of $C^{\rightarrow -\circ \& \top}$ into $S^{\rightarrow -\circ \& \top}$

Proof: By induction on the structure of Q and R. ☑

We can immediately prove the faithfulness of this translation with respect to typing. This result expresses the adequacy of the system in Figure 3.1 as an emulation of the typing semantics of $C^{\rightarrow -\circ \& \top}$. 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 $\mathcal{J} :: J$.

Theorem 4.3 (*Soundness of CS for typing*)

i. If $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} Q \Uparrow A$ *and* $Q \xrightarrow{\text{CS}} U$, *then* $\Gamma; \Delta \vdash_{\Sigma} U : A$;

ii. if $\mathcal{R} :: \Gamma; \Delta_1 \vdash^{C}_{\Sigma} R \downarrow A$, $\Gamma; \Delta_2 \vdash_{\Sigma} S : A > a$ *and* $R \setminus S \xrightarrow{\text{CS}} V$, *then* $\Gamma; \Delta_1, \Delta_2 \vdash_{\Sigma} V : a$.

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

Notice that this statement implies not only that types are preserved during the translation process, but also, by virtue of surjectivity, that η-long objects of $C^{\rightarrow -\circ \& \top}$ are mapped to η-long terms in the spine calculus. We will obtain an indirect proof of the completeness of CS with respect to typing in Section 4.4.

4.3 Soundness of CS with respect to Reduction

We have seen in the previous section that CS is sound with respect to the typing semantics of $C^{\rightarrow -\circ \& \top}$ and $S^{\rightarrow -\circ \& \top}$. We dedicate the present section to proving that it also preserves reductions. This task is complicated by the fact that the reduction semantics of $S^{\rightarrow -\circ \& \top}$, in particular rule **Sr_nil** is specialized to η-long forms. Consider for example the $C^{\rightarrow -\circ \& \top}$ term

$$R = (\downarrow \lambda x{:}a.\, \Uparrow (f \Uparrow x)) \Uparrow c \Uparrow d$$

for a signature with $f : a \rightarrow a \rightarrow a$, $c : a$ and $d : a$. This term is not in η-long form (its η-expansion is $(\downarrow \lambda x{:}a.\, \lambda y{:}a.\, \Uparrow (f \Uparrow x \Uparrow y)) \Uparrow c \Uparrow d$). One β- and two AC-reductions rewrite R to the canonical form

$$R' = f \Uparrow c \Uparrow d$$

whose translation in $S^{\rightarrow\multimap\&\top}$ (given the initial auxiliary spine NIL) is

$$V = f \cdot ((c \cdot \text{NIL}); (d \cdot \text{NIL}); \text{NIL})$$

On the other hand, CS would translate R to the $S^{\rightarrow\multimap\&\top}$ term

$$U = (\lambda x\!:\!a.\, f \cdot (x \cdot \text{NIL}); \text{NIL}) \cdot ((c \cdot \text{NIL}); (d \cdot \text{NIL}); \text{NIL})$$

which cannot be reduced further than

$$(f \cdot (c \cdot \text{NIL}); \text{NIL}) \cdot ((d \cdot \text{NIL}); \text{NIL}),$$

a different term from V. We can recover V by appending the spines, as described at the end of Section 3.

As we can see from this example, CS does not commute with reduction in the general case. We can track the problem to the fact that, while β-reduction and surjectivity are orthogonal concepts in $\lambda^{\rightarrow\multimap\&\top}$ and to a large extent in $C^{\rightarrow\multimap\&\top}$ (see Appendix A), they are intimately related in $S^{\rightarrow\multimap\&\top}$. However, as long as we are interested only in η-long terms, the definitions given in the previous section ensure the CS 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 that certain $C^{\rightarrow\multimap\&\top}$ terms in our statements be well-typed. This is stricter than needed, but typing is the only way we can enforce surjectivity and extensionality.

Rules **Cr_beta_lin** and **Cr_beta_int** generate their reduct by means of a meta-level substitution. The corresponding reduction in $S^{\rightarrow\multimap\&\top}$ operate in a similar way. Therefore, we need to show that CS commutes with substitution.

Lemma 4.4 (*Substitution in CS*)

i. If $\mathcal{CS} :: Q' \xrightarrow{\text{CS}} U$ *and* $\mathcal{CS}_Q :: Q \xrightarrow{\text{CS}} V$, *then* $[\downarrow Q/x]Q' \xrightarrow{\text{CS}} [V/x]U$.

ii. If $\mathcal{CS} :: R \setminus S \xrightarrow{\text{CS}} U$ *and* $\mathcal{CS}_Q :: Q \xrightarrow{\text{CS}} V$, *then* $[\downarrow Q/x]R \setminus [V/x]S \xrightarrow{\text{CS}} [V/x]U$.

Proof:

The proof proceeds by induction on the structure of $\mathcal{CS}$. All cases are quite simple. We will analyze the cases where $\mathcal{CS}$ ends with rules $\mathbf{CS\_redex}$ and $\mathbf{CS\_var}$ (limited to the subcase where the variable in question is precisely x) to familiarize the reader with reasoning within the spine calculus.

$$\mathbf{CS\_redex} \quad \mathcal{CS} = \frac{\mathcal{CS}'}{\dfrac{Q' \xrightarrow{\text{CS}} U'}{\downarrow Q' \setminus S \xrightarrow{\text{CS}} U' \cdot S}\,\mathbf{CS\_redex}}$$

where $R = \downarrow Q'$ and $U = U' \cdot S$.

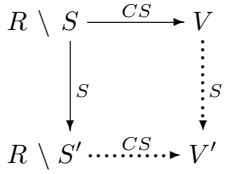
$\mathcal{CS}^{**} :: [\downarrow Q/x]Q' \xrightarrow{\text{CS}} [V/x]U'$	by induction hypothesis (i) on $\mathcal{CS}'$,
$\mathcal{CS}^{*} :: [\downarrow Q/x](\downarrow Q') \setminus [V/x]S \xrightarrow{\text{CS}} [V/x]U' \cdot [V/x]S$	by rule $\mathbf{CS\_redex}$ on $\mathcal{CS}^{**}$,
$[\downarrow Q/x](\downarrow Q') \setminus [V/x]S \xrightarrow{\text{CS}} [V/x](U' \cdot S)$	by definition of substitution.

$$\mathbf{CS\_var} \quad \mathcal{CS} = \frac{}{x \setminus S \xrightarrow{\text{CS}} x \cdot S}\,\mathbf{CS\_var}$$

where $R = x$ and $U = x \cdot S$.

$\mathcal{CS}^{*} :: \downarrow Q \setminus [V/x]S \xrightarrow{\text{CS}} V \cdot [V/x]S$	by rule $\mathbf{CS\_redex}$ on $\mathcal{CS}_Q$,
$[\downarrow Q/x]x \setminus [V/x]S \xrightarrow{\text{CS}} [V/x](x \cdot S)$	by definition of substitution. ☑

We need one more technical result prior to tackling the main theorem of this section. More precisely, we show that, when translating a pre-atomic term, reductions within the corresponding spine are mapped directly to reductions in the resulting $S^{\rightarrow\multimap\&\top}$ term, as expressed by the diagram on the right. In particular, β-reductions are mapped to β-reductions and NIL-reductions yield NIL-reductions.



Lemma 4.5 (*Spine reduction*)

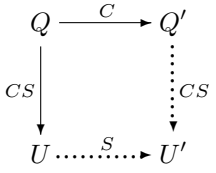
If $\mathcal{CS} :: R \setminus S \xrightarrow{\text{CS}} V$ *and* $\mathcal{F} :: S \xrightarrow{S} S'$*, then there is a term* V' *such that* $R \setminus S' \xrightarrow{\text{CS}} V'$ *and* $V \xrightarrow{S} V'$*.*

Proof:

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

It is easy to show that this result remains valid when considering the transitive and reflexive closure of the involved relations.

At this point, we are in a position to prove that CS is sound with respect to the reduction semantics of $C^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$, one of the few points where we need typing to ensure surjectivity. Note that in applications this property will always be satisfied. It is expressed by the diagram on the right.



Theorem 4.6 (*Soundness of* CS *for reducibility*)

i. If $\mathcal{E} :: Q \xrightarrow{C} Q'$ *and* $\mathcal{CS} :: Q \xrightarrow{\text{CS}} U$ *with* $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} Q \Uparrow A$*, then there is a term* U' *such that* $U \xrightarrow{S} U'$ *and* $Q' \xrightarrow{\text{CS}} U'$*.*

ii. If $\mathcal{E} :: R \xrightarrow{C} R'$ *and* $\mathcal{CS} :: R \setminus S \xrightarrow{\text{CS}} U$ *with* $\mathcal{R} :: \Gamma; \Delta \vdash^{C}_{\Sigma} R \downarrow A$ *and* $\mathcal{S} :: \Gamma; \Delta' \vdash_{\Sigma} S : A > a$*, then there is a term* U' *such that* $U \xrightarrow{S} U'$ *and* $R' \setminus S \xrightarrow{\text{CS}} U'$*.*

Proof:

This simple proof proceeds by induction on the structure of $\mathcal{E}$ and inversion on $\mathcal{CS}$. We will develop the cases where the last rule applied in $\mathcal{E}$ is **Cr_beta_lin**, **Cr_AC** or **Cr_lapp2**, the second congruence rule for linear application.

$$\textbf{Cr\_beta\_lin} \quad \mathcal{E} = \frac{}{(\downarrow(\hat{\lambda}x\!:\!A.\,Q_1))\,\hat{}\,Q_2 \xrightarrow{C} \downarrow[\downarrow Q_2/x]Q}\ \textbf{Cr\_beta\_lin}$$

where $R = (\downarrow(\hat{\lambda}x\!:\!A.\,Q_1))\,\hat{}\,Q_2$ and $R' = \downarrow[\downarrow Q_2/x]Q$.

By inversion on the structure of $\mathcal{CS}$, we obtain that there exist terms U_1 and U_2 such that $U = (\hat{\lambda}x : A.\,U_1) \cdot (U_2 \,\hat{;}\, S)$ and there are derivations $\mathcal{CS}_1$ and $\mathcal{CS}_2$ such that $\mathcal{CS}_1 :: Q_1 \xrightarrow{\text{CS}} U_1$ and $\mathcal{CS}_2 :: Q_2 \xrightarrow{\text{CS}} U_2$:

$$\mathcal{CS} = \dfrac{\mathcal{CS}_2 \quad \dfrac{\dfrac{\mathcal{CS}_1}{Q_1 \xrightarrow{\text{CS}} U_1}}{\hat{\lambda}x\!:\!A.\,Q_1 \xrightarrow{\text{CS}} \hat{\lambda}x\!:\!A.\,U_1}\ \textbf{CS\_llam}}{Q_2 \xrightarrow{\text{CS}} U_2 \qquad \downarrow\hat{\lambda}x\!:\!A.\,Q_1 \setminus U_2 \,\hat{;}\, S \xrightarrow{\text{CS}} (\hat{\lambda}x\!:\!A.\,U_1) \cdot (U_2 \,\hat{;}\, S)}\ \textbf{CS\_redex}}{(\downarrow(\hat{\lambda}x\!:\!A.\,Q_1))\,\hat{}\,Q_2 \setminus S \xrightarrow{\text{CS}} (\hat{\lambda}x\!:\!A.\,U_1) \cdot (U_2 \,\hat{;}\, S)}\ \textbf{CS\_lapp}$$

$\mathcal{CS}^{**} :: [\downarrow Q_2/x]Q_1 \xrightarrow{\text{CS}} [U_2/x]U_1$ by the Substitution Lemma 4.4 on $\mathcal{CS}_1$ and $\mathcal{CS}_2$,

$\mathcal{CS}^{*} :: \downarrow[\downarrow Q_2/x]Q_1 \setminus S \xrightarrow{\text{CS}} [U_2/x]U_1 \cdot S$ by rule **CS_redex** on $\mathcal{CS}^{**}$,

$\mathcal{F} :: (\hat{\lambda}x\!:\!A.\,U_1) \cdot (U_2 \,\hat{;}\, S) \xrightarrow{S} [U_2/x]U_1 \cdot S$ by rule **Sr_beta_lin**.

$$\textbf{Cr\_AC} \quad \mathcal{E} = \frac{}{\downarrow(\Uparrow R') \xrightarrow{C} R'}\ \textbf{Cr\_AC}$$

where $R = \downarrow(\Uparrow R')$.

By inversion on the structure of $\mathcal{CS}$, we have that $U = (H \cdot S') \cdot S$ for spines S and S' and there is a derivation $\mathcal{CS}$ such that $\mathcal{CS}' :: R' \setminus \text{NIL} \xrightarrow{CS} H \cdot S'$:

$$\mathcal{CS} = \dfrac{\dfrac{\mathcal{CS}'}{\dfrac{R' \setminus \text{NIL} \xrightarrow{CS} H \cdot S'}{\Uparrow R' \xrightarrow{CS} H \cdot S'}\,\mathbf{CS\_atm}}}{\downarrow(\Uparrow R') \setminus S \xrightarrow{CS} (H \cdot S') \cdot S}\,\mathbf{CS\_redex}$$

A	$= a'$	and
$\mathcal{Q}'$	$:: \Gamma; \Delta \vdash^{C}_{\Sigma} R' \downarrow a'$	by inversion on rule $\mathbf{lC\_redex}$ and $\mathbf{lC\_atm}$ for $\mathcal{R}$,
a'	$= a$	and
S	$= \text{NIL}$	by inversion on $\mathcal{S}$,
$\mathcal{F}$	$:: (H \cdot S') \cdot \text{NIL} \xrightarrow{S} H \cdot S'$	by rule $\mathbf{Sr\_nil}$.

$$\mathbf{Cr\_lapp2} \qquad \mathcal{E} = \dfrac{\dfrac{\mathcal{E}'}{Q_2 \xrightarrow{C} Q_2'}}{R_1 \hat{} Q_2 \xrightarrow{C} R_1 \hat{} Q_2'}\,\mathbf{Cr\_lapp2}$$

where $R = R_1 \hat{} Q_2$ and $R' = R_1 \hat{} Q_2'$.

$\mathcal{CS}_2$	$:: Q_2 \xrightarrow{CS} U_2$	and
$\mathcal{CS}_1$	$:: R_1 \setminus U_2 \hat{;} S \xrightarrow{CS} U$	by inversion on rule $\mathbf{CS\_iapp}$ for $\mathcal{CS}$,
A	$= A_2 \multimap A_1$	and
$\mathcal{Q}'$	$:: \Gamma; \Delta_2 \vdash^{C}_{\Sigma} Q_2 \Uparrow A_2$	by inversion on rule $\mathbf{lC\_lapp}$ for $\mathcal{R}$,
$\mathcal{F}'$	$:: U_2 \xrightarrow{S} U_2'$	and
$\mathcal{CS}_2'$	$:: Q_2' \xrightarrow{CS} U_2'$	by induction hypothesis (i) on $\mathcal{E}'$, $\mathcal{CS}_2$ and $\mathcal{Q}'$,
$\mathcal{F}''$	$:: U_2 \hat{;} S \xrightarrow{S} U_2' \hat{;} S$	by rule congruence from $\mathcal{F}'$,
$\mathcal{CS}_1'$	$:: R_1 \setminus U_2' \hat{;} S \xrightarrow{CS} U'$	and
$\mathcal{F}$	$:: U \xrightarrow{S} U'$	by the Spine Reduction Lemma 4.5 on $\mathcal{CS}_1$ and $\mathcal{F}''$,
$\mathcal{CS}^*$	$:: R_1 \hat{} Q_2' \setminus S \xrightarrow{CS} U'$	by rule $\mathbf{Cr\_lapp2}$ on $\mathcal{CS}_1'$ and $\mathcal{CS}_2'$. ☑

The typing assumptions appearing in the statement of this theorem would be unnecessary if we were to replace our NIL-reduction with the generalized version that allows appending spines, *i.e.*, if we were to give up on our surjectivity requirements.

A careful inspection of this proof reveals CS maps β-reductions in $C^{\rightarrow \multimap \&\top}$ to β-reductions in $S^{\rightarrow \multimap \&\top}$, and AC-reductions to NIL-reductions; moreover, no typing derivation is needed in the former case:

Corollary 4.7 (*Soundness of CS for reducibility*)

i. If $Q \xrightarrow{C}_{\beta} Q'$ and $Q \xrightarrow{CS} U$, then there is a term U' such that $U \xrightarrow{S}_{\beta} U'$ and $Q' \xrightarrow{CS} U'$.

ii. If $Q \xrightarrow{C}_{\text{AC}} Q'$ and $Q \xrightarrow{CS} U$ with $\Gamma; \Delta \vdash^{C}_{\Sigma} Q \Uparrow A$, then there is a term U' such that $U \xrightarrow{S}_{\text{NIL}} U'$ and $Q' \xrightarrow{CS} U'$.

The notion of soundness we adopted relative to the reduction semantics of our calculi ensures that every reduction in the source language correspond to one reduction in the target language. We define completeness dually: every reduction in the target language should correspond to some reduction in the source language. We will give an indirect proof of the completeness of CS with respect to the reduction semantics of our calculi in Section 4.5, when considering the inverse of our translation.

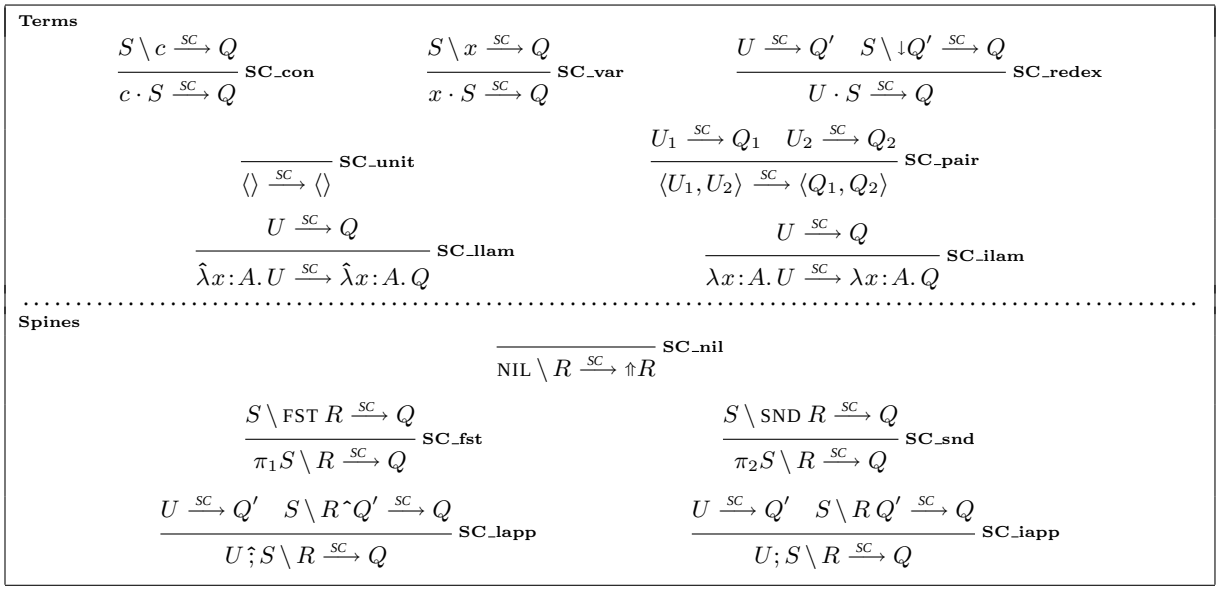


Figure 4.4: Translation of $S^{\rightarrow -\circ \& \top}$ into $C^{\rightarrow -\circ \& \top}$

4.4 SC: A Translation from $S^{\rightarrow -\circ \& \top}$ to $C^{\rightarrow -\circ \& \top}$

In this section and in the next, we consider the problem of translating terms from $S^{\rightarrow -\circ \& \top}$ back to $C^{\rightarrow -\circ \& \top}$, an essential operation to interpret $S^{\rightarrow -\circ \& \top}$ objects in the usual notation. CS cannot be used for this purpose since the rules at the bottom of Figure 4.3 are not syntax-directed with respect to $S^{\rightarrow -\circ \& \top}$ spines. The approach we take is instead to define an independent translation, SC, that maps entities in $S^{\rightarrow -\circ \& \top}$ to terms in $C^{\rightarrow -\circ \& \top}$. We will prove later that it is precisely the inverse of CS. The SC translation is specified by means of the judgments

$U \xrightarrow{SC} Q$ — *U translates to pre-canonical term Q*
$S \setminus R \xrightarrow{SC} Q$ — *S translates to pre-canonical term Q, given pre-atomic seed R*

and defined in Figure 4.4. We write $\mathcal{SC}$, possibly annotated, for derivations of either judgments. The notion of spine does not have a proper equivalent in $C^{\rightarrow -\circ \& \top}$: 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 $C^{\rightarrow -\circ \& \top}$ term. This is achieved by the judgment $S \setminus R \xrightarrow{SC} Q$: the auxiliary term R (the *seed*) is initialized to the translation of some head for the spine S (rules **SC_con**, **SC_var** and **SC_redex**); it is successively used as an accumulator for the translation of the operators appearing in S (rules **SC_fst**, **SC_snd**, **SC_lapp** and **SC_iapp**); when the empty spine is eventually reached (rule **SC_nil**), the overall translation has been completed and Q is returned. As in CS, the use of an accumulator handles the opposite associativity of $S^{\rightarrow -\circ \& \top}$ and $C^{\rightarrow -\circ \& \top}$.

The faithfulness of SC 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 surjectivity.

Theorem 4.8 (*Soundness of SC for typing*)

i. If $\mathcal{U} :: \Gamma; \Delta \vdash_\Sigma U : A$, and $\mathcal{SC} :: U \xrightarrow{SC} Q$, then $\Gamma; \Delta \vdash^C_\Sigma Q \Uparrow A$.

ii. If $\mathcal{S} :: \Gamma; \Delta_1 \vdash_\Sigma S : A > a$, $\mathcal{R} :: \Gamma; \Delta_2 \vdash^C_\Sigma R \downarrow A$ and $\mathcal{SC} :: S \setminus R \xrightarrow{SC} Q$, then $\Gamma; \Delta_1, \Delta_2 \vdash^C_\Sigma Q \Uparrow a$.

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

We dedicate the remainder of this section to proving that SC is the inverse of CS. 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 CS and SC with respect to typing and reduction.

We begin our endeavor by proving that SC is actually a function from $S^{\rightarrow -\circ \& \top}$ to $C^{\rightarrow -\circ \& \top}$.

Lemma 4.9 (*Functionality of SC*)

i. For every $S^{\rightarrow\multimap\&\top}$ term U, there is a unique pre-canonical $C^{\rightarrow\multimap\&\top}$ term Q such that $U \xrightarrow{\text{SC}} Q$.

ii. For every spine S and pre-atomic seed R, there is a unique pre-canonical $C^{\rightarrow\multimap\&\top}$ term Q such that $S \setminus R \xrightarrow{\text{SC}} Q$.

Proof: By induction on the structure of U and S. ☑

Based on this fact, it is easy to show that SC and CS are each other's inverse.

Lemma 4.10 (*Bijectivity*)

CS and SC are bijections between the set of $C^{\rightarrow\multimap\&\top}$ terms and the set of $S^{\rightarrow\multimap\&\top}$ terms. Moreover, they are each other's inverse. More precisely, we have that

i. If $\mathcal{SC} :: U \xrightarrow{\text{SC}} Q$, then $Q \xrightarrow{\text{CS}} U$.

ii. If $\mathcal{SC} :: S \setminus R \xrightarrow{\text{SC}} Q$ and $\mathcal{CS} :: R \setminus S \xrightarrow{\text{CS}} U$, then $Q \xrightarrow{\text{CS}} U$.

iii. If $\mathcal{CS} :: Q \xrightarrow{\text{CS}} U$, then $U \xrightarrow{\text{SC}} Q$.

iv. If $\mathcal{CS} :: R \setminus S \xrightarrow{\text{CS}} U$ and $\mathcal{SC} :: S \setminus R \xrightarrow{\text{SC}} Q$, then $U \xrightarrow{\text{SC}} Q$.

Proof:

The proof that SC is the right-inverse of CS (items *i* and *ii*) proceeds by induction on the structure of $\mathcal{SC}$ (and inversion on $\mathcal{CS}$ when present). The proof that CS is the right-inverse of SC (items *iii* and *iv*) is similarly done by induction on the structure of $\mathcal{CS}$ (and possible inversion on $\mathcal{SC}$).

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

This property opens the door to easy proofs of the completeness direction of every soundness theorem achieved so far. We first consider the completeness of CS with respect to typing. In this and other results below, we refrain from presenting an auxiliary proposition relating pre-atomic terms and spines.

Corollary 4.11 (*Completeness of CS for typing*)

If $Q \xrightarrow{\text{CS}} U$ and $\Gamma; \Delta \vdash_\Sigma U : A$, then $\Gamma; \Delta \vdash_\Sigma Q \Uparrow A$.

Proof:

By the Bijectivity Lemma 4.10, $U \xrightarrow{\text{SC}} Q$. Then, the soundness of SC for typing yields a derivation of $\Gamma; \Delta \vdash^C_\Sigma Q \Uparrow A$. ☑

In a similar fashion, we prove the completeness of SC with respect to typing.

Corollary 4.12 (*Completeness of SC for typing*)

If $U \xrightarrow{\text{SC}} Q$ and $\Gamma; \Delta \vdash^C_\Sigma Q \Uparrow A$, then $\Gamma; \Delta \vdash_\Sigma U : A$.

Proof: Similar to the proof of Corollary 4.11. ☑

4.5 Soundness of SC with respect to Reduction

We will now analyze the interaction between SC as a translation from $S^{\rightarrow\multimap\&\top}$ and $C^{\rightarrow\multimap\&\top}$, and the notion of reduction inherent to these two languages. The main results of our investigation will be that SC preserves β-reductions and maps NIL-reductions in $S^{\rightarrow\multimap\&\top}$ to AC-reductions in $C^{\rightarrow\multimap\&\top}$. We will also take advantage of the fact that this translation is the inverse of CS to prove the completeness counterpart of these statements.

The discovery in the previous section that CS and SC are bijective accounts for a simple proof of the completeness of the latter translation with respect to the reduction semantics of the involved calculi.

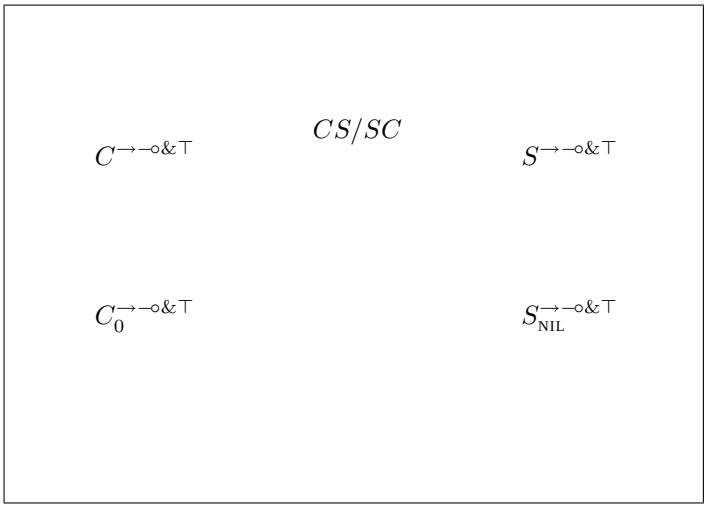
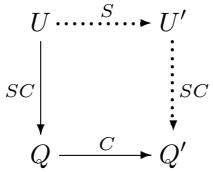


Figure 4.5: CS and SC

Corollary 4.13 (*Completeness of SC for reduction*)

i. If $Q \xrightarrow{C}_{\beta} Q'$ and $U \xrightarrow{\text{SC}} Q$, then there is a term U' such that $U \xrightarrow{S}_{\beta} U'$ and $U' \xrightarrow{\text{SC}} Q'$.

ii. If $U \xrightarrow{\text{SC}} Q$, $Q \xrightarrow{C}_{\text{AC}} Q'$, and $\Gamma;\Delta \vdash_{\Sigma} U : A$, then there is a term U' such that $U \xrightarrow{S}_{\text{NIL}} U'$ and $U' \xrightarrow{\text{SC}} Q'$.



Proof:

We rely on the Bijectivity Lemma 4.10 and the soundness of SC with respect to typing (Theorem 4.8) to reduce the above statement to the Soundness Corollary 4.7 for CS. ☑

We conclude this section by showing that SC is sound with respect to the reduction semantics of $S^{\rightarrow\multimap\&\top}$. The required steps in order to achieve this result are reminiscent of the path we followed when proving the analogous statement for CS. There is however one difference: the statements below do not need to mention any typing information.

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

Lemma 4.14 (*Substitution in SC*)

i. If $\mathcal{Q} :: U \xrightarrow{\text{SC}} Q$ and $\mathcal{Q}_V :: V \xrightarrow{\text{SC}} R$, then $[V/x]U \xrightarrow{\text{SC}} [R/x]Q$.

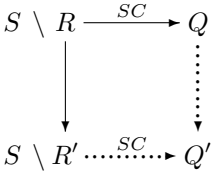
ii. If $\mathcal{Q} :: S \setminus R' \xrightarrow{\text{SC}} Q$ and $\mathcal{Q}_V :: V \xrightarrow{\text{SC}} R$, then $[V/x]S \setminus [R/x]R' \xrightarrow{\text{SC}} [R/x]Q$.

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 4.3.

Lemma 4.15 (*Seed reduction*)

If $\mathcal{SC} :: S \setminus R \xrightarrow{\text{SC}} Q$ and $\mathcal{E} :: R \xrightarrow{C} R'$, then there is a term Q' such that $S \setminus R' \xrightarrow{\text{SC}} Q'$ and $Q \xrightarrow{C} Q'$.



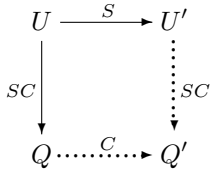
Proof: *By induction on the structure of $\mathcal{SC}$.* ☑

Finally, we have the following soundness theorem, that states that SC preserves reductions.

Theorem 4.16 (*Soundness of SC for reducibility*)

i. If $\mathcal{F} :: U \xrightarrow{S} U'$ and $\mathcal{SC} :: U \xrightarrow{\text{SC}} Q$, then there is a term Q' such that $Q \xrightarrow{C} Q'$ and $U' \xrightarrow{\text{SC}} Q'$.

ii. If $\mathcal{F} :: S \xrightarrow{S} S'$ and $\mathcal{SC} :: S \setminus R \xrightarrow{\text{SC}} Q$, then there is a term Q' such that $Q \xrightarrow{C} Q'$ and $S' \setminus R \xrightarrow{\text{SC}} Q'$.



Proof: By induction on the structure of $\mathcal{SC}$. ☑

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

As in the case of CS, a close inspection of this proof reveals that SC maps the β-reductions of $S^{\rightarrow\multimap\&\top}$ to the β-reductions of $C^{\rightarrow\multimap\&\top}$, and faithfully relates NIL-reductions to AC-reductions. We express this fact in the following corollary.

Corollary 4.17 (*Soundness of SC for reducibility*)

i. If $U \xrightarrow{S}_{\text{NIL}} U'$ and $U \xrightarrow{\text{SC}} Q$, then there is a term Q' such that $Q \xrightarrow{C}_{\text{AC}} Q'$ and $U' \xrightarrow{\text{SC}} Q'$.

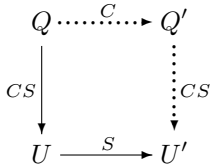
ii. If $U \xrightarrow{S}_{\beta} U'$ and $U \xrightarrow{\text{SC}} Q$, then there is a term Q' such that $Q \xrightarrow{C}_{\beta} Q'$ and $U' \xrightarrow{\text{SC}} Q'$. □

Therefore, CS and SC constitute an isomorphism between $C^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$ that faithfully respects reductions. The operations of this translation are depicted in Figure 4.5.

The previous theorem, together with the fact that SC and CS form a pair of inverse functions, allows us to achieve a simple proof of the completeness of CS with respect to the reduction semantics of $S^{\rightarrow\multimap\&\top}$.

Corollary 4.18 (*Completeness of CS for reduction*)

If $Q \xrightarrow{\text{CS}} U$ and $U \xrightarrow{S} U'$, then there is a term Q' such that $Q \xrightarrow{C} Q'$ and $Q' \xrightarrow{\text{CS}} U'$.



Proof:

By the Bijectivity Lemma 4.10, there is a derivation of $U \xrightarrow{SC} Q$. By the soundness of SC with respect to reduction, there is a term Q' such that $Q \xrightarrow{C} Q'$ and $U' \xrightarrow{SC} Q'$. Again by bijectivity, we have that that $Q' \xrightarrow{CS} U'$ is derivable. ☑

4.6 Relating $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$

In Section 4.1 we introduced $C^{\rightarrow\multimap\&\top}$ as an intermediate formal language aimed at simplifying the analysis of the relation between $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$. We displayed two pairs of inverse translations between the coercion calculus and each of $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$ and proved that they preserved the typing and reduction semantics of these languages. In this section, we will chain these results to relate $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$ directly.

We write the translation λS between $\lambda^{\rightarrow\multimap\&\top}$ and $S^{\rightarrow\multimap\&\top}$ by means of the following judgments:

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

λS is defined as the composition of λC and CS by the following clauses:

$$\begin{array}{llllll} M \xrightarrow{\lambda S} U & \text{iff} & M \xrightarrow{\lambda C} Q & \text{and} & Q \xrightarrow{CS} U \\ M \setminus S \xrightarrow{\lambda S} U & \text{iff} & M \xrightarrow{\lambda C} R & \text{and} & R \setminus S \xrightarrow{CS} U \end{array}$$

for appropriate $C^{\rightarrow\multimap\&\top}$ terms Q and R, that exist since λC is functional in its first argument (Lemma A.9). Since CS is functional as well, our translation is a function. It should be noted that, since λC produces AC-normal $C^{\rightarrow\multimap\&\top}$ terms (Lemma A.10) and CS faithfully preserves reductions (Corollary 4.7), the range of λS is to well-typed $S^{\rightarrow\multimap\&\top}$ terms in NIL-normal form. These properties are summarized in Figure 4.6.

The soundness of λS for typing is a direct consequence of this definition and of the analogous results proved for λC and CS:

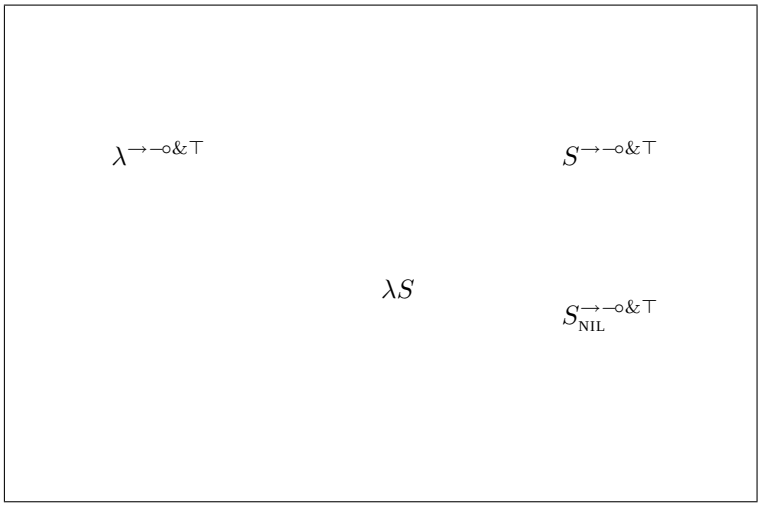


Figure 4.6: λS

Theorem 4.19 (*Soundness of λS for typing*)

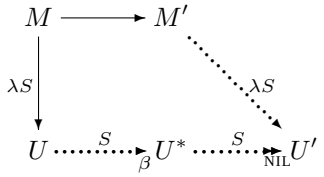
If $\Gamma;\Delta \vdash_\Sigma M \Uparrow A$ *and* $M \xrightarrow{\lambda S} U$, *then* $\Gamma;\Delta \vdash_\Sigma U : A$. □

The associated completeness result is obtained in a similar way.

The soundness of λS with respect to reductions requires some additional care since on the one hand the similar property for λC maps each reduction of $\lambda^{\to\multimap\&\top}$ into a combination of a β-reduction and zero or more AC-reductions in $C^{\to\multimap\&\top}$, and on the other hand CS relies on typing to ensure surjectivity. With these considerations in mind, we obtain the following statement:

Theorem 4.20 (*Soundness of λS for reducibility*)

If $M \longrightarrow M'$, $M \xrightarrow{\lambda S} U$, *and* $\Gamma;\Delta \vdash_\Sigma M \Uparrow A$, *then there are terms* U^* *and* U' *such that* $U \xrightarrow{S}_\beta U^*$, $U^* \xrightarrow{S}_{\text{NIL}}^* U'$ *and* $M' \xrightarrow{\lambda S} U'$. □



Again, the corresponding completeness result is obtained analogously

The definition and analysis of the reverse translation $S\lambda$, from $S^{\to\multimap\&\top}$ back to $\lambda^{\to\multimap\&\top}$, proceeds in the same way. We rely on the judgments

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

defined by

$U \xrightarrow{S\lambda} M$	iff	$U \xrightarrow{SC} Q$		and	$Q \xrightarrow{C\lambda} M$
$S \setminus N \xrightarrow{S\lambda} M$	iff	$N \xrightarrow{\lambda C} R$,	$S \setminus R \xrightarrow{SC} Q$	and	$Q \xrightarrow{C\lambda} M$

Observe that the translation of spines relies on the map λC, from $\lambda^{\to\multimap\&\top}$ from $C^{\to\multimap\&\top}$. In practice, *i.e.*, when this judgment is invoked as a subroutine of $U \xrightarrow{S\lambda} M$, this step is bypassed altogether, as one can observe from the rules in Figure 4.4.

By lemmas 4.9 and A.15, $S\lambda$ is a function and it maps NIL-equivalent objects in $S^{\to\multimap\&\top}$ to the same object in $\lambda^{\to\multimap\&\top}$ (by lemmas A.21 and A.22). This is shown in Figure 4.7. Furthermore, Lemmas 4.10 and A.18 entail that $S\lambda$ and λS form a pair of inverse bijections, when the domain of the former and the range of the later is restricted to well-typed $S^{\to\multimap\&\top}_{\text{NIL}}$ terms.

The soundness and completeness of $S\lambda$ for typing is a consequence of the analogous properties of SC and $C\lambda$. We report here only the statement of soundness.

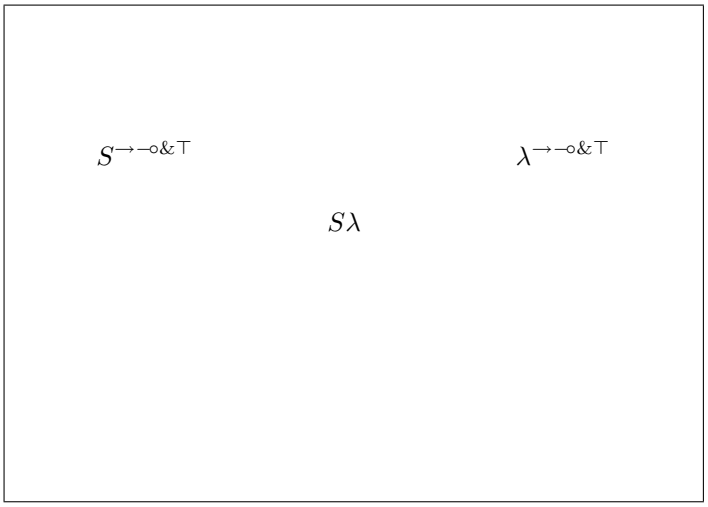


Figure 4.7: $S\lambda$

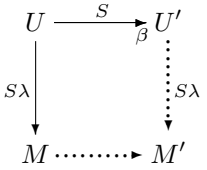
Theorem 4.21 (*Soundness of $S\lambda$ for typing*)

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

Finally, $S\lambda$ enjoys the following soundness result as far as reducibility is concerned.

Theorem 4.22 (*Soundness of $S\lambda$ for β-reducibility*)

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



The associated completeness statement is obtained similarly.

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

In this section, we will deduce the main properties of $S^{\to\multimap\&\top}$ from the analogous results presented for $\lambda^{\to\multimap\&\top}$ in Section 2, or more precisely from the adaptation of those properties to $C^{\to\multimap\&\top}$ analyzed in Appendix A. We will first examine NIL-reducibility and ultimately the existence and uniqueness of NIL-normal forms in Section 5.1. Then, in Section 5.2, we will turn to properties such as the existence of unique normal forms for the complete reduction semantics of $S^{\to\multimap\&\top}$. Finally, in Section 5.3, we hint at further properties related to weak head-normal forms, η-expansion, and equality in $S^{\to\multimap\&\top}$.

5.1 Properties of NIL-Reducibility

In this section, we study the notion of NIL-reducibility, an omnipresent nuisance when directly investigating the meta-theory of $S^{\to\multimap\&\top}$, and when studying direct translations from $\lambda^{\to\multimap\&\top}$. In doing so, we will take advantage of our detailed study of AC-reducibility for $C^{\to\multimap\&\top}$ in Appendix A.

The analysis of the interplay between typing and NIL-reduction reveals that this relation enjoys the subject reduction property. Moreover, the use of NIL-reduction in the reverse direction, *i.e.*, as an expansion rule, preserves typing too. We combine these two results in the following lemma.

Lemma 5.1 (NIL-*reduction/expansion preserves typing*)

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

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

Proof:

Since SC maps NIL-reductions to AC-reductions (Corollary 4.17), we make use of the analogous results for $C^{\rightarrow\multimap\&\top}$ (Lemmas A.21 and A.22) followed by the soundness of CS for typing (Theorem 4.3) to obtain the desired result. ☑

We now concentrate on the properties of $S^{\rightarrow\multimap\&\top}$ 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 $S^{\rightarrow\multimap\&\top}$.

Lemma 5.2 (*Strong* NIL*-normalization*)

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

Proof:

Again, we rely on the fact that SC and CS faithfully relate NIL- and AC-reductions to map this property to $C^{\rightarrow\multimap\&\top}$. It then reduces to lemma A.3. ☑

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 reduced to a common reduct by a further application unless they are already identical. This property can be iterated, as expressed in the following confluence lemma, that applies equally to terms and spines.

Lemma 5.3 (*Confluence*)

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

Proof:

Again, we rely on the faithfulness of CS and SC to express this property in $C^{\rightarrow\multimap\&\top}$, which is then resolved by the corresponding result in corollary A.5. ☑

As already defined in Section 3, 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 5.4 (*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:

This is again an immediate consequence of the analogous result for $C^{\rightarrow\multimap\&\top}$ (lemma A.6). ☑

As said, 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, and write $S^{\rightarrow\multimap\&\top}_{\text{NIL}}$ for the sublanguage of $S^{\rightarrow\multimap\&\top}$ that consists only of NIL-normal terms.

We will take advantage of the following technical result below that states that substitution preserves NIL-reducibility.

Lemma 5.5 (*Substitution*)

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

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

Proof: By induction on the structure of $\mathcal{F}$. ☑

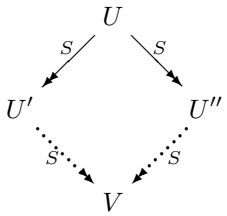
5.2 Properties of Reducibility

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 fact that similar results hold for $C^{\rightarrow\multimap\&\top}$, and that we have well-behaved translations to and from this calculus. The $C^{\rightarrow\multimap\&\top}$ results cited in this section are the subject of Appendix A, where they are deduced from the analogous properties of $\lambda^{\rightarrow\multimap\&\top}$ given in Section 2. An alternative would have been to give direct proofs of these properties.

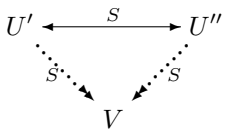
We begin by showing that $S^{\rightarrow\multimap\&\top}$ admits confluence and the Church-Rosser property. Differently from $\lambda^{\rightarrow\multimap\&\top}$ but similarly to $C^{\rightarrow\multimap\&\top}$, the statement of this property must include typing assumptions in order to express certain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

Theorem 5.6 (*Church-Rosser*)

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



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



Proof:

We will carry out a detailed proof in the case of confluence only by mapping this property to $C^{\rightarrow\multimap\&\top}$ thanks to the existence of a reduction- and typing-preserving isomorphism between $S^{\rightarrow\multimap\&\top}$ and this language, and then relying on the analogous property of the coercion calculus proved in Appendix A. The Church-Rosser property is handled similarly.

Since, by Lemma A.15, SC is a total function over $S^{\rightarrow\multimap\&\top}$, there is a unique term Q such that $U \xrightarrow{SC} Q$ is derivable. By typing soundness (Theorem 4.8), we obtain that $\Gamma;\Delta \vdash^C_\Sigma Q \Uparrow A$. By iterated applications of the soundness of SC over reduction (theorem 4.16), we deduce that there are terms Q' and Q'' such that $Q \xrightarrow{C}{}^* Q'$ and $U' \xrightarrow{SC} Q'$, and similarly $Q \xrightarrow{C}{}^* Q''$ and $U'' \xrightarrow{SC} Q''$. By the confluence property of $C^{\rightarrow\multimap\&\top}$ (corollary A.5), we know that there exists a term Q^* such that $Q' \xrightarrow{C}{}^* Q^*$ and $Q'' \xrightarrow{C}{}^* Q^*$ are derivable.

By the bijectivity of SC (Lemma 4.10), $Q' \xrightarrow{CS} U'$ and $Q'' \xrightarrow{CS} U''$. By the soundness of CS with respect to reductions (theorem 4.6), there are terms V' and V'' such that $U' \xrightarrow{S}{}^* V'$ and $Q^* \xrightarrow{CS} V'$, and similarly $U'' \xrightarrow{S}{}^* V''$ and $Q^* \xrightarrow{CS} V''$. However, since CS is a function (Lemma 4.2), $V' = V''$; 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 $S^{\rightarrow\multimap\&\top}$ equivalent of the transitivity Lemma 2.3 discussed in Section 2. As in $\lambda^{\rightarrow\multimap\&\top}$ and $C^{\rightarrow\multimap\&\top}$, we must distinguish the linear and the unrestricted cases, but we have no convenient notation that spans uniformly over terms and spines. Therefore, the lemma below has four parts.

Lemma 5.7 (*Transitivity*)

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

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

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

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

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 **lC_lvar**, there is a (trivial) typing derivation of $\Gamma; z : A \vdash^{C}_{\Sigma} z \downarrow A$. On the basis of this fact, by the soundness of SC for typing (Theorem 4.8), there is a term Q such that $S \setminus z \xrightarrow{SC} Q$ and $\Gamma; \Delta, x : B, z : A \vdash^{C}_{\Sigma} Q \Uparrow a$ are derivable. By the same theorem, there is a term Q' and derivations of $V \xrightarrow{SC} Q'$ and $\Gamma; \Delta' \vdash^{C}_{\Sigma} \downarrow Q' \downarrow B$. By the transitivity lemma A.29 for $C^{\rightarrow\multimap\&\top}$, $\Gamma; \Delta, \Delta', z : A \vdash^{C}_{\Sigma} [\downarrow Q'/x]Q \Uparrow a$ is derivable.

By rule **CS_var**, there is a derivation of $z \setminus S \xrightarrow{CS} z \cdot S$. By the bijectivity lemma 4.10, there is a derivation of $Q \xrightarrow{CS} z \cdot S$. Again by the bijectivity lemma, there is a derivation $Q' \xrightarrow{CS} V$ is derivable. By the substitution lemma 4.4, there is a derivation of $[\downarrow Q'/x]Q \xrightarrow{CS} z \cdot ([V/x]S)$ (remember that $x \neq z$). By the soundness of CS with respect to typing (theorem 4.3), $\Gamma; \Delta, \Delta', z : A \vdash_{\Sigma} z \cdot ([V/x]S) : a$ is derivable. By inversion on rule **lS_lvar**, $\Gamma; \Delta, \Delta' \vdash_{\Sigma} [V/x]S : A > a$ is derivable as well. ☑

The next property we are interested in proving for $S^{\rightarrow\multimap\&\top}$ 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 as Lemma 5.1.

Lemma 5.8 (*Subject reduction*)

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

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

Proof:

Again, we rely on the fact that CS/SC are a typing- and reduction-preserving isomorphism between $S^{\rightarrow\multimap\&\top}$ and $C^{\rightarrow\multimap\&\top}$ to reduce this statement to the analogous result for the coercion calculus (lemma A.30). ☑

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 5.9 (*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 rely again on the techniques already deployed for theorem 5.6 and lemmas 5.7 and 5.8 to reduce this statement to the strong normalizability of $C^{\rightarrow\multimap\&\top}$ (theorem A.31). ☑

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 5.10 (*Uniqueness of normal forms*)

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

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

Proof: Again, this is a consequence of the analogous result for $C^{\rightarrow\multimap\&\top}$ (Corollary A.32). ☑

As in the case of $\lambda^{\rightarrow\multimap\&\top}$ and $C^{\rightarrow\multimap\&\top}$, the above results allow us to speak 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 3.1 by simply removing rule **lS_redex**.

5.3 Other Properties

A number of other properties of the spine calculus are investigated in [CP97a]. Although aimed at the specific task of proving the soundness and completeness of a linear higher-order unification algorithm expressed in $S^{\rightarrow -\circ \& \top}$, many are of general interest. In that paper, we formally define and analyze the notion of (weak) head-normal reduction for the spine calculus [CP97a, Section 2.3]. This relation reduces the superficial redices of a generic term, but is not defined within spines. We show that it is confluent in general and strongly normalizing for well-typed terms, and therefore that typable objects have unique head-normal forms consisting of a superficial layer that is redex-free and a deeper layer that is arbitrary. We also define a lazy algorithm based on them to efficiently check the equality of two $S^{\rightarrow -\circ \& \top}$ terms and spines, and prove its correctness with respect to the simple-minded procedure that relies on canonical forms [CP97a, Section 2.4]. Finally, we study in detail the notion of η-expansion [CP97a, Section 2.5].

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 briefly discuss the implementation (Section 6.1), point out a relationship between the spine calculus and the logic programming notion of uniform derivability (Section 6.2), raise the issue of adding polymorphism (Section 6.3), and discuss related work (Section 6.4).

6.1 Implementations of the Spine Calculus

The most frequent operations on λ-terms in a logical framework or logic programming language are type-checking, equality testing, and unification. While in principle the cost of such operations is dominated by the time required to normalize the terms involved, in practice we mostly deal with terms that are in canonical form. This means term traversal is the most important and time-critical operation. All three principal algorithms (type-checking, equality testing, unification) have to access first the head of an atomic term and then its arguments. As a result we visit each subterm at each layer twice: once while descending to the head, and once when we recursively traverse the arguments. In the spine representation, the term is laid out in the order it is traversed, roughly saving a constant factor of two.

In addition, in an implementation with binary application, we may have to save pointers to the unprocessed arguments on a stack while accessing the head of a term. Recursively, this means we have to maintain a stack of size $O(n)$ instead of order $O(\log(n))$ as for the spine calculus.

The spine calculus has been used in the reimplementation of *Elf* [Pfe94] in the *Twelf* system [PS99], and also in an experimental implementation of LLF [CP02]. A similar representation with other optimizations is also used in the *Teyjus* implementation [Nad01] of *λProlog*. The technique was originally suggested by an empirical study of higher-order logic programs [MP92]. We did not carry out a systematic study to compare the spine representation with the traditional representation in terms of binary application, since we simultaneously improved other aspects of the system (in particular by the introduction of explicit substitutions and a crude form of compilation). On the other hand, on first-order programs (which are mostly orthogonal to the other implementation improvements), we did indeed observe a constant-factor speed-up roughly consistent with the predicted factor of two.

6.2 Relations 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

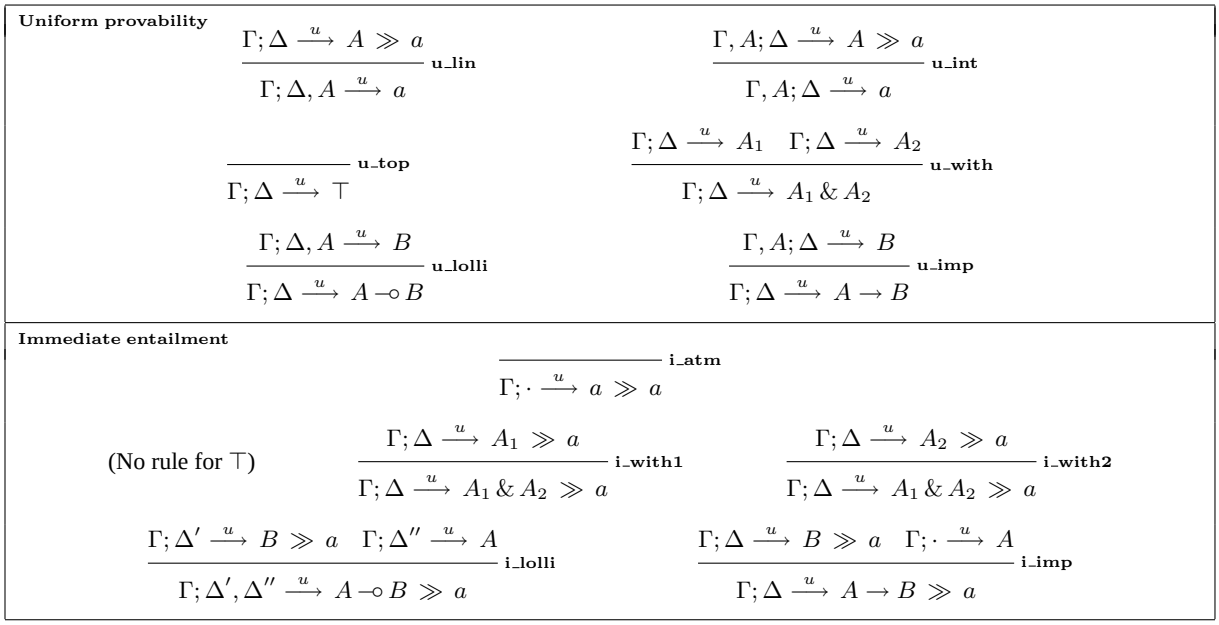


Figure 6.1: Uniform Derivability

ILL$^{\to\multimap\&\top}$ 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 6.1. 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*$^{\to\multimap\&\top}$. 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.

There is a striking correspondence between the proof system displayed in Figure 6.1 and the typing inference system for $S^{\to\multimap\&\top}$ given in Figure 2.1. 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 unrestricted *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.

$$\frac{\Gamma;\Delta \xrightarrow{u} A \quad \Gamma;\Delta \xrightarrow{u} A \gg a}{\Gamma;\Delta, \Delta' \xrightarrow{u} a}$$

Clearly, since uniform derivations are cut-free, the system in Figure 6.1 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 in *ILL*$^{\to\multimap\&\top}$. This sets the basis for a form of the Curry-Howard isomorphism [How80] between normal, well-typed $S^{\to\multimap\&\top}$ terms and valid uniform derivations in *ILL*$^{\to\multimap\&\top}$. Other authors have come to similar conclusions, as described in Section 6.4 below.

The underlying relation between spine representation and focusing is of foundational significance as it generalizes to any abstract logic programming language. Namely, we postulate that for every logic that admits a uniform proof system there is a spine calculus that can be used as a term assignment system for it.

In practice, this correspondence is of little use for logic programming languages that do not return proof-terms as an account of their execution, for example *Prolog*, *Lolli* [HM94] and *λProlog* [Mil89, Mil01]. The situation is different for logic programming languages based on a type theory like *LF* [HHP93] or *LLF* [CP02] rather than a logic. It is currently exploited in the *Twelf* implementation of *LF* to efficiently construct proof-terms directly in spine form during execution. Previous implementations essentially wove in a transformation akin to that in Figure 4.4 at each step, including when visiting portions of the search tree that would later be discarded.

6.3 Interaction with Polymorphism

One of the important features of our spine calculus is that it keeps terms in long normal form, even if substitution introduces new β-redices. This means that types do not need to be passed around in many algorithms that would otherwise require it, such as equality testing or pattern unification. In the presence of polymorphism, η-long forms are still well-defined [DHW93] and exist in many expressive λ-calculi [Gha97], but they are no longer preserved by β-reduction or substitution. This is true even in the absence of linearity. For example,

$$\begin{array}{rl} & \vdash_\Sigma \Lambda\alpha.\, \lambda x\!:\!\alpha.\, x \Uparrow \forall\alpha.\, \alpha \to \alpha, \\ \textit{but} & \not\vdash_\Sigma \lambda x\!:\!a \to a.\, x \Uparrow (a \to a) \to (a \to a), \\ \textit{and} & \not\vdash_\Sigma \lambda x\!:\!\top.\, x \Uparrow \top \to \top. \end{array}$$

In both cases we have to η-expand x in order to obtain a long-normal form.

$$\begin{array}{l} \vdash_\Sigma \lambda x\!:\!a \to a.\, \lambda y\!:\!a.\, x\ y \Uparrow (a \to a) \to (a \to a) \\ \vdash_\Sigma \lambda x\!:\!\top.\, \langle\rangle \Uparrow \top \to \top \end{array}$$

We conjecture that in case of the polymorphic λ-calculus (possibly augmented with linear functions, products, and unit) it is sufficient to store the type α for each root $H \cdot S$ of variable type. During substitution for a type variable α, we locally η-expand the spine S in roots $H \cdot S : \alpha$. In a calculus with explicit substitutions this effect can be achieved when descending into the spine of a root of variable type. The overhead of such an implementation appears minimal when compared to the cost of having to carry types explicitly when traversing a term for the purpose of equality testing or unification in the presence of extensionality. We conjecture that it is possible to generalize this technique further to calculi with variable type constructors and dependencies by storing types with roots whenever the head of the type is a variable.

6.4 Related Work

The uniform derivation system given in Figure 6.1 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 [How80], although a formal spine-like calculus is not defined.

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. Huet's *constructive engine* [Hue89] uses some ideas reminiscent of the spine calculus in an implementation of the Calculus of Construction.

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 with named variables. *LJT* is a 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 on the formula currently in 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 the strong normalization theorem for $\bar{\lambda}$ subsumes the cut-elimination property for this logic.

In [DP98, DP99], Dyckhoff and Pinto use the work of Herbelin to investigate the meta-theory of the sequent calculus and natural deduction presentations of intuitionistic implicational logic. They obtain a simple proof of

a classical equivalence result for them by relating terms in the λ-calculus and in $\bar{\lambda}$ [Her95]. In [DP98], they furthermore propose a simplification of Herbelin's proof of strong cut-elimination for LJT.

Schwichtenberg [Sch99] adopts a similar approach relative 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 commutative 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.

A Curry-Howard-like correspondence between a calculus akin to Herbelin's $\bar{\lambda}$ [Her95] and the calculus of explicit substitution [ACCL91] is further pursued by Dyckhoff and Urban in [DU01]. They show in particular that the studied language is strongly normalizing. Espírito Santo conducts similar investigation in [San00] with an eye on η-long terms.

7 Conclusions

In this paper, we have formalized an alternative presentation of the linear λ-calculus $\lambda^{\rightarrow\multimap\&\top}$ which 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 the studies of term assignment systems for sequent calculi [Her95, DP98, DP99, Sch99, San00, DU01] 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, CP02]. 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 formally.

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 indebted to Helmut Schwichtenberg for the fruitful discussions and for pointing out literature relevant to the topics treated in this paper. Finally, this paper profited from insightful input from Roy Dyckhoff and Randy Pollack at a recent Dagstuhl seminar.

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.

[CP97a] Iliano Cervesato and Frank Pfenning. Linear higher-order pre-unification. Technical Report CMU-CS-97-160, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1997.

[CP97b] Iliano Cervesato and Frank Pfenning. A linear spine calculus. Technical Report CMU-CS-97-125, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, April 1997.

[CP02] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information & Computation*, 179(1):19–75, November 2002.

[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.

[DHW93] Gilles Dowek, Gérard Huet, and Benjamin Werner. On the definition of the eta-long normal form in type systems of the cube. In Herman Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, Nijmegen, The Netherlands, May 1993.

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

[DP98] Roy Dyckhoff and Luis Pinto. Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60:107–118, 1998.

[DP99] Roy Dyckhoff and Luis Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155, 1999.

[DU01] Roy Dyckhoff and Christian Urban. Strong normalization of Herbelin's explicit substitution calculi with substitution propagation. In Pierre Lescanne, editor, *Proceedings of the Fourth Workshop on Explicit Substitutions Theory and Applications — WESTAPP 01*, pages 26–45, Utrecht, the Netherlands, 2001. Logic Group Preprint series No 210, Institute of Philosophy, University of Utrecht, ISBN 90-393-2764-5.

[Gha97] Neil Ghani. Eta-expansions in dependent type theory — the calculus of constructions. In P. de Groote and J.R. Hindley, editors, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 164–180, Nancy, France, April 1997. Springer-Verlag LNCS 1210.

[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.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, 1980*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard, 1979.

[Hue89] Gérard Huet. The constructive engine. In R. Narasimhan, editor, *A Perspective in Theoretical Computer Science*. World Scientific Publishing, 1989. Commemorative Volume for Gift Siromoney.

[IP98] Samin Ishtiaq and David Pym. A relevant analysis of natural deduction. *Journal of Logic and Computation*, 8(6):809–838, 1998.

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

[Mil89] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Proceedings of the International Workshop on Extensions of Logic Programming*, pages 253–281, Tübingen, Germany, 1989. Springer-Verlag LNAI 475.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[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.

[Mil01] Dale Miller. Lambda Prolog: An introduction to the language and its logic. Current draft available from `http://www.cse.psu.edu/~dale/lProlog`, 2001.

[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.

[Nad01] Gopalan Nadathur. The metalanguage lambda prolog and its implementation. In Herbert Kuchen and Kazunori Ueda, editors, *Proceedings of the Fifth International Symposium on Functional and Logic Programming — FLOPS 2001*, pages 1–20, Tokyo, Japan, 2001. Springer Verlag, LNCS 2024.

[Pfe91] Frank Pfenning. Unification and anti-unification in the Calculus of Constructions. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, Amsterdam, The Netherlands, July 1991.

[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.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[San00] José Espírito Santo. Revisiting the correspondence between cut elimination and normalisation. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Proceedings of the 27th International Colloquium on Automata, Languages and Programming — ICALP' 2000*, pages 600–611, Geneva, Switzerland, 2000. Springer Verlag LNCS 1853.

[Sch99] Helmut Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. *Theoretical Computer Science*, 212:247–260, 1999.

A The Coercion Calculus $C^{\rightarrow\multimap\&\top}$

The syntax and the typing and reduction semantics of the coercion calculus $C^{\rightarrow\multimap\&\top}$ have been given in Section 4.1. In this appendix, we define and analyze the translations λC and $C\lambda$ between $\lambda^{\rightarrow\multimap\&\top}$ and $C^{\rightarrow\multimap\&\top}$ in Sections A.2 and A.3, respectively. Before this, we spend Section A.1 studying the main properties of the AC-reduction of the coercion calculus. We conclude in Section A.4 by deducing the main results for $C^{\rightarrow\multimap\&\top}$ reductions from the analogous properties for $\lambda^{\rightarrow\multimap\&\top}$ listed in Section 2.

A.1 Meta-Theory of AC-Reducibility

AC-reduction ($\downarrow\Uparrow R \xrightarrow{C}_{\mathrm{AC}} R$) is closely related to the NIL-reduction of $S^{\rightarrow\multimap\&\top}$ ($(H \cdot S) \cdot \mathrm{NIL} \xrightarrow{S}_{\mathrm{NIL}} H \cdot S$), but does not have an equivalent in $\lambda^{\rightarrow\multimap\&\top}$. This complicates proving the correctness of a direct translation between this language and the spine calculus [CP97b]. Fortunately, we can isolate the main properties of $\xrightarrow{C}_{\mathrm{AC}}$ and therefore achieve simple proofs of these results. We will apply them in Sections A.2 and A.3 to ascertain the correctness of λC and $C\lambda$, and in Section 5 to deduce the analogous properties of NIL-reducibility.

The analysis of the interplay between typing and AC-reduction reveals that this relation enjoys the subject reduction property, as stated by the following lemma.

Lemma A.1 (AC-*reduction preserves typing*)

If $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow\Downarrow A$ *and* $\mathcal{E} :: T \xrightarrow{C}_{\mathrm{AC}} T'$, *then* $\Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow\Downarrow A$.

Proof: By induction on the structure of $\mathcal{E}$ and inversion on $\mathcal{Q}$. ☑

A further property, that we use in Section A.4 of this appendix and also in Section 5 of the body of this paper, is that the use of AC-reduction in the reverse direction, *i.e.*, as an expansion rule, preserves typing too.

Lemma A.2 (AC-*expansion preserves typing*)

If $\mathcal{E} :: T \xrightarrow{C}_{\mathrm{AC}} T'$ *and* $\mathcal{Q}' :: \Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow\Downarrow A$, *then* $\Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow\Downarrow A$.

Proof: By induction on the structure of $\mathcal{E}$ and inversion on $\mathcal{Q}'$. ☑

We now concentrate on the properties of $C^{\rightarrow\multimap\&\top}$ and $\xrightarrow{C}_{\mathrm{AC}}$ as a rewriting system. An application of rule $\mathbf{Cr\_AC}$ reduces an AC-redex by eliminating adjacent $\downarrow$and $\Uparrow$coercions. Therefore, only as many AC-reductions can be chained starting from a given term as the number of AC-redices present in it. This implies that any sequence of AC-reductions is terminating in $C^{\rightarrow\multimap\&\top}$.

Lemma A.3 (*Strong* AC-*normalization*)

Every maximal sequence of AC-*reductions starting at a term* T *is finite.*

Proof: A formal proof goes by induction on the structure of T. ☑

This property entails also that, given a term T, there is only a finite number of terms T^* such that $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$ is derivable. Therefore checking whether $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$ has a derivation is decidable.

If the AC-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 pre-canonical and pre-atomic terms.

Lemma A.4 (*Local confluence*)

If $\mathcal{E}' :: T \xrightarrow{C}_{\mathrm{AC}} T'$ *and* $\mathcal{E}'' :: T \xrightarrow{C}_{\mathrm{AC}} T''$, *then either* $T' = T''$ *or there is a term* T^* *such that* $T' \xrightarrow{C}_{\mathrm{AC}} T^*$ *and* $T'' \xrightarrow{C}_{\mathrm{AC}} T^*$.

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

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

Corollary A.5 (*Confluence*)

If $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T'$ *and* $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T''$, *then there is a term* T^* *such that* $T' \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$ *and* $T'' \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$. □

We say that a term is in AC-*normal form* if it does not contain any AC-redex. Since $\xrightarrow{C}_{\text{AC}}$ eliminates an AC-redex, an exhaustive application to a pre-canonical term Q or pre-atomic term R yields an AC-normal term. A combination of the results above ensures that an AC-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 A.6 (*Uniqueness of* AC-*normal forms*)

For every term T there is a unique AC-*normal term T^* such that $T \xrightarrow{C}{}^*_{\text{AC}} T^*$.*

Proof:

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

Assume that there are two such term, T_1^* and T_2^* say. Then by confluence, they must have a common AC-reduct T^{**}. However, since neither T_1^* nor T_2^* admit AC-reductions, it must be the case that $T_1^* = T_2^* = T^{**}$. ☑

We denote *the* AC-normal form of a pre-canonical term Q (pre-atomic term R) as $\text{NF}_{\text{AC}}(Q)$ ($\text{NF}_{\text{AC}}(R)$, respectively). Furthermore, we write $C_0^{\rightarrow\multimap\&\top}$ for the sublanguage of $C^{\rightarrow\multimap\&\top}$ that consists only of AC-normal terms.

In Section A.4, we will take advantage of the following technical result that states that substitution preserves AC-reducibility.

Lemma A.7 (*Substitution*)

If $\mathcal{E} :: T \xrightarrow{C}{}^*_{\text{AC}} T'$ and $\mathcal{E}_R :: R \xrightarrow{C}{}^*_{\text{AC}} R'$, then $[R/x]T \xrightarrow{C}{}^*_{\text{AC}} [R'/x]T'$.

Proof: By induction on the structure of $\mathcal{E}$. ☑

Observe that the substituted term ought to be pre-atomic, while the term on which the substitution is performed can be either pre-canonical or pre-atomic.

A.2 λC: A Translation from $\lambda^{\rightarrow\multimap\&\top}$ to $C^{\rightarrow\multimap\&\top}$

The translation from $\lambda^{\rightarrow\multimap\&\top}$ to $C^{\rightarrow\multimap\&\top}$, abbreviated λC, maps $\lambda^{\rightarrow\multimap\&\top}$ terms to objects in $C^{\rightarrow\multimap\&\top}$. λC is specified by means of the following judgments:

$M \xrightarrow{\lambda C} Q$	*M translates to pre-canonical term Q*
$N \xrightarrow{\lambda C} R$	*N translates to pre-atomic term R*

The rules defining them are displayed in Figure A.1. The side conditions in rules $\lambda\mathbf{C\_atm}$ and $\lambda\mathbf{C\_redex}$ specify the admissible structure of their first argument (M or N); 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 λC, the structure of the first argument determines uniquely which rule can be used in the translation process. We write $\mathcal{LC}$, possibly annotated, for derivations of either judgments.

We can immediately prove the faithfulness of this translation with respect to typing. This result expresses the adequacy of the system in Figure 4.1 as an emulation of the typing semantics of $\lambda^{\rightarrow\multimap\&\top}$. We will take advantage of this fact below.

Theorem A.8 (*Soundness of λC for typing*)

If $\mathcal{C} :: \Gamma; \Delta \vdash_\Sigma M \Uparrow\Downarrow A$ and $\mathcal{LC} :: M \xrightarrow{\lambda C} T$, then $\Gamma; \Delta \vdash^C_\Sigma T \Uparrow\Downarrow A$;

Proof:

This proof proceeds by simultaneous induction on the structure of $\mathcal{C}$. Most cases are resolved by simple invocations of the induction hypothesis. However, rules $\mathrm{l}\lambda\mathbf{\_atm}$ and $\mathrm{l}\lambda\mathbf{\_redex}$ require some care in order to satisfy the side conditions in rules $\lambda\mathbf{C\_atm}$ and $\lambda\mathbf{C\_redex}$, respectively. We will focus our analysis on these proof patterns.

Pre−canonical terms

$$\frac{N \xrightarrow{\lambda C} R}{N \xrightarrow{\lambda C} \Uparrow R}\ \lambda\mathbf{C\_atm} \quad (\text{for } N = c,\ x,\ \text{FST}\ N',\ \text{SND}\ N',\ N'\hat{}M,\ N'\ M)$$

$$\frac{}{\langle\rangle \xrightarrow{\lambda C} \langle\rangle}\ \lambda\mathbf{C\_unit} \qquad \frac{M_1 \xrightarrow{\lambda C} Q_1 \quad M_2 \xrightarrow{\lambda C} Q_2}{\langle M_1, M_2\rangle \xrightarrow{\lambda C} \langle Q_1, Q_2\rangle}\ \lambda\mathbf{C\_pair}$$

$$\frac{M \xrightarrow{\lambda C} Q}{\hat{\lambda}x{:}A.\,M \xrightarrow{\lambda C} \hat{\lambda}x{:}A.\,Q}\ \lambda\mathbf{C\_llam} \qquad \frac{M \xrightarrow{\lambda C} Q}{\lambda x{:}A.\,M \xrightarrow{\lambda C} \lambda x{:}A.\,Q}\ \lambda\mathbf{C\_ilam}$$

Pre−atomic terms

$$\frac{M \xrightarrow{\lambda C} Q}{M \xrightarrow{\lambda C} \downarrow Q}\ \lambda\mathbf{C\_redex} \quad (\text{for } M = \langle\rangle, \langle M', M''\rangle,\ \hat{\lambda}x{:}A.\,M',\ \lambda x{:}A.\,M')$$

$$\frac{}{c \xrightarrow{\lambda C} c}\ \lambda\mathbf{C\_con} \qquad \frac{}{x \xrightarrow{\lambda C} x}\ \lambda\mathbf{C\_var}$$

$$\frac{N \xrightarrow{\lambda C} R}{\text{FST}\ N \xrightarrow{\lambda C} \text{FST}\ R}\ \lambda\mathbf{C\_fst} \qquad \frac{N \xrightarrow{\lambda C} R}{\text{SND}\ N \xrightarrow{\lambda C} \text{SND}\ R}\ \lambda\mathbf{C\_snd}$$

$$\frac{N \xrightarrow{\lambda C} R \quad M \xrightarrow{\lambda C} Q}{N\hat{}M \xrightarrow{\lambda C} R\hat{}Q}\ \lambda\mathbf{C\_lapp} \qquad \frac{N \xrightarrow{\lambda C} R \quad M \xrightarrow{\lambda C} Q}{N\ M \xrightarrow{\lambda C} R\ Q}\ \lambda\mathbf{C\_iapp}$$

Figure A.1: Translation of $\lambda^{\rightarrow -\circ \& \top}$ into $C^{\rightarrow -\circ \& \top}$

$$\mathbf{l\lambda\_atm}: \quad \mathcal{C} = \frac{\begin{array}{c}\mathcal{A}'\\ \Gamma;\Delta \vdash_\Sigma M \downarrow a\end{array}}{\Gamma;\Delta \vdash_\Sigma M \Uparrow a}\ \mathbf{l\lambda\_atm}$$

where $A = a$.

$M = c,\ x,\ \text{FST}\ M',\ \text{SND}\ M',\ M'\hat{}M''$ or $M'\ M''$	by the Surjectivity Lemma 2.1 on $\mathcal{C}$,
$T = \Uparrow R$	and
$\mathcal{LC}' :: M \xrightarrow{\lambda C} R$	by inversion on rule $\lambda\mathbf{C\_atm}$,
$\mathcal{R}' :: \Gamma;\Delta \vdash^C_\Sigma R \downarrow a$	by induction hypothesis on $\mathcal{A}'$ and $\mathcal{LC}'$,
$\mathcal{Q}' :: \Gamma;\Delta \vdash^C_\Sigma \Uparrow R \Uparrow a$	by rule $\lambda\mathbf{C\_atm}$ on $\mathcal{R}'$.

$$\mathbf{l\lambda\_redex}: \quad \mathcal{C} = \frac{\begin{array}{c}\mathcal{C}'\\ \Gamma;\Delta \vdash_\Sigma M \Uparrow A\end{array}}{\Gamma;\Delta \vdash_\Sigma M \downarrow A}\ \mathbf{l\lambda\_redex}$$

This case is more delicate than the previous situation because we have no tool such as the Surjectivity Lemma to invoke the induction hypothesis. We must instead distinguish subcases on the structure of the type A. More precisely, the major discriminant is whether A is an atomic or a composite type:

Subcase $A \neq a$:

$M = \langle\rangle, \langle M', M''\rangle,\ \hat{\lambda}x{:}A.\,M'$ or $\lambda x{:}A.\,M'$	by the Surjectivity Lemma 2.1 on $\mathcal{C}'$,
$T = \downarrow Q$	and
$\mathcal{LC}' :: M \xrightarrow{\lambda C} Q$	by inversion on rule $\lambda\mathbf{C\_redex}$,
$\mathcal{Q}' :: \Gamma;\Delta \vdash^C_\Sigma Q \Uparrow A$	by induction hypothesis on $\mathcal{C}'$ and $\mathcal{LC}'$,
$\mathcal{R}' :: \Gamma;\Delta \vdash^C_\Sigma \downarrow Q \downarrow A$	by rule $\lambda\mathbf{C\_redex}$ on $\mathcal{Q}'$.

Subcase $A = a$:

$\mathcal{A}' :: \Gamma; \Delta \vdash_\Sigma M \downarrow a$ by inversion on rule $\lambda\mathbf{C}\_\mathbf{atm}$ on $\mathcal{C}'$,

$\mathcal{R}' :: \Gamma; \Delta \vdash^C_\Sigma R \downarrow a$ by induction hypothesis on $\mathcal{A}'$ and $\mathcal{LC}$.

This last subcase cancels occurrences of rule $\mathbf{l\lambda\_redex}$ immediately followed by $\mathbf{l\lambda\_atm}$ in a typing derivation. In this way, it essentially removes an implicit AC-redex from the original derivation. ☑

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

We will obtain an indirect proof of the completeness of λC with respect to typing in Section A.3. As a preparatory step, we get some insight in the manner λC operates.

We first show that λC is a function, *i.e.*, that every term has a unique translation, as established by the following lemma.

Lemma A.9 (*Functionality of λC*)

i. For every M in $\lambda^{\to\multimap\&\top}$, there is a unique pre-canonical term Q in $C^{\to\multimap\&\top}$ such that $M \xrightarrow{\lambda C} Q$.

ii. For every M in $\lambda^{\to\multimap\&\top}$, there is a unique pre-atomic term R in $C^{\to\multimap\&\top}$ such that $M \xrightarrow{\lambda C} R$.

Proof: This simple proof proceeds by induction on the structure of M. ☑

λC translates every term in $\lambda^{\to\multimap\&\top}$ to an object in AC-normal form. Therefore, the range of this function is the set of terms in $C_0^{\to\multimap\&\top}$:

Lemma A.10 (*Range of λC*)

If $\mathcal{LC} :: M \xrightarrow{\lambda C} T$, then T is in AC*-normal form.*

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

We have just seen that λC is sound with respect to the typing semantics of $\lambda^{\to\multimap\&\top}$ and $C^{\to\multimap\&\top}$. We dedicate the remainder of this section to proving that it preserves also reductions. This task is complicated by the fact that β-reductions in $\lambda^{\to\multimap\&\top}$ do not correspond to β-reductions in $C^{\to\multimap\&\top}$, but in general to β-reductions followed by zero or more AC-reductions.

Consider for example the simple $\lambda^{\to\multimap\&\top}$ redex $(\hat{\lambda}x : a.\, f \hat{} x) \hat{} c$ which reduces in one (β-)step to $f \hat{} c$. Its pre-atomic translation to $C^{\to\multimap\&\top}$ according to λC is

$$(\downarrow(\hat{\lambda}x{:}a.\, \Uparrow(f \hat{} (\Uparrow x)))) \hat{} (\Uparrow c)$$

(Its pre-canonical translation wraps this expression with one more occurrence of $\Uparrow$). One β-reduction step yields

$$\downarrow(\Uparrow(f \hat{} (\Uparrow(\downarrow(\Uparrow c)))))$$

It then takes two AC-reductions to simplify this term into $f \hat{} (\Uparrow c)$, which is the pre-atomic image of $f \hat{} c$ according to λC.

Rules **lr_beta_lin** and **lr_beta_int** generate their reduct means of a meta-level substitution. The corresponding reduction in $C^{\to\multimap\&\top}$ operate in a similar way. Therefore, we need to show that λC commutes reasonably well with substitution. This is achieved in the next lemma.

Lemma A.11 (*Substitution in λC*)

Assume that $\mathcal{C}$ is a derivation of either $\Gamma; \Delta, x : A \vdash_\Sigma M \Uparrow\Downarrow B$ or $\Gamma, x : A; \Delta \vdash_\Sigma M \Uparrow\Downarrow B$, and moreover $\mathcal{A} :: \Gamma; \Delta' \vdash_\Sigma N \downarrow A$.

If $\mathcal{LC} :: M \xrightarrow{\lambda C} T$ and $\mathcal{LC}_N :: N \xrightarrow{\lambda C} R$, then $[N/x]M \xrightarrow{\lambda C} [R/x]T$.

Proof:

The proof proceeds by induction on the structure of $\mathcal{LC}$. All cases are quite simple except for the situation where M is precisely the variable x (subcase of rule $\lambda\mathbf{C\_atm}$).

$\lambda\mathbf{C\_var}$: $\quad \mathcal{LC} = \dfrac{}{x \xrightarrow{\lambda C} x}\ \lambda\mathbf{C\_var}$

where $M = x$ and $T = x$.

$\mathcal{LC}_N :: [N/x]x \xrightarrow{\lambda C} [R/x]x$ by definition of substitution

$\lambda\mathbf{C\_atm}$: $\quad \mathcal{LC} = \dfrac{\mathcal{LC}'}{x \xrightarrow{\lambda C} \Uparrow x}\ \lambda\mathbf{C\_atm}$

where $M = x$ and $T = \Uparrow x$.

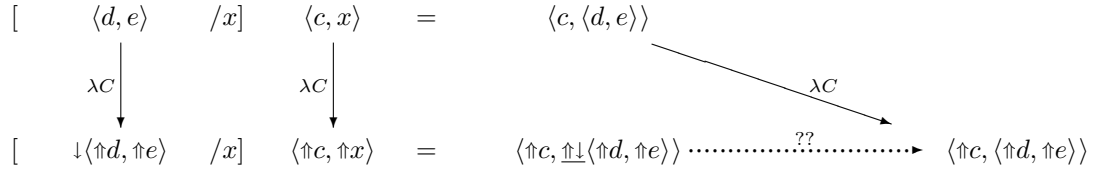
$A = a$ by inversion on rule $\mathbf{l\lambda\_atm}$ for $\mathcal{C}$,

$N = c,\ y,\ \text{FST}\ N',\ \text{SND}\ N',\ N'\hat{}N''$ or $N'\ N''$ by a simple induction on $\mathcal{A}$,

$\mathcal{LC}' :: N \xrightarrow{\lambda C} \Uparrow R$ by rule $\lambda\mathbf{C\_atm}$ on $\mathcal{LC}_N$,

$[N/x]x \xrightarrow{\lambda C} [R/x]\Uparrow x$ by definition of substitution. ☑

The presence of typing derivations in the above result are needed to ensure that the terms M and N are in η-long form. When this property does not hold, our set of reductions in the coercion calculus are not sufficient to support our translation. An example shall clarify this point: consider the terms $M = \langle c, x\rangle$ and $N = \langle d, e\rangle$. Then, clearly $[N/x]M = \langle c, \langle d, e\rangle\rangle$. Now λC (pre-atomically) translates M and N to $Q = \langle \Uparrow c, \Uparrow x\rangle$ and $R = \downarrow\langle \Uparrow d, \Uparrow e\rangle$, respectively. Then, by performing the substitution at the level of $C^{\rightarrow\multimap\&\top}$, we obtain $[R/x]Q = \langle \Uparrow c, \Uparrow\downarrow\langle \Uparrow d, \Uparrow e\rangle\rangle$, while $[N/x]M \xrightarrow{\lambda C} \langle \Uparrow c, \langle \Uparrow d, \Uparrow e\rangle\rangle$. Notice however that $[R/x]Q$ embeds the sequence of operators $\Uparrow\downarrow$_, which is not handled by any of the reduction rules of $C^{\rightarrow\multimap\&\top}$ (in particular not by $\mathbf{Sr\_AC}$, which expects these constructs to occur in reverse order). Schematically,



Anomalies of this type do not arise when operating on η-long terms only since a variable x of composite type would not occur immediately prefixed by the $\Uparrow$_ coercion.

In a functional redex, both the substituting and the substituted terms are pre-canonical. The following corollary adapts the above lemma to handle this situation. Observe that it may introduce AC-reductions

Corollary A.12 (*Substitution in λC*)

Let $\mathcal{A} :: \Gamma; \Delta' \vdash_\Sigma N \downarrow A$.

1. *Assume there is a derivation of either $\Gamma; \Delta, x : A \vdash_\Sigma M \Uparrow B$ or $\Gamma, x : A; \Delta \vdash_\Sigma M \Uparrow B$. Then, if $\mathcal{LC} :: M \xrightarrow{\lambda C} Q'$ and $\mathcal{LC}_N :: N \xrightarrow{\lambda C} Q$, then $[N/x]M \xrightarrow{\lambda C} Q^*$ where $[\downarrow Q/x]Q' \xrightarrow{C}{}^*_{\text{AC}}\ Q^*$.*
2. *Assume there is a derivation of either $\Gamma; \Delta, x : A \vdash_\Sigma M \downarrow B$ or $\Gamma, x : A; \Delta \vdash_\Sigma M \downarrow B$. Then, if $\mathcal{LC} :: M \xrightarrow{\lambda C} R$ and $\mathcal{LC}_N :: N \xrightarrow{\lambda C} Q$, then $[N/x]M \xrightarrow{\lambda C} Q^*$ where $\downarrow[\downarrow Q/x]R \xrightarrow{C}{}^*_{\text{AC}}\ Q^*$.*

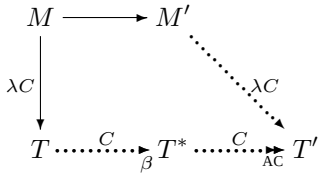
Proof:

1. We distinguish cases on the basis of the last proof rule applied in $\mathcal{LC}_N$.

- If this derivation ends in rule $\lambda\mathbf{C\_atm}$, then $Q = \Uparrow R$ and there is a derivation of $N \xrightarrow{\lambda C} R$. By the previous lemma, $[N/x]M \xrightarrow{\lambda C} [R/x]Q'$. Then a simple induction shows that $[\downarrow(\Uparrow R)/x]Q' \xrightarrow{C}{}^{*}_{\mathrm{AC}} [R/x]Q'$.
- Otherwise, by the Surjectivity Lemma 2.1, we can extend $\mathcal{LC}_N$ with an application of rule $\lambda\mathbf{C\_redex}$. The desired result then follows by a direct application of the above Substitution Lemma.

2. We proceed similarly to the first part of this corollary, but by cases on the structure of $\mathcal{LC}$. ☑

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



Theorem A.13 (*Soundness of λC for reducibility*)

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

If $\mathcal{D} :: M \longrightarrow M'$ *and* $\mathcal{LC} :: M \xrightarrow{\lambda C} T$, *then there are terms* T^* *and* T' *such that* $T \xrightarrow{C}_\beta T^*$, $T^* \xrightarrow{C}{}^{*}_{\mathrm{AC}} T'$ *and* $M' \xrightarrow{\lambda C} T'$.

Proof:

The proof proceeds by induction on the structure of $\mathcal{D}$. 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**). We develop in full the cases where the last rule applied in $\mathcal{D}$ is **lr_beta_lin**.

$$\mathbf{lr\_beta\_lin}: \quad \mathcal{D} = \frac{}{(\hat{\lambda}x\!:\!A.\, M_1)\,\hat{}\,M_2 \longrightarrow [M_2/x]M_1}\ \mathbf{lr\_beta\_lin}$$

where $M = (\hat{\lambda}x\!:\!A.\, M_1)\,\hat{}\,M_2$ and $N = [M_2/x]M_1$.

$\mathcal{C}_1 :: \Gamma; \Delta_1, x : A \vdash_\Sigma M_1 \Uparrow B$ and

$\mathcal{C}_2 :: \Gamma; \Delta_2 \vdash_\Sigma M_2 \Uparrow A$ by inversion on $\mathcal{C}$,

By inversion on $\mathcal{LC}$, there are terms Q_1 and Q_2, and derivations $\mathcal{LC}_1$ and $\mathcal{LC}_2$ that allow expanding $\mathcal{LC}$ as follows:

$$\mathcal{LC} = \dfrac{\dfrac{\dfrac{\mathcal{LC}_1}{M_1 \xrightarrow{\lambda C} Q_1}\,\lambda\mathbf{C\_llam}}{\hat{\lambda}x\!:\!A.\, M_1 \xrightarrow{\lambda C} \hat{\lambda}x\!:\!A.\, Q_1}\,\lambda\mathbf{C\_redex} \qquad \dfrac{\mathcal{LC}_2}{M_2 \xrightarrow{\lambda C} Q_2}}{(\hat{\lambda}x\!:\!A.\, M_1)\,\hat{}\,M_2 \xrightarrow{\lambda C} (\downarrow(\hat{\lambda}x\!:\!A.\, Q_1))\,\hat{}\,Q_2}\,\lambda\mathbf{C\_lapp}$$

where $T = (\downarrow(\hat{\lambda}x\!:\!A.\, Q_1))\,\hat{}\,Q_2$.

$\mathcal{E}_\beta :: (\downarrow(\hat{\lambda}x\!:\!A.\, Q_1))\,\hat{}\,Q_2 \xrightarrow{C}_\beta \downarrow[\downarrow Q_2/x]Q_1$ by rule $\mathbf{Cr\_beta\_lin}$,

$\mathcal{E}_{\mathrm{AC}} :: \downarrow[\downarrow Q_2/x]Q_1 \xrightarrow{C}{}^{*}_{\mathrm{AC}} Q'$ and

$\mathcal{LC}' :: [M_2/x]M_1 \xrightarrow{\lambda C} Q'$ by the Substitution Corollary A.12 on $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{LC}_1$, and $\mathcal{LC}_2$. ☑

This result can be lifted to the reflexive and transitive closures of the mentioned reduction relations.

The notion of soundness we adopted relative 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 λC with respect to the reduction semantics of our calculi in Section A.3, when considering the inverse of our translation.

A.3 $C\lambda$: A Translation from $C^{\rightarrow\multimap\&\top}$ to $\lambda^{\rightarrow\multimap\&\top}$

In this section, we consider the problem of translating terms from $C^{\rightarrow\multimap\&\top}$ back to $\lambda^{\rightarrow\multimap\&\top}$. λC cannot be used for this purpose since its co-domain is $C_0^{\rightarrow\multimap\&\top}$, the subset of $C^{\rightarrow\multimap\&\top}$ consisting only of AC-normal forms.

The approach we take is instead to define an independent translation, $C\lambda$, that maps entities in $C^{\rightarrow\multimap\&\top}$ to terms in $\lambda^{\rightarrow\multimap\&\top}$. We will prove later that it is the inverse of λC in a sense to be made precise. $C\lambda$ is specified by means of the judgments

$$\begin{array}{ll} Q \xrightarrow{\mathrm{C}\lambda} M & \textit{Q translates to M} \\ R \xrightarrow{\mathrm{C}\lambda} N & \textit{R translates to N} \end{array}$$

that simply erase the coercions $\Uparrow$ and $\downarrow$ from a term. We omit the obvious inference rules defining this translation We will write $\mathcal{CL}$, variously annotated, for derivations of either of these judgments.

The faithfulness of $C\lambda$ 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 surjectivity.

Theorem A.14 (*Soundness of $C\lambda$ for typing*)

If $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow\Downarrow A$ *and* $\mathcal{CL} :: T \xrightarrow{\mathrm{C}\lambda} M$, *then* $\Gamma; \Delta \vdash_{\Sigma} M \Uparrow\Downarrow A$.

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

Prior to showing that $C\lambda$ preserves reductions, we will prove that $C\lambda$ is the inverse of λC. 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 λC and $C\lambda$ with respect to typing and reduction.

We begin our endeavor by proving that $C\lambda$ is actually a function from $C^{\rightarrow\multimap\&\top}$ to $\lambda^{\rightarrow\multimap\&\top}$.

Lemma A.15 (*Functionality of $C\lambda$*)

i. For every pre-canonical $C^{\rightarrow\multimap\&\top}$ term Q, there is a unique $\lambda^{\rightarrow\multimap\&\top}$ term M such that $Q \xrightarrow{\mathrm{C}\lambda} M$.

ii. For every pre-atomic $C^{\rightarrow\multimap\&\top}$ term R, there is a unique $\lambda^{\rightarrow\multimap\&\top}$ term M such that $R \xrightarrow{\mathrm{C}\lambda} M$.

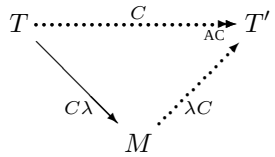
Proof: By induction on the structure of Q and R. ☑

We wish $C\lambda$ to be the inverse of λC. This property does not hold in its full strength. The problem is that these two functions have different domains and ranges. Indeed, λC produces elements in $C_0^{\rightarrow\multimap\&\top}$, a strict subset or $C^{\rightarrow\multimap\&\top}$. On the other hand, $C\lambda$ accepts arbitrary terms in $C^{\rightarrow\multimap\&\top}$. We bridge these differences in the lemma below by relying on AC-reduction.

Lemma A.16 (*Right invertibility*)

Assume that $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow\Downarrow A$.

If $\mathcal{CL} :: T \xrightarrow{\mathrm{C}\lambda} M$, *then* $M \xrightarrow{\lambda C} T'$ *where* $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T'$.



Proof:

The proof proceeds by induction on the structure of $\mathcal{LC}$ and inversion on $\mathcal{Q}$. We rely on the same reasoning pattern already used in the proofs of the substitution lemma for λC (Lemma A.11) and in the soundness theorem A.13. The most complex cases involve rules $\mathbf{C\lambda\_redex}$ and $\mathbf{C\lambda\_atm}$. ☑

Again, the typing assumption in the statement of this lemma is aimed at enforcing surjectivity. Consider for example the $C^{\rightarrow\multimap\&\top}$ term $\Uparrow\downarrow\langle\Uparrow c, \Uparrow d\rangle$, which is not in η-long form. $C\lambda$ would translate it to $\langle c, d\rangle$, which would in turn be mapped to the (η-long) term $\langle\Uparrow c, \Uparrow d\rangle$. However, none of our reduction rules can bridge the gap between the two $C^{\rightarrow\multimap\&\top}$ expressions. No such problems arise when working exclusively with η-long terms.

The reverse of this property holds in a much stronger sense: translating a $\lambda^{\rightarrow\multimap\&\top}$ term to $C^{\rightarrow\multimap\&\top}$ and then back yields the very same original term, without any need for a typing derivation. We have the following lemma.

Lemma A.17 (*Left invertibility*)

If $\mathcal{LC} :: M \xrightarrow{\lambda C} T$, *then* $T \xrightarrow{C\lambda} M$.

Proof: By induction on the structure of $\mathcal{LC}$. ☑

The left-invertibility lemma states that composing $C\lambda$ with λC transforms a $\lambda^{\to\multimap\&\top}$ term to itself; therefore it corresponds to the identity function on $\lambda^{\to\multimap\&\top}$. On the other hand, the right-invertibility lemma A.16 states that $C\lambda$ is the right inverse of λC on well-typed $C_0^{\to\multimap\&\top}$ 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 $\lambda^{\to\multimap\&\top}$ and $C_0^{\to\multimap\&\top}$.

Corollary A.18 (*Bijectivity*)

λC and $C\lambda$ are bijections between the set of well-typed $\lambda^{\to\multimap\&\top}$ terms and the set of well-typed $C_0^{\to\multimap\&\top}$ terms. Moreover, they are each other's inverse.

Proof:

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

By Lemmas A.9, A.10 and Theorem A.13, we know that λC is a function from the well-typed portion of $\lambda^{\to\multimap\&\top}$ to the well-typed subset of $C_0^{\to\multimap\&\top}$. By the functionality lemma A.15, $C\lambda$ maps $C^{\to\multimap\&\top}$ terms to $C^{\to\multimap\&\top}$ objects; in particular, it associates well-typed AC-normal $C^{\to\multimap\&\top}$ terms to well-typed $\lambda^{\to\multimap\&\top}$ terms. Moreover, since terms that are already AC-normal cannot be further AC-reduced, the right-invertibility Lemma states that $C\lambda$ is the left inverse of λC on well-typed $C_0^{\to\multimap\&\top}$ terms. Finally, by the left-invertibility lemma, λC is the left inverse of $C\lambda$ on $\lambda^{\to\multimap\&\top}$, and in particular on its well-typed fragment.

On the basis of these hypotheses, the previous algebraic observation allows us to conclude that λC and $C\lambda$ are indeed bijections between well-typed objects in $\lambda^{\to\multimap\&\top}$ and well-typed terms in $C_0^{\to\multimap\&\top}$, and that they are one another's inverse. ☑

This property opens the door to easy proofs of the completeness direction of every soundness theorem achieved so far. We first consider the completeness of λC with respect to typing.

Corollary A.19 (*Completeness of λC for typing*)

If $M \xrightarrow{\lambda C} T$ *and* $\Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$, *then* $\Gamma; \Delta \vdash_{\Sigma} M \Uparrow A$.

Proof:

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

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

Corollary A.20 (*Completeness of $C\lambda$ for typing*)

If $T \xrightarrow{C\lambda} M$ *and* $\Gamma; \Delta \vdash_{\Sigma} M \Uparrow A$, *then* $\Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$.

Proof:

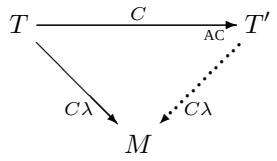
By the right-invertibility lemma, $M \xrightarrow{\lambda C} T'$ where $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T'$. By the soundness of λC for typing, we obtain that $\Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow A$. Finally, since AC-expansion preserves typing (Lemma A.2), we get $\Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$. ☑

We will now analyze the interaction between $C\lambda$ as a translation from $C^{\to\multimap\&\top}$ and $\lambda^{\to\multimap\&\top}$, and the notion of reduction inherent to these two languages. The main results of our investigation will be that $C\lambda$ preserves β-reductions, but identifies AC-convertible terms. We will also take advantage of the fact that this translation is the inverse of λC to prove the completeness counterpart of these statements.

It will be convenient to start by getting a deeper understanding of how AC-reducibility relates to $C\lambda$. Consider the equivalence relation $\overset{C}{\equiv}_{\mathrm{AC}}$ induced by the AC-reduction congruence $\xrightarrow{C}_{\mathrm{AC}}$. Its equivalence classes consist of all

the terms of $C^{\to\multimap\&\top}$ that AC-reduce to the same AC-normal form. $C\lambda$ uniformly maps every object in such an equivalence class to the same $\lambda^{\to\multimap\&\top}$ term. In order to prove this fact, we first show that AC-reducing a term does not affect its translation.

Lemma A.21 (*Invariance of $C\lambda$ under* AC*-reduction*)
If $\mathcal{Q} :: T \xrightarrow{C}_{\mathrm{AC}} T'$ *and* $\mathcal{CL} :: T \xrightarrow{\mathrm{C}\lambda} M$, *then* $T' \xrightarrow{\mathrm{C}\lambda} M$.

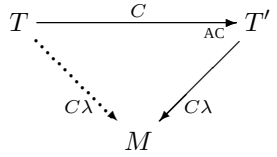


Proof: By induction on the structure of $\mathcal{Q}$. ☑

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

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

Lemma A.22 (*Invariance of $C\lambda$ under* AC*-expansion*)
If $\mathcal{Q} :: T \xrightarrow{C}_{\mathrm{AC}} T'$ *and* $\mathcal{CL}' :: T' \xrightarrow{\mathrm{C}\lambda} M$, *then* $T \xrightarrow{\mathrm{C}\lambda} M$.

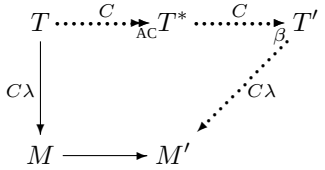


Proof: By induction on the structure of $\mathcal{Q}$. ☑

Strong AC-normalization (Lemma A.6) enables to easily shift these properties to the reflexive and transitive closure of $\xrightarrow{C}_{\mathrm{AC}}$, and to the corresponding equivalence relation.

The AC-invariance properties we just achieved together with the discovery above that λC and $C\lambda$ 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 A.23 (*Completeness of $C\lambda$ for reduction*)
Assume that $\Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow\Downarrow A$.
If $T \xrightarrow{\mathrm{C}\lambda} M$ *and* $M \longrightarrow M'$, *then there is an* AC*-normal term* T^* *and a term* T' *such that* $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$, $T^* \xrightarrow{C}_{\beta} T'$ *and* $T' \xrightarrow{\mathrm{C}\lambda} N$.



Proof:

By the left-invertibility lemma A.17, there is an AC-normal term T^* such that $M \xrightarrow{\lambda C} T^*$ and $T \xrightarrow{C}{}^{*}_{\mathrm{AC}} T^*$ are derivable. By the soundness of λC with respect to reduction (Theorem A.13), there are terms T' and T'' such that

$$T^* \xrightarrow{C}_{\beta} T' \xrightarrow{C}{}^{*}_{\mathrm{AC}} T'' \qquad \text{and} \qquad M' \xrightarrow{\lambda C} T''.$$

By the right-invertibility lemma A.16, we have that $T'' \xrightarrow{\mathrm{C}\lambda} M'$. Finally, by the functionality and invariance of $C\lambda$ under AC-reductions (Lemmas A.15 and A.21), we obtain that $T' \xrightarrow{\mathrm{C}\lambda} M'$. ☑

We conclude this section by showing that $C\lambda$ is sound with respect to the reduction semantics of $C^{\to\multimap\&\top}$. The above invariance lemmas capture this property in the case of AC-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 λC. There are however three important differences. First, the proofs are much simpler in the present case. Second, the statements below do not mention any typing information. Third, AC-reductions do not appear in these statements. This overall simplification derives from the fact that, because of the presence of AC-reduction, $C^{\to\multimap\&\top}$ has more structure than $\lambda^{\to\multimap\&\top}$. Therefore, $C\lambda$ can simply forget about the extra structure of the $C^{\to\multimap\&\top}$ terms it acts upon.

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

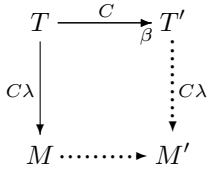
Lemma A.24 (*Substitution in $C\lambda$*)
If $\mathcal{CL} :: T \xrightarrow{\mathrm{C}\lambda} M$ *and* $\mathcal{CL}_R :: R \xrightarrow{\mathrm{C}\lambda} N$, *then* $[R/x]T \xrightarrow{\mathrm{C}\lambda} [N/x]M$.

Proof: By induction on the structure of $\mathcal{CL}$. ☑

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

Theorem A.25 (*Soundness of $C\lambda$ for β-reducibility*)

If $\mathcal{D} :: T \xrightarrow{C}_{\beta} T'$ and $\mathcal{CL} :: T \xrightarrow{C\lambda} M$, then there is a term M' such that $M \longrightarrow M'$ and $T' \xrightarrow{C\lambda} M'$.

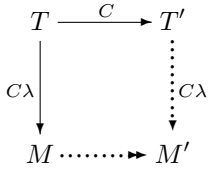


Proof: By induction on the structure of $\mathcal{R}$. ☑

We can summarize the previous theorem, stating the soundness of $C\lambda$ for β-reducibility, and the invariance lemma A.21, expressing the soundness of $C\lambda$ for AC-reducibility in a single statement mentioning the generic notion of reduction of $C^{\rightarrow\multimap\&\top}$.

Corollary A.26 (*Soundness of $C\lambda$ for reducibility*)

If $T \xrightarrow{C} T'$ and $T \xrightarrow{C\lambda} M$, then there is a term M' such that $M \longrightarrow^* M'$ and $T' \xrightarrow{C\lambda} M'$.



Proof:

Depending on whether $\xrightarrow{C}$ is $\xrightarrow{C}_{\text{AC}}$ or $\xrightarrow{C}_{\beta}$, this statement corresponds to Lemma A.21 or to theorem A.25, respectively. In the former case, $M' = M$ and $\longrightarrow^*$ is instantiated to the identity. ☑

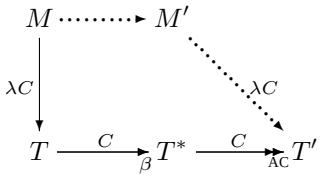
Clearly, the above result holds also relatively to the reflexive and transitive closure of $\xrightarrow{C}$.

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

Corollary A.27 (*Completeness of λC for reduction*)

Assume that $\Gamma; \Delta \vdash_{\Sigma} T \Uparrow A$.

If $M \xrightarrow{\lambda C} T$ and $T \xrightarrow{C}_{\beta} T^* \xrightarrow{C}_{\text{AC}}^* T'$ with T' in AC-normal form, then there is a term M' such that $M \longrightarrow M'$ and $M' \xrightarrow{\lambda C} T'$.



Proof:

By the left-invertibility lemma A.17, there is a derivation of $T \xrightarrow{C\lambda} M$. By the soundness of $C\lambda$ with respect to β-reduction, there is a term M' such that $M \longrightarrow M'$ and $T^* \xrightarrow{C\lambda} M'$. By the invariance of $C\lambda$ under AC-reduction, there is a derivation of $T' \xrightarrow{C\lambda} M'$. By composing various typing soundness results, we obtain that $\Gamma; \Delta \vdash_{\Sigma}^{C} T \Uparrow A$, so that we can apply the left-invertibility lemma, obtaining that $M' \xrightarrow{\lambda C} T'$ is derivable. ☑

A.4 Properties of $C^{\rightarrow\multimap\&\top}$

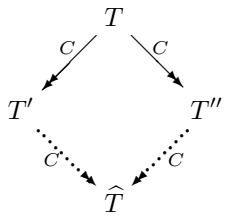
We will now present the main properties of $C^{\rightarrow\multimap\&\top}$, ultimately strong normalization and the uniqueness of normal forms with respect to both AC- and β-reducibility. In order to do so, we will take advantage of the facts that similar results hold for $\lambda^{\rightarrow\multimap\&\top}$, 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 $C^{\rightarrow\multimap\&\top}$ admits confluence and the Church-Rosser property.

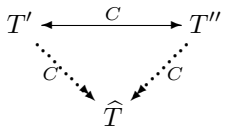
Theorem A.28 (*Church-Rosser*)

Confluence: *Assume that $\mathcal{Q} :: \Gamma; \Delta \vdash_{\Sigma}^{C} T \Uparrow A$.*

If $\mathcal{D}' :: T \xrightarrow{C}{}^* T'$ and $\mathcal{D}'' :: T \xrightarrow{C}{}^* T''$, then there is a term $\widehat{T}$ such that $T' \xrightarrow{C}{}^* \widehat{T}$ and $T'' \xrightarrow{C}{}^* \widehat{T}$.



Church-Rosser: *Assume that* $\mathcal{Q}' :: \Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow A$ *and* $\mathcal{Q}'' :: \Gamma; \Delta \vdash^{C}_{\Sigma} T'' \Uparrow A$.
If $\mathcal{D} :: T' \stackrel{C}{\equiv} T''$, *then there is a term* $\widehat{T}$ *such that* $T' \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}$ *and* $T'' \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}$.



Proof:

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

Since, by lemma A.15, $C\lambda$ is a total function over $C^{\rightarrow\multimap\&\top}$, there is a unique term M such that $T \stackrel{C\lambda}{\longrightarrow} M$ is derivable. By typing soundness, we obtain that $\Gamma; \Delta \vdash_{\Sigma} M \Uparrow A$. By iterated applications of the soundness of $C\lambda$ over reduction, we deduce that there are terms M' and M'' such that $M \longrightarrow^{*} M'$ and $T' \stackrel{C\lambda}{\longrightarrow} M'$, and similarly $M \longrightarrow^{*} M''$ and $T'' \stackrel{C\lambda}{\longrightarrow} M''$. 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 $\lambda^{\rightarrow\multimap\&\top}$, 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 $C^{\rightarrow\multimap\&\top}$ terms T^{*} and T^{**} such that $M' \stackrel{\lambda C}{\longrightarrow} T^{*}$ with $T' \stackrel{C}{\longrightarrow}{}^{*}_{\text{AC}} T^{*}$ and $M'' \stackrel{\lambda C}{\longrightarrow} T^{**}$ with $T'' \stackrel{C}{\longrightarrow}{}^{*}_{\text{AC}} T^{**}$. By the soundness of λC with respect to reductions, there are terms $\widehat{T}'$ and $\widehat{T}''$ such that $T^{*} \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}'$ and $N \stackrel{\lambda C}{\longrightarrow} \widehat{T}'$, and similarly $T^{**} \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}''$ and $N \stackrel{\lambda C}{\longrightarrow} \widehat{T}''$. However, since λC is a function, $\widehat{T}' = \widehat{T}''$; let us call this term $\widehat{T}$. By composing the various reductions above, we obtain the desired derivations of $T' \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}$ and $T'' \stackrel{C}{\longrightarrow}{}^{*} \widehat{T}$. ☑

Next, we consider the $C^{\rightarrow\multimap\&\top}$ equivalent of the Transitivity Lemma 2.3 discussed in Section 2. As in $\lambda^{\rightarrow\multimap\&\top}$, we must distinguish the linear and the unrestricted cases.

Lemma A.29 (*Transitivity*)

i. If $\mathcal{Q} :: \Gamma; \Delta, x{:}B \vdash^{C}_{\Sigma} T \Uparrow A$ *and* $\mathcal{R} :: \Gamma; \Delta' \vdash^{C}_{\Sigma} R \downarrow B$, *then* $\Gamma; \Delta, \Delta' \vdash^{C}_{\Sigma} [R/x]T \Uparrow A$.

ii. If $\mathcal{Q} :: \Gamma, x{:}B; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$ *and* $\mathcal{R} :: \Gamma; \cdot \vdash^{C}_{\Sigma} R \downarrow B$, *then* $\Gamma; \Delta \vdash^{C}_{\Sigma} [R/x]T \Uparrow A$.

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. ☑

The next property we are interested in proving for $C^{\rightarrow\multimap\&\top}$ is subject reduction. Remember that we have already proved this property in the subcase of AC-reduction.

Lemma A.30 (*Subject reduction*)

If $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$ *and* $\mathcal{D} :: T \stackrel{C}{\longrightarrow} T'$, *then* $\Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow A$.

Proof:

By the soundness of $C\lambda$ with respect to typing, there are a term M and derivations of $T \stackrel{C\lambda}{\longrightarrow} M$ and $\Gamma; \Delta \vdash_{\Sigma} M \Uparrow A$. By the soundness of $C\lambda$ with respect to reductions, there are a term M' and derivations of $T' \stackrel{C\lambda}{\longrightarrow} M'$ and $M \longrightarrow^{*} M'$. By the subject reduction property of $\lambda^{\rightarrow\multimap\&\top}$, $\Gamma; \Delta \vdash_{\Sigma} M' \Uparrow A$ is derivable.

Now, by the soundness of λC with respect to typing, there is a term T^{*} such that $\Gamma; \Delta \vdash^{C}_{\Sigma} T^{*} \Uparrow A$ and $M' \stackrel{C\lambda}{\longrightarrow} T^{*}$ are derivable. On the other hand, by the right-invertibility lemma A.16, there is a term T^{**} such that $M' \stackrel{\lambda C}{\longrightarrow} T^{**}$ and $T' \stackrel{S}{\longrightarrow}{}^{*}_{\text{NIL}} T^{**}$ are derivable. However, since, by Lemma A.9, λC is a function, we have that $T^{*} = T^{**}$. Then, in order to conclude this proof, we simply take advantage of the fact that AC-expansion preserves typing (Lemma A.2) to obtain the desired derivation of $\Gamma; \Delta \vdash^{C}_{\Sigma} T' \Uparrow A$. ☑

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

Theorem A.31 (*Strong normalization*)

If $\mathcal{Q} :: \Gamma; \Delta \vdash^{C}_{\Sigma} T \Uparrow A$, *then* T *is strongly normalizing.*

Proof:

Assume we have a (possibly infinite) sequence of terms $T_0, T_1, T_2, \ldots$ such that $T = T_0$ and there are derivations for the following reductions:

$$\sigma = T_0 \xrightarrow{C} T_1 \xrightarrow{C} T_2 \xrightarrow{C} \ldots$$

By the soundness of $C\lambda$ with respect to reducibility, every β-reduction in σ corresponds to a reduction in $\lambda^{\rightarrow\multimap\&\top}$ (Theorem A.25) while every AC-reduction disappears (Lemma A.21). This entails that there is a sequence of $\lambda^{\rightarrow\multimap\&\top}$ terms $M_0, M_1, M_2, \ldots$ such that on the one hand there are derivations of $T_i \xrightarrow{C\lambda} M_{\varphi(i)}$ where φ maps maximal subsequences of σ linked by AC-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 $T \xrightarrow{C\lambda} M_0$. Therefore, by the soundness of $C\lambda$ 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 AC-reduction (Lemma A.6), the maximal subsequences of AC-reducts collapsed by φ are finite. ☑

Strong normalization ensures that exhaustive reductions of a well-typed $C^{\rightarrow\multimap\&\top}$ term 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, up to the renaming of bound variables.

Corollary A.32 (*Uniqueness of normal forms*)

If $\mathcal{Q} :: \Gamma; \Delta \vdash^C_\Sigma T \Uparrow\Downarrow A$*, then there is a unique normal term* T' *such that* $T \xrightarrow{C}{}^* T'$.

Proof:

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

As in the case of $\lambda^{\rightarrow\multimap\&\top}$, the above results entitle us to speak about *the* normal form (or equivalently *the* canonical form) of a term T, whenever this object are well-typed. We denote this term $\mathrm{Can}(T)$. A calculus that accepts only canonical objects can be obtained from the typing system displayed in Figure 4.1 by simply removing rule **lC_redex**.

A term 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. We use $\overline{T}$ to denote the weak-head normal form of a term T.