Logical Methods in Computer Science
Vol. 7 (1:6) 2011, pp. 1–35
www.lmcs-online.org

Submitted Oct. 3, 2009
Published Mar. 22, 2011

A FOCUSED SEQUENT CALCULUS FRAMEWORK FOR PROOF-SEARCH IN PURE TYPE SYSTEMS

STÉPHANE LENGRAND [a], ROY DYCKHOFF [b], AND JAMES MCKINNA [c]

[a] CNRS, École Polytechnique, France
e-mail address: Lengrand@LIX.Polytechnique.fr

[b] School of Computer Science, University of St Andrews, Scotland
e-mail address: rd@cs.st-andrews.ac.uk

[c] Radboud University, Nijmegen, The Netherlands
e-mail address: james.mckinna@cs.ru.nl

ABSTRACT. Basic proof-search tactics in logic and type theory can be seen as the root-first applications of rules in an appropriate sequent calculus, preferably without the redundancies generated by permutation of rules. This paper addresses the issues of defining such sequent calculi for *Pure Type Systems* (PTS, which were originally presented in natural deduction style) and then organizing their rules for effective proof-search. We introduce the idea of *Pure Type Sequent Calculus with meta-variables* (PTSCα), by enriching the syntax of a permutation-free sequent calculus for propositional logic due to Herbelin, which is strongly related to natural deduction and already well adapted to proof-search. The operational semantics is adapted from Herbelin's and is defined by a system of local rewrite rules as in cut-elimination, using explicit substitutions. We prove confluence for this system. Restricting our attention to PTSC, a type system for the *ground* terms of this system, we obtain the *Subject Reduction* property and show that each PTSC is logically equivalent to its corresponding PTS, and the former is strongly normalising iff the latter is. We show how to make the logical rules of PTSC into a syntax-directed system PS for proof-search, by incorporating the conversion rules as in syntax-directed presentations of the PTS rules for type-checking. Finally, we consider how to use the explicitly scoped meta-variables of PTSCα to represent partial proof-terms, and use them to analyse interactive proof construction. This sets up a framework PE in which we are able to study proof-search strategies, type inhabitant enumeration and (higher-order) unification.

1998 ACM Subject Classification: F.4.1.
Key words and phrases: Type theory, PTS, sequent calculus, strong normalisation, proof-search, meta-variables, interactive proof construction.

DOI:10.2168/LMCS-7 (1:6) 2011

Contents

Introduction

Pure Type Systems (PTS) (see e.g. [Bar91]) were independently introduced by Berardi [Ber88] and Terlouw [Ter89] as a generalisation of Barendregt's λ-cube, and form a convenient framework for representing a range of different extensions of the simply-typed λ-calculus. *System F*, *System* F_ω [Gir72], *System* $\lambda\Pi$ [Daa80, HHP87], and the *Calculus of Constructions (CoC)* [CH88] are examples of such systems, on which several major proof assistants are based (e.g. Coq [Coq], Lego [LP92], and the *Edinburgh Logical Framework* [HHP87]; *Higher-Order Logic* can also be presented as a PTS, but this is *not* the basis of its principal implementation [HOL]).

With typed λ-calculus as their basis, such systems are traditionally presented in natural deduction style, with rules introducing and eliminating logical constants (aka type constructors). Dowek [Dow93] and Muñoz [Muñ01] show how to perform proof-search in this style, by enumerating type inhabitants.

This however misses out on the advantages of sequent calculus [Gen35] for proof-search. As suggested by Plotkin [Plo87], a Gentzen-style sequent calculus (with left and right introduction rules) can be used as a basis for proof-search in the case of $\lambda\Pi$ [PW91, Pym95] (later extended to any PTS [GR03a, GR03c]). However, the permutations of inference steps available in a Gentzen-style calculus (such as G3 [Kle52]) introduce some extra non-determinism in proof-search.

Herbelin [Her94, Her95] introduced a *permutation-free* calculus LJT for intuitionistic logic, exploiting the focusing ideas of Andreoli [And92], Danos *et al.* [DJS95] and (ultimately) ideas from Girard's linear logic [Gir87]. Herbelin's calculus has been considered as a basis for proof-search in intuitionistic logic [DP99b], generalising the uniform proof approach to logic programming (see [MNPS91] for hereditary Harrop logic). A version with cut rules and proof-terms forms an explicit substitution calculus $\overline{\lambda}$ [Her94, DU03] with a strong connection to (call-by-name) β-reduction and abstract machines such as that of Krivine [Kri].

This builds, as in the Curry-Howard correspondence, a computational interpretation of sequent calculus proofs on the basis of which type theory can be reformulated, now with a view to formalising proof-search. In earlier work [LDM06, Len06], we reformulated the language and proof theory of PTSs in terms of *Pure Type Sequent Calculi (PTSC)*. The present paper completes this programme, introducing *Pure Type Sequent Calculi with meta-variables (PTSCα)*, together with an operationalisation of proof-search in PTS in terms of PTSCα. It follows earlier work [PD98], relating $\overline{\lambda}$ to proof-search in the $\Lambda\Pi$ calculus [PW91, Pym95]. Introducing meta-variables for proof-search is the main technical novelty of this paper over [LDM06].

This gives a secure but simple theoretical basis for the implementation of PTS-based systems such as Coq [Coq] and Lego [LP92]; these proof assistants feature interactive proof construction methods using proof-search tactics. As observed by [McK97], the primitive tactics are not in exact correspondence with the elimination rules of the underlying natural deduction formalism: while the tactic intro *does* correspond to the right-introduction rule for Π-types (whether in natural deduction or in sequent calculus), the tactics apply in Coq and Refine in Lego, however, are much closer (in spirit) to the left-introduction rule $\Pi\mathsf{L}$ for Π-types in the focused sequent calculus LJT than to the Π-elimination rule in natural deduction. The $\Pi\mathsf{L}$ rule types the construct $M{\cdot}l$ of $\overline{\lambda}$, representing a list of terms with head M and tail l:

$$\frac{\Gamma \vdash M : A \quad \Gamma; \langle M/x \rangle B \vdash l : C}{\Gamma; \Pi x^A.B \vdash M{\cdot}l : C}\,\Pi\mathsf{L}$$

However, the aforementioned tactics are also able to postpone the investigation of the first premiss and start investigating the second. This leads to incomplete proof-terms and unification constraints to be solved. Here, we integrate these features into PTSC using explicitly scoped meta-variables. The resulting framework, called PTSCα, supports the analysis and definition of interactive proof construction tactics (as in Coq and Lego), as well as type inhabitant enumeration (see [Dow93, Muñ01]).

Of course, formalising proof-search mechanisms has already been investigated, if only to design tactic languages like Delahaye's $\mathcal{L}_{tac}$ and $\mathcal{L}_{pdt}$ [Del01]. Also noteworthy here are McBride's and Jojgov's PhD theses [McB00, GJ02], which consider extensions of type theory to admit partial proof objects. Using meta-variables similar to ours, Jojgov shows how to manage explicitly their progressive instantiation via a definitional mechanism and compares this with Delahaye's $\mathcal{L}_{tac}$ and $\mathcal{L}_{pdt}$.

While formalising the connections with this line of research remains as future work, the novelty of our approach here is to use the sequent calculus to bridge the usual gap (particularly wide for PTS and their implementations) between the rules defining a logic and the rules describing proof-search steps. A by-product of this bridge is ensuring correctness of proof-search, whose output thus need not be type-checked (which it currently is, in most proof assistants).

One reason why this is possible in our framework is that it can decompose (and thus account for) some mechanisms that are usually externalised and whose outputs usually need to be type-checked, such as unification (including higher-order [Hue76]). Indeed, it integrates the idea, first expounded in [Dow93], that proof-search and unification generalise in type theory to a single process.

The rules of our framework may not be deterministic enough to be considered as specifying an algorithm, but they are atomic enough to provide an operational semantics in which algorithms such as the above can be specified. They thus provide a semantics not only for type inhabitation algorithms, but also more generally for tactic languages, and, more originally, for unification algorithms.

As an example, we consider commutativity of conjunction expressed in (the $\mathsf{PTSC}\alpha$ corresponding to) System F, previously presented in [LDM06] without meta-variables. We show here how meta-variables improve the formalisation of proof-search.

Our work may be compared with that of our predecessors as follows:

	Type Theory	Inference rules	Proof-terms	Formalisation of incomplete proofs (by e.g. meta-variables)
[Pym95]	$\lambda\Pi$	G3	λ	YES
[Dow93]	CoC	NJ	λ	YES
[PD98]	$\lambda\Pi(\Sigma)$	LJT	$\overline{\lambda}$	NO
[GR03c]	PTS	G3	λ	NO
[GJ02]	λHOL	NJ	λ	YES
This paper	PTS	LJT	$\overline{\lambda}$	YES

Note that, in contrast to [Pym95, GR03a, GR03c], we use a focused sequent calculus (LJT) instead of an unfocused one (G3). The former forces proof-search to be 'goal-directed' in the tradition of logic programming and uniform proofs, while the latter is more relaxed and would accommodate saturation-based reasoning. Our choice here is motivated by a tighter connection with natural deduction and by the tactics currently used in proof assistants such as Coq and Lego. While [Pym95] does identify permutations of inference rules which would allow the recovery of a goal-directed strategy, [GR03c] focuses instead on the elimination of a cut-rule which then sheds a surprising light on the open problem of Expansion Postponement [GR03b].

Our move from G3 to LJT is also particularly convenient to capture the process of higher-order unification as a proof-search mechanism. Pym and Wallen address proof-search [PW91] in the particular case of $\lambda\Pi$, the type theory of the Edinburgh Logical Framework, using a black-box higher-order unification algorithm adapted from that of Huet. They discuss how well-typedness of meta-variable instantiations computed by unification can be exploited to control the search space. Meanwhile no meta-variables (or similar technology supporting unification) feature in [GR03a, GR03c].

In any case, this line of research keeps a traditional λ-calculus syntax for proof-terms, which thus does not reflect the structure of proof trees. We sought instead a formalism whose terms reflect how proofs and unifiers are constructed, and so moved from λ-calculus to $\overline{\lambda}$.

The paper's structure is as follows: Section 1 presents the syntax of $\mathsf{PTSC}\alpha$, the full language of terms and lists containing meta-variables, and gives the rewrite rules for normalisation. Section 2 relates this syntax with that of λ-calculus in PTS style and thereby derives the confluence of the $\mathsf{PTSC}\alpha$-calculus. Section 3 presents a parametric typing system PTSC for ground terms (i.e. the restriction to $\mathsf{PTSC}\alpha$-terms containing *no* meta-variables), and states and proves properties such as *Subject Reduction.* Section 4 establishes the correspondence between a PTSC and the PTS with the same parameters; we show type preservation and the strong normalisation result. Section 5 discusses proof-search in a PTSC. Section 6 introduces the inference system for $\mathsf{PTSC}\alpha$, as a way to formalise incomplete proofs and operationalise proof-search. Section 7 shows the aforementioned example. These are followed by a conclusion and discussion of directions for further work.

Some ideas and results of this paper (namely Sections 2, 3 and 4, which were already presented in [LDM06]) have been formalised and machine-checked in the Coq system [Sil09] using a de Bruijn index representation, as in e.g. [Len06].

1. Syntax and operational semantics of PTSCα

1.1. **Syntax.** We consider an extension (with type annotations) of the proof-term syntax $\overline{\lambda}$ of Herbelin's focused sequent calculus LJT [Her95]. As in $\overline{\lambda}$, the grammar of PTSCα features two syntactic categories: that of *terms* and that of *lists*.

The syntax depends on a given set $\mathcal{S}$ of *sorts*, written $s, s', \ldots$, a denumerable set $\mathcal{X}$ of *variables*, written $x, y, z, \ldots$, and two denumerable sets of *meta-variables*: those for terms, written $\alpha, \alpha', \ldots$, and those for lists, written $\beta, \beta', \ldots$. These meta-variables come with an intrinsic notion of *arity*.

Definition 1.1 (Terms and Lists)**.** The set $\mathcal{T}$ of *terms* (denoted $M, N, P, \ldots, A, B, \ldots$) and the set $\mathcal{L}$ of *lists* (denoted $l, l', \ldots$) are inductively defined by:

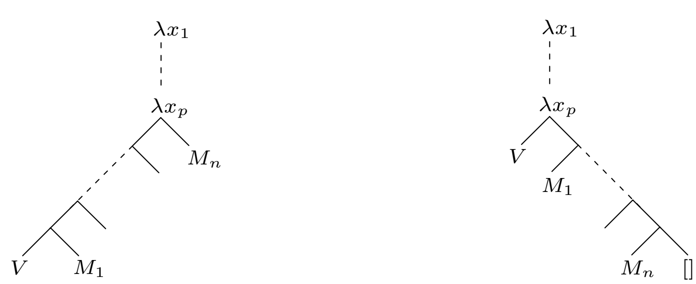
$$\begin{array}{rl} M, N, P, A, B & ::= \Pi x^A.B \mid \lambda x^A.M \mid s \mid x\ l \mid M\ l \mid \langle M/x\rangle N \mid \alpha(M_1, \ldots, M_n) \\ l, l' & ::= [] \mid M \cdot l \mid l @ l' \mid \langle M/x\rangle l \mid \beta(M_1, \ldots, M_n) \end{array}$$

where n is the arity of α and β.

The constructs $\Pi x^A.M$, $\lambda x^A.M$, and $\langle N/x\rangle M$ bind x in M, and $\langle M/x\rangle l$ binds x in l, thus defining the free variables of a term M (resp. a list l), denoted $\mathsf{FV}(M)$ (resp. $\mathsf{FV}(l)$), as well as α-conversion, issues of which are treated in the usual way. Note that $\mathsf{FV}(\alpha(M_1, \ldots, M_n)) = \mathsf{FV}(\beta(M_1, \ldots, M_n)) = \bigcup_{i=1}^{n} \mathsf{FV}(M_n)$; see the discussion on meta-variables below. A term M is *closed* if $\mathsf{FV}(M) = \emptyset$. As usual, let $A \to B$ denote $\Pi x^A.B$ when $x \notin \mathsf{FV}(B)$.

Terms and lists without meta-variables are called *ground terms* and *ground lists*, respectively. (Previously, these were just called terms and lists in [LDM06]).

Lists are used to represent sequences of arguments of a function; the term $x\ l$ (resp. $M\ l$) represents the application of x (resp. M) to the list of arguments l. Note that a variable alone is not a term; it must be applied to a list, possibly the empty list, denoted $[]$. The list $M \cdot l$ has head M and tail l, with a typing rule corresponding to the left-introduction of Π-types (cf. Section 3). The following figure shows the generic structure of a λ-term $\lambda x_1 \ldots \lambda x_p.V\ M_1 \ldots M_n$, and its $\overline{\lambda}$-representation as the term $\lambda x_1 \ldots \lambda x_p.V\ (M_1 \cdot \ldots M_n \cdot [])$, as follows:



Successive applications give rise to list concatenation, denoted $l @ l'$ (with @ acting as an explicit constructor). For instance, the list $(M_1 \cdot \ldots M_n \cdot [])@(M_{n+1} \cdot \ldots M_p \cdot [])$ will reduce to $M_1 \cdot \ldots M_n \cdot M_{n+1} \cdot \ldots M_p \cdot []$.

The terms $\langle M/x\rangle N$ and $\langle M/x\rangle l$ are explicit substitutions, on terms and lists, respectively. They will be used in two ways: first, to instantiate a universally quantified variable, and second, to describe explicitly the interaction between the constructors in the normalisation process (given in Section 1.2).

More intuition about Herbelin's calculus, its syntax and operational semantics, may be found in [Her95].

Among the features added to the syntax of $\overline{\lambda}$, our meta-variables can be seen as *higher-order variables*. As in CRS [Klo80], unknown terms are represented with (meta/higher-order) variables *applied* to the series of (term-)variables that could occur freely in those terms, e.g. $\alpha(x, y)$ (more formally, $\alpha(x\ [], y\ []))$ represents an unknown term M in which x and y could occur free (and no other). Such arguments x, y can later be instantiated, so that $\alpha(N, P)$ represents $\left\{{}^{N,P}\!/\!_{x,y}\right\}M$. In other words, a meta-variable by itself stands for something closed, i.e. a term under a series of bindings covering all its free variables, e.g. $x.y.M$ when $\mathsf{FV}(M) \subseteq \{x, y\}$ (using a traditional notation for higher-order terms, see e.g. [Ter03], Ch. 11).[1] This allows us to consider a simple notion of α-conversion, with $\lambda x^s.\alpha(x\ [], y\ []) \equiv_\alpha \lambda z^s.\alpha(z\ [], y\ [])$. Henceforth, however, we will elide further discussion of such matters, and simply write $=$ to denote $\equiv_\alpha$.

This kind of meta-variable differs from that in [Muñ01], which is rather in the style of ERS [Kha90] where the variables that could occur freely in the unknown term are not specified explicitly. The drawback of our approach is that we have to *know* in advance the free variables that might occur free in the unknown term, but in a typed setting such as proof-search these are actually the variables declared in the typing environment. Moreover, although specifying explicitly the variables that could occur free in an unknown term might seem heavy, it actually avoids the usual (non-)confluence problems when terms contain meta-variables in the style of ERS.[2] The solution in [Muñ01] has the drawback of not simulating β-reduction (although the reductions reach the expected normal forms).

1.2. **Operational semantics.** The operational semantics of $\mathsf{PTSC}\alpha$ is given by the system of reduction rules in Figure 1, comprising sub-systems B, x', and xsubst', and combinations thereof. This system extends that of [LDM06] with rules $\mathsf{A4}, \mathsf{C}\alpha, \mathsf{D}\beta$. Side-conditions to avoid variable capture can be inferred from the rules. We prove confluence in Section 2.

We denote by $\longrightarrow_G$ the contextual closure of the reduction relation defined by any system G of rewrite rules.[3] The transitive closure of $\longrightarrow_G$ is denoted by $\longrightarrow^+{}_G$, its reflexive and transitive closure is denoted by $\longrightarrow^*{}_G$, and its symmetric reflexive and transitive closure is denoted by $\longleftrightarrow^*{}_G$. The set of strongly normalising elements (those from which no infinite $\longrightarrow_G$-reduction sequence starts) is SN^G. When not specified, G is assumed to be the system B, x' from Fig. 1.

We now show that system x' is terminating. If we add rule B, then the system fails to be terminating unless we only consider terms that are typed in a normalising typing system.

[1] We develop this in Section 6 below. There is no binding mechanism for meta-variables in the syntax of $\mathsf{PTSC}\alpha$, but at the meta-level there is a natural notion of instantiation, also presented in Section 6. We thus emphasise the fact that instantiation of meta-variables never occurs during computation; in that respect, meta-variables really behave like constants or term constructors.

[2] See the discussion at the end of Section 2.

[3] Via contextual closure, a rewrite rule for terms can thus apply deep inside lists, and vice versa.

$$\begin{array}{llll} \mathsf{B} & (\lambda x^A.M)\ (N\cdot l) & \longrightarrow & (\langle N/x\rangle M)\ l \end{array}$$

$$\mathsf{x'}\left\{\begin{array}{l}
\begin{array}{llll}
\mathsf{B1} & M\ [] & \longrightarrow & M \\
\mathsf{B2} & (x\ l)\ l' & \longrightarrow & x\ (l@l') \\
\mathsf{B3} & (M\ l)\ l' & \longrightarrow & M\ (l@l') \\
\\
\mathsf{A1} & (M\cdot l')@l & \longrightarrow & M\cdot(l'@l) \\
\mathsf{A2} & []@l & \longrightarrow & l \\
\mathsf{A3} & (l@l')@l'' & \longrightarrow & l@(l'@l'') \\
\mathsf{A4} & l@[] & \longrightarrow & l
\end{array} \\
\\
\mathsf{xsubst'}\text{:}\left\{\begin{array}{lllll}
\mathsf{C1} & \langle P/y\rangle\lambda x^A.M & \longrightarrow & \lambda x^{\langle P/y\rangle A}.\langle P/y\rangle M & \\
\mathsf{C2} & \langle P/y\rangle(y\ l) & \longrightarrow & P\ \langle P/y\rangle l & \\
\mathsf{C3} & \langle P/y\rangle(x\ l) & \longrightarrow & x\ \langle P/y\rangle l & \text{if } x\neq y \\
\mathsf{C4} & \langle P/y\rangle(M\ l) & \longrightarrow & \langle P/y\rangle M\ \langle P/y\rangle l & \\
\mathsf{C5} & \langle P/y\rangle\Pi x^A.B & \longrightarrow & \Pi x^{\langle P/y\rangle A}.\langle P/y\rangle B & \\
\mathsf{C6} & \langle P/y\rangle s & \longrightarrow & s & \\
\\
\mathsf{C}\alpha & \langle P/y\rangle\alpha(M_1,\dots,M_n) & \longrightarrow & \alpha(\langle P/y\rangle M_1,\dots,\langle P/y\rangle M_n) & \\
\\
\mathsf{D1} & \langle P/y\rangle[] & \longrightarrow & [] & \\
\mathsf{D2} & \langle P/y\rangle(M\cdot l) & \longrightarrow & (\langle P/y\rangle M)\cdot(\langle P/y\rangle l) & \\
\mathsf{D3} & \langle P/y\rangle(l@l') & \longrightarrow & (\langle P/y\rangle l)@(\langle P/y\rangle l') & \\
\\
\mathsf{D}\beta & \langle P/y\rangle\beta(M_1,\dots,M_n) & \longrightarrow & \beta(\langle P/y\rangle M_1,\dots,\langle P/y\rangle M_n) &
\end{array}\right.
\end{array}\right.$$

Figure 1: Reduction Rules

We can define an encoding $\mathcal{S}(\_)$, given in Fig. 2, that maps terms and lists into a first-order syntax given by the following signature:

$$\{\star/0, \mathsf{i}/1, \mathsf{ii}/2, \mathsf{cut}/2, \mathsf{sub}/2\} \cup \{\mathsf{tuple}^n/n \mid n \in \mathbb{N}\}$$

which we then equip with the well-founded precedence relation defined by

$$\star \prec \mathsf{i} \prec \mathsf{ii} \prec \mathsf{tuple}^0 \prec \ldots \prec \mathsf{tuple}^n \prec \mathsf{tuple}^{n+1} \prec \ldots \prec \mathsf{cut} \prec \mathsf{sub}$$

The *lexicographic path ordering* (lpo) induced on the first-order terms is also well-founded (definitions and results can be found in [KL80], or [Ter03, ch. 6]).

Theorem 1.2.

- *If* $M \longrightarrow_{\mathsf{x'}} M'$ *then* $\mathcal{S}(M) >_{\mathsf{lpo}} \mathcal{S}(M')$.
- *If* $l \longrightarrow_{\mathsf{x'}} l'$ *then* $\mathcal{S}(l) >_{\mathsf{lpo}} \mathcal{S}(l')$.

Proof. By simultaneous induction on M, l. □

Corollary 1.3. *System* $\mathsf{x'}$ *is terminating (on all terms and lists).* □

$\mathcal{S}(s)$	$= \star$
$\mathcal{S}(\lambda x^A.M)$	$= \mathsf{ii}(\mathcal{S}(A), \mathcal{S}(M))$
$\mathcal{S}(\Pi x^A.M)$	$= \mathsf{ii}(\mathcal{S}(A), \mathcal{S}(M))$
$\mathcal{S}(x\ l)$	$= \mathsf{i}(\mathcal{S}(l))$
$\mathcal{S}(M\ l)$	$= \mathsf{cut}(\mathcal{S}(M), \mathcal{S}(l))$
$\mathcal{S}(\langle M/x \rangle N)$	$= \mathsf{sub}(\mathcal{S}(M), \mathcal{S}(N))$
$\mathcal{S}(\alpha(M_1, \ldots, M_n))$	$= \mathsf{tuple}^n(\mathcal{S}(M_1), \ldots, \mathcal{S}(M_n))$
$\mathcal{S}([])$	$= \star$
$\mathcal{S}(M \cdot l)$	$= \mathsf{ii}(\mathcal{S}(M), \mathcal{S}(l))$
$\mathcal{S}(l @ l')$	$= \mathsf{ii}(\mathcal{S}(l), \mathcal{S}(l'))$
$\mathcal{S}(\langle M/x \rangle l)$	$= \mathsf{sub}(\mathcal{S}(M), \mathcal{S}(l))$
$\mathcal{S}(\beta(M_1, \ldots, M_n))$	$= \mathsf{tuple}^n(\mathcal{S}(M_1), \ldots, \mathcal{S}(M_n))$

Figure 2: First-order encoding

2. λ-terms and Confluence

In this section we define translations between the syntax of $\mathsf{PTSC}\alpha$ and that of *Pure Type Systems* (PTS), i.e. a variant of λ-terms. Since, in the latter, the only reduction rule (namely, β) is confluent, we infer from the translations the confluence of $\mathsf{PTSC}\alpha$.

We briefly recall the framework of PTS. Terms have the following syntax:

$$t, u, v, T, U, V, \ldots ::= x \mid\ s \mid\ \Pi x^T.t \mid\ \lambda x^T.t \mid\ t\ u$$

with an operational semantics given by the contextual closure of the β-reduction rule $(\lambda x^v.t)\ u \longrightarrow_\beta \{{}^u\!/\!_x\}t$, in which the substitution is implicit, i.e. is a meta-operation.

Notice now that meta-variables in $\mathsf{PTSC}\alpha$ behave like constants of fixed arities during reduction; so it would be natural to reduce the confluence problem of $\mathsf{PTSC}\alpha$ to that of a λ-calculus extended with such constants. We avoid proving confluence of such an extension of PTS with constants. Instead we consider such a constant, say of arity k, directly as a free variable applied to (at least) k arguments (indeed, such an approach could also justify confluence for the extended system).

Consequently we set aside some of the traditional variables of PTS for the specific purpose of encoding meta-variables of $\mathsf{PTSC}\alpha$: for each meta-variable α (resp. β) of arity k, we reserve in the syntax of PTS a variable which we write α^k (resp. β^k).

For the remainder of this section, we therefore restrict our attention to that fragment, $\mathsf{PTS}\alpha$, of PTS-terms where such a variable α^k (resp. β^k) is never bound and is applied to at least k (resp. $k+1$) arguments. The only subtlety, explained below, is why β^k is applied to at least $k+1$ arguments (instead of the expected k).

Remark 2.1. The fragment $\mathsf{PTS}\alpha$ is stable under β-reduction,[4] and thus satisfies confluence.

Fig. 3 shows the translation of the syntax of $\mathsf{PTSC}\alpha$ into $\mathsf{PTS}\alpha$. While the translation of meta-variables for terms is natural, that of meta-variables for lists is more subtle, since the translation of lists is parameterised by the future head variable. How can we relate such

[4] By the capture-avoiding properties of β-reduction and the fact that, if an occurrence of a free variable is applied to (at least) k arguments, so are its residuals after a β-step.

$\mathcal{B}(\Pi x^A.B)$	$:=$	$\Pi x^{\mathcal{B}(A)}.\mathcal{B}(B)$	
$\mathcal{B}(\lambda x^A.M)$	$:=$	$\lambda x^{\mathcal{B}(A)}.\mathcal{B}(M)$	
$\mathcal{B}(s)$	$:=$	s	
$\mathcal{B}(x\ l)$	$:=$	$\{{}^{x}\!/_{z}\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}(M\ l)$	$:=$	$\{{}^{\mathcal{B}(M)}\!/_{z}\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}(\langle P/x\rangle M)$	$:=$	$\{{}^{\mathcal{B}(P)}\!/_{x}\}\mathcal{B}(M)$	
$\mathcal{B}(\alpha(M_1,\ldots,M_n))$	$:=$	$\alpha^n\ \mathcal{B}(M_1)\ldots\mathcal{B}(M_n)$	
$\mathcal{B}^y([])$	$:=$	y	
$\mathcal{B}^y(M\cdot l)$	$:=$	$\{{}^{y\ \mathcal{B}(M)}\!/_{z}\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}^y(l@l')$	$:=$	$\{{}^{\mathcal{B}^y(l)}\!/_{z}\}\mathcal{B}^z(l')$	z fresh
$\mathcal{B}^y(\langle P/x\rangle l)$	$:=$	$\{{}^{\mathcal{B}(P)}\!/_{x}\}\mathcal{B}^y(l)$	
$\mathcal{B}^y(\beta(M_1,\ldots,M_n))$	$:=$	$\beta^n\ y\ \mathcal{B}(M_1)\ldots\mathcal{B}(M_n)$	

Figure 3: From $\mathsf{PTSC}\alpha$ to $\mathsf{PTS}\alpha$

a variable to a list of terms that is (yet) unknown? We simply give it as an extra argument (the first one) of the encoded meta-variable.

Theorem 2.2 (Simulation of $\mathsf{PTSC}\alpha$)**.** *$\longrightarrow_\beta$ simulates $\longrightarrow_{\mathsf{Bx'}}$ through $\mathcal{B}$.*

Proof. If $M \longrightarrow_{\mathsf{B}} N$ then $\mathcal{B}(M) \longrightarrow^*_\beta \mathcal{B}(N)$, if $l \longrightarrow_{\mathsf{B}} l'$ then $\mathcal{B}^y(l) \longrightarrow^*_\beta \mathcal{B}^y(l')$, if $M \longrightarrow_{\mathsf{x'}} N$ then $\mathcal{B}(M) = \mathcal{B}(N)$ and if $l \longrightarrow_{\mathsf{x'}} l'$ then $\mathcal{B}^y(l) = \mathcal{B}^y(l')$, which are proved by simultaneous induction on the derivation step and case analysis. □

$\mathcal{A}(s)$	$:=$	s	
$\mathcal{A}(\Pi x^T.U)$	$:=$	$\Pi x^{\mathcal{A}(T)}.\mathcal{A}(U)$	
$\mathcal{A}(\lambda x^T.t)$	$:=$	$\lambda x^{\mathcal{A}(T)}.\mathcal{A}(t)$	
$\mathcal{A}(\alpha^k\ t_1\ldots t_k)$	$:=$	$\alpha(\mathcal{A}(t_1),\ldots,\mathcal{A}(t_k))$	
$\mathcal{A}(\beta^k\ t\ t_1\ldots t_k)$	$:=$	$\mathcal{A}_{\beta(\mathcal{A}(t_1),\ldots,\mathcal{A}(t_k))}(t)$	
$\mathcal{A}(t)$	$:=$	$\mathcal{A}_{[]}(t)$	otherwise
$\mathcal{A}_l(\alpha^k\ t_1\ldots t_k)$	$:=$	$\alpha(\mathcal{A}(t_1),\ldots,\mathcal{A}(t_k))\ l$	
$\mathcal{A}_l(\beta^k\ t\ t_1\ldots t_k)$	$:=$	$\mathcal{A}_{\beta(\mathcal{A}(t_1),\ldots,\mathcal{A}(t_k))@l}(t)$	
$\mathcal{A}_l(t\ u)$	$:=$	$\mathcal{A}_{\mathcal{A}(u)\cdot l}(t)$	otherwise
$\mathcal{A}_l(x)$	$:=$	$x\ l$	
$\mathcal{A}_l(t)$	$:=$	$\mathcal{A}(t)\ l$	otherwise

Figure 4: From $\mathsf{PTS}\alpha$ to $\mathsf{PTSC}\alpha$

Fig. 4 shows the translation from $\mathsf{PTS}\alpha$ into $\mathsf{PTSC}\alpha$.[5] It is simply the adaptation to the higher-order case of Prawitz's translation from natural deduction to sequent calculus [Pra65]: the translation $\mathcal{A}(t)$ of an application relies on a list-parameterised version $\mathcal{A}_l(t)$ of the translation. Example 2.8 below illustrates how the definitions in Fig. 4 and Fig. 3 expand.

[5]Note how we spot the situations which arise from encoded meta-variables, using the explicitly displayed arity to identify the arguments.

It is not obvious that the inductive definition of the translation is well-founded. To see this we need the following notion:

Definition 2.3 (List-needing terms)**.** We say that a λ-term t *needs* a list l if the pair (t, l) satisfies the following property: if $l = []$ then t is either a variable or an application that is not of the form $\alpha^k\ t_1 \dots t_k$.[6]

The inductive definition of the translation is done by structural induction on the term, subject to the consideration that $\mathcal{A}_l(t)$ is defined before $\mathcal{A}(t)$ if t needs l, and that $\mathcal{A}_l(t)$ is defined after $\mathcal{A}(t)$ if not. The terminology comes from the fact that t needs l if and only if $\mathcal{A}_l(t)$ is **not** a $\mathsf{B1}$-redex.

In order to prove confluence, we first need the following results:

Lemma 2.4.

(1) *$\mathcal{A}(t)$ is an x'-normal form.*
If l is x'-normal and t needs l then $\mathcal{A}_l(t)$ is x'-normal.
(2) *If $l \longrightarrow_{\mathsf{Bx}'} l'$ then $\mathcal{A}_l(t) \longrightarrow_{\mathsf{Bx}'} \mathcal{A}_{l'}(t)$.*
(3) *$\mathcal{A}_{l'}(t)\ l \longrightarrow^*_{\mathsf{x}'} \mathcal{A}_{l'@l}(t)$ and $\mathcal{A}(t)\ l \longrightarrow^*_{\mathsf{x}'} \mathcal{A}_l(t)$.*
(4) *$\langle \mathcal{A}(u)/x \rangle \mathcal{A}(t) \longrightarrow^*_{\mathsf{x}'} \mathcal{A}(\{u/x\}t)$ and $\langle \mathcal{A}(u)/x \rangle \mathcal{A}_l(t) \longrightarrow^*_{\mathsf{x}'} \mathcal{A}_{\langle \mathcal{A}(u)/x \rangle l}(\{u/x\}t)$.*

Proof. Each point is obtained by straightforward induction on t. Note that in order to prove point 4 we need rules $\mathsf{A3}$ and $\mathsf{A4}$. These are not needed (for simulation of β-reduction and for confluence) when only ground terms are concerned. $\square$

Theorem 2.5 (Simulation of PTS)**.**
$\longrightarrow_{\mathsf{Bx}'}$ (strongly) simulates $\longrightarrow_\beta$ through $\mathcal{A}$.

Proof. If $t \longrightarrow_\beta u$ then $\mathcal{A}(t) \longrightarrow^+_{\mathsf{Bx}'} \mathcal{A}(u)$ and $\mathcal{A}_l(t) \longrightarrow^+_{\mathsf{Bx}'} \mathcal{A}_l(u)$, each proved by induction on the derivation step, using Lemma 2.4.4 for the base case and Lemma 2.4.3. $\square$

Now we study the composition of the two translations:

Lemma 2.6. *Suppose M and l are x'-normal forms.*
(1) *If t needs l then $\mathcal{A}_l(t) = \mathcal{A}(\{t/x\}\mathcal{B}^x(l))$ (for any $x \notin \mathsf{FV}(l)$).*
(2) *$M = \mathcal{A}(\mathcal{B}(M))$.*

Proof. By simultaneous induction on l and M. Again, rules $\mathsf{A3}$ and $\mathsf{A4}$ (as well as $\mathsf{C}\alpha$ and $\mathsf{D}\beta$) are needed for this lemma to capture the notion of normal form corresponding to the PTS-terms, when meta-variables are present. $\square$

Theorem 2.7.
(1) $\mathcal{B}(\mathcal{A}(t)) = t$
(2) $M \longrightarrow^*_{\mathsf{x}'} \mathcal{A}(\mathcal{B}(M))$

Proof.
(1) $\mathcal{B}(\mathcal{A}(t)) = t$ and $\mathcal{B}(\mathcal{A}_l(t)) = \{t/x\}\mathcal{B}^x(l)$ (with $x \neq \mathsf{FV}(l)$) are obtained by simultaneous induction on t.
(2) $M \longrightarrow^*_{\mathsf{x}'} \mathcal{A}(\mathcal{B}(M))$ holds by induction on the longest sequence of x'-reduction from M (x' is terminating): by Lemma 2.6.2, it holds if M is an x'-normal form, and if $M \longrightarrow_{\mathsf{x}'} N$ then we can apply the induction hypothesis on N and by Theorem 2.2 we have the result. $\square$

[6] Remember that we suppose that α^k is applied to at least k arguments.

Example 2.8. Here is an example illustrating Theorem 2.7.1:

$$
\begin{array}{rlrl}
 & \mathcal{B}(\mathcal{A}(\beta^k(x\ y)t_1 \dots t_k)) & = & \mathcal{B}(\mathcal{A}_D(x\ y)) \\
= & \mathcal{B}(x\ (y\ [])\cdot D) & = & \mathcal{B}^x((y\ [])\cdot D) \\
= & \left\{{}^{x\ \mathcal{B}(y\ [])}\!/_{z}\right\} \mathcal{B}^z(D) & = & \left\{{}^{x\ \mathcal{B}^y([])}\!/_{z}\right\} \mathcal{B}^z(D) \\
= & \left\{{}^{x\ y}\!/_{z}\right\} \mathcal{B}^z(D) & = & \left\{{}^{x\ y}\!/_{z}\right\} (\beta^k\ z\ \mathcal{B}(A(t_1)) \dots \mathcal{B}(A(t_k))) \\
= & \beta^k(x\ y)\mathcal{B}(\mathcal{A}(t_1)) \dots \mathcal{B}(\mathcal{A}(t_k)) & &
\end{array}
$$

where $D = \beta(A(t_1), \dots, A(t_k))$.

We finally get confluence:

Corollary 2.9 (Confluence)**.** $\longrightarrow_{\mathsf{x}'}$ *and* $\longrightarrow_{\mathsf{Bx}'}$ *are confluent.*

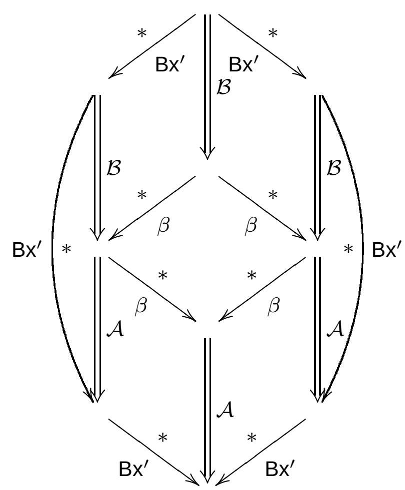


Figure 5: Confluence by simulation

Proof. We use the simulation technique, as for instance in [KL05]: consider two reduction sequences starting from a term in $\mathsf{PTSC}\alpha$. They can be simulated through $\mathcal{B}$ by β-reductions, and since $\mathsf{PTS}\alpha$ is confluent, we can close the diagram. Now the lower part of the diagram can be simulated through $\mathcal{A}$ back in $\mathsf{PTSC}\alpha$, which closes the diagram there as well, as shown in Fig. 5 for Bx'. Notice that the proof of confluence has nothing to do with typing and does not rely on any result in Section 3 (in fact, we use confluence in the proof of Subject Reduction in the Appendix). □

Considering meta-variables in the style of CRS [Klo80] avoids the usual problem of non-confluence coming from the critical pair between B and C4 which generate the two terms $\langle N/x\rangle\langle P/y\rangle M$ and $\langle\langle N/x\rangle P/y\rangle\langle N/x\rangle M$. Indeed, with ERS-style meta-variables these two terms need not reduce to a common term, but with the CRS-approach, they now can (using the rules $\mathsf{C}\alpha$ and $\mathsf{D}\beta$). Again, note how the critical pair between B3 and itself (or B2) needs rule A3 in order to be closed, while it was only there for convenience when all terms were ground.

3. Typing system and properties

Throughout this section we consider PTSC, that is, the restriction to *ground* terms of $\mathsf{PTSC}\alpha$. We thus do not need to consider any notion of meta-variable, nor that of any special variable distinguished among PTS terms, such as those considered in the previous section.

Given the set of sorts $\mathcal{S}$, a particular PTSC is specified by a set $\mathcal{A} \subseteq \mathcal{S}^2$ and a set $\mathcal{R} \subseteq \mathcal{S}^3$. We shall see an example in Section 4.2.

Definition 3.1 (Typing Environments)**.**

- A *typing environment* (henceforth simply: 'environment', for brevity's sake) is a list Γ of pairs taken from $\mathcal{X} \times \mathcal{T}$, denoted $(x : A)$.
- We define the *domain* of an environment and the *application of a substitution to an environment* as follows:

$$\begin{array}{ll} \mathsf{Dom}(\emptyset) = \emptyset & \mathsf{Dom}(\Gamma, (x : A)) = \mathsf{Dom}(\Gamma), x \\ \langle P/y \rangle(\emptyset) = \emptyset & \langle P/y \rangle(\Gamma, (x : A)) = \langle P/y \rangle\Gamma, (x : \langle P/y \rangle A) \end{array}$$

- It is useful (see Section 6) to define $\mathsf{Dom}(\Gamma)$ as a list, for which the meaning of $x \in \mathsf{Dom}(\Gamma)$ is clear. If $\mathcal{M}$ is a *set* of variables, $\mathcal{M} \subseteq \mathsf{Dom}(\Gamma)$ means for all $x \in \mathcal{M}$, $x \in \mathsf{Dom}(\Gamma)$. Similarly, $\mathsf{Dom}(\Gamma) \cap \mathsf{Dom}(\Delta)$ is the *set* $\{x \in \mathcal{X} \mid x \in \mathsf{Dom}(\Gamma) \wedge x \in \mathsf{Dom}(\Delta)\}$.

  We define the following *inclusion relation* between environments:

$$\Gamma \sqsubseteq \Delta \text{ if for all } (x : A) \in \Gamma, \text{ there is } (x : B) \in \Delta \text{ with } A \longleftrightarrow^* B.$$

The inference rules in Fig. 6 inductively define the derivability of three kinds of statement:

(1) $\Gamma\ \mathsf{wf}$
Intuitively, the derivability of this statement means that the environment Γ is well-formed.

(2) $\Gamma \vdash M : A$ 'term typing'
Intuitively, the derivability of this statement means that M is of type A in the environment Γ (is a proof of A from the assumptions in Γ).

(3) $\Gamma; B \vdash l : C$ 'list typing'
The position of B in the sequent is a special place called the *stoup*. Intuitively, the derivability of this statement means that, in the environment Γ, the list l codes for an actual list of terms such that, when something of type B is applied to them, the result is of type C (this codes for a natural deduction of C from B by a series of Π-elimination rules, whose minor premisses are derived by the proofs-terms in l using the assumptions in Γ).

Side-conditions are used, such as $(s_1, s_2, s_3) \in \mathcal{R}$, $x \notin \mathsf{Dom}(\Gamma)$, $A \longleftrightarrow^* B$ or $\Gamma \sqsubseteq \Delta$, and we use the abbreviation $\Gamma \sqsubseteq \Delta\ \mathsf{wf}$ for $\Gamma \sqsubseteq \Delta$ and $\Delta\ \mathsf{wf}$. We freely abuse the notation in the customary way, by not distinguishing between a statement and its derivability according to the rules of Fig. 6.

There are three conversion rules conv_R, conv'_R, and conv_L in order to deal with the two kinds of typing statement and, for list typing, also to be able to convert the type in the stoup.

Because substituting for a variable in an environment affects the rest of the environment (which could depend on that variable), the two rules for explicit substitutions (Cut_2 and Cut_4) must have a particular shape that manipulates the environment, if the PTSC is to satisfy basic required properties like those of a PTS.

$$\frac{}{\emptyset\ \mathsf{wf}}\,\mathsf{empty} \qquad \frac{\Gamma \vdash A\!:\!s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, (x\!:\!A)\ \mathsf{wf}}\,\mathsf{extend}$$

$$\frac{\Gamma\ \mathsf{wf} \quad (s, s') \in \mathcal{A}}{\Gamma \vdash s\!:\!s'}\,\mathsf{sorted} \qquad \frac{\Gamma \vdash A\!:\!s_1 \quad \Gamma, (x\!:\!A) \vdash B\!:\!s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^A.B\!:\!s_3}\,\Pi\mathsf{wf}$$

$$\frac{\Gamma \vdash \Pi x^A.B\!:\!s \quad \Gamma, (x\!:\!A) \vdash M\!:\!B}{\Gamma \vdash \lambda x^A.M\!:\!\Pi x^A.B}\,\Pi\mathsf{R} \qquad \frac{\Gamma; A \vdash l\!:\!B \quad (x\!:\!A) \in \Gamma}{\Gamma \vdash x\ l\!:\!B}\,\mathsf{Select}_x \qquad \frac{\Gamma \vdash A\!:\!s}{\Gamma; A \vdash [\,]\!:\!A}\,\mathsf{axiom}$$

$$\frac{\Gamma \vdash M\!:\!A \quad \Gamma \vdash B\!:\!s \quad A \longleftrightarrow^* B}{\Gamma \vdash M\!:\!B}\,\mathsf{conv}_R \qquad \frac{\Gamma \vdash \Pi x^A.B\!:\!s \quad \Gamma \vdash M\!:\!A \quad \Gamma; \langle M/x\rangle B \vdash l\!:\!C}{\Gamma; \Pi x^A.B \vdash M \cdot l\!:\!C}\,\Pi\mathsf{L}$$

$$\frac{\Gamma; C \vdash l\!:\!A \quad \Gamma \vdash B\!:\!s \quad A \longleftrightarrow^* B}{\Gamma; C \vdash l\!:\!B}\,\mathsf{conv}'_R \qquad \frac{\Gamma; A \vdash l\!:\!C \quad \Gamma \vdash B\!:\!s \quad A \longleftrightarrow^* B}{\Gamma; B \vdash l\!:\!C}\,\mathsf{conv}_L$$

$$\frac{\Gamma; C \vdash l'\!:\!A \qquad \Gamma; A \vdash l\!:\!B}{\Gamma; C \vdash l' @ l\!:\!B}\,\mathsf{Cut}_1 \qquad \frac{\Gamma \vdash P\!:\!A \qquad \Gamma, (x\!:\!A), \Delta; B \vdash l\!:\!C \quad \Gamma, \langle P/x\rangle\Delta \sqsubseteq \Delta'\ \mathsf{wf}}{\Delta'; \langle P/x\rangle B \vdash \langle P/x\rangle l\!:\!\langle P/x\rangle C}\,\mathsf{Cut}_2$$

$$\frac{\Gamma \vdash M\!:\!A \qquad \Gamma; A \vdash l\!:\!B}{\Gamma \vdash M\ l\!:\!B}\,\mathsf{Cut}_3 \qquad \frac{\Gamma \vdash P\!:\!A \qquad \Gamma, (x\!:\!A), \Delta \vdash M\!:\!C \quad \Gamma, \langle P/x\rangle\Delta \sqsubseteq \Delta'\ \mathsf{wf}}{\Delta' \vdash \langle P/x\rangle M\!:\!C'}\,\mathsf{Cut}_4$$

where either ($C' = C \in \mathcal{S}$) or $C \notin \mathcal{S}$ and $C' = \langle P/x\rangle C$

Figure 6: Typing rules of a PTSC

Example 3.2. Here is, as an example, a derivation of $x : s_1 \vdash x\ [\,] : s_1$ in a PTSC where $(s_1, s_2) \in \mathcal{A}$.

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset\ \mathsf{wf}}\,\mathsf{empty} \qquad (s_1, s_2) \in \mathcal{A}}{\emptyset \vdash s_1\!:\!s_2}\,\mathsf{sorted}}{x\!:\!s_1\ \mathsf{wf}}\,\mathsf{extend} \qquad (s_1, s_2) \in \mathcal{A}}{x\!:\!s_1 \vdash s_1\!:\!s_2}\,\mathsf{sorted}}{x\!:\!s_1; s_1 \vdash [\,]\!:\!s_1}\,\mathsf{axiom}}{x\!:\!s_1 \vdash x\ [\,]\!:\!s_1}\,\mathsf{Select}_x$$

The lemmas of this section are proved by straightforward inductions on typing derivations:

Lemma 3.3 (Properties of typing statements)**.** *If $\Gamma \vdash M\!:\!A$ (respectively, $\Gamma; B \vdash l\!:\!C$) then $\mathsf{FV}(M) \subseteq \mathsf{Dom}(\Gamma)$ (respectively, $\mathsf{FV}(l) \subseteq \mathsf{Dom}(\Gamma)$), and the following statements can be derived with strictly smaller typing derivations:*

(1) Γ *wf*
(2) $\Gamma \vdash A\!:\!s$ *for some* $s \in \mathcal{S}$*, or* $A \in \mathcal{S}$
(resp. $\Gamma \vdash B\!:\!s$ *and* $\Gamma \vdash C\!:\!s'$ *for some* $s, s' \in \mathcal{S}$*)* □

Corollary 3.4 (Properties of well-formed environments)**.**
(1) *If* $\Gamma, x : A, \Delta$ *wf then* $\Gamma \vdash A : s$ *for some* $s \in \mathcal{S}$ *with* $x \notin \mathsf{Dom}(\Gamma, \Delta)$ *and* $\mathsf{FV}(A) \subseteq \mathsf{Dom}(\Gamma)$ *(and in particular* $x \notin \mathsf{FV}(A)$*)*
(2) *If* Γ, Δ *wf then* Γ *wf.* □

Lemma 3.5 (Weakening)**.** *Suppose* Γ, Γ' *wf and* $\mathsf{Dom}(\Gamma') \cap \mathsf{Dom}(\Delta) = \emptyset$.
(1) *If* $\Gamma, \Delta \vdash M\!:\!A$ *then* $\Gamma, \Gamma', \Delta \vdash M\!:\!A$.
(2) *If* $\Gamma, \Delta; B \vdash l\!:\!C$*, then* $\Gamma, \Gamma', \Delta; B \vdash l\!:\!C$.
(3) *If* Γ, Δ *wf, then* Γ, Γ', Δ *wf.* □

We can also strengthen the *weakening property* into the *thinning property* by induction on the typing derivation. This allows to weaken the environment, permute it, and convert the types inside, as long as it remains well-formed:

Lemma 3.6 (Thinning)**.** *Suppose* $\Gamma \sqsubseteq \Delta$ *wf.*
(1) *If* $\Gamma \vdash M\!:\!A$ *then* $\Delta \vdash M\!:\!A$.
(2) *If* $\Gamma; B \vdash l\!:\!C$*, then* $\Delta; B \vdash l\!:\!C$. □

Using all of the results above, we obtain Subject Reduction:

Theorem 3.7 (Subject Reduction in a PTSC)**.**
(1) *If* $\Gamma \vdash M\!:\!A$ *and* $M \longrightarrow M'$*, then* $\Gamma \vdash M'\!:\!A$
(2) *If* $\Gamma; B \vdash l\!:\!C$ *and* $l \longrightarrow l'$*, then* $\Gamma; B \vdash l'\!:\!C$

Proof. See the Appendix. □

4. Correspondence with PTS

4.1. **Type preservation.** There is a logical correspondence between a PTSC given by the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ and the associated PTS given by the same sets.

We prove this by showing that (when restricted to ground terms) the translations preserve typing.

Terms in PTS are typed according to the typing rules in Fig. 4.1, which depend on the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$. Besides confluence for β-reduction, PTSs have the following meta-theoretic properties (for proofs, see e.g. [Bar92]):

Theorem 4.1.
(1) *If* $\Gamma \vdash_{\mathsf{PTS}} t\!:\!T$ *and* $\Gamma \sqsubseteq \Delta$ *wf then* $\Delta \vdash_{\mathsf{PTS}} t\!:\!T$ *(where the relation* $\sqsubseteq$ *is defined similarly to that of PTSC, but with* β*-equivalence).*
(2) *If* $\Gamma \vdash_{\mathsf{PTS}} t\!:\!T$ *and* $\Gamma, y : T, \Delta \vdash_{\mathsf{PTS}} u\!:\!U$
then $\Gamma, \left\{{}^{t}\!/\!{}_{y}\right\}\Delta \vdash_{\mathsf{PTS}} \left\{{}^{t}\!/\!{}_{y}\right\}u\!:\!\left\{{}^{t}\!/\!{}_{y}\right\}U$.
(3) *If* $\Gamma \vdash_{\mathsf{PTS}} t\!:\!T$ *and* $t \longrightarrow_{\beta} u$ *then* $\Gamma \vdash_{\mathsf{PTS}} u\!:\!T$. □

$$\frac{}{\emptyset\ \mathsf{wf}} \qquad \frac{\Gamma \vdash_{\mathsf{PTS}} T\!:\!s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, (x:T)\ \mathsf{wf}} \qquad \frac{\Gamma\ \mathsf{wf} \quad (x:T) \in \Gamma}{\Gamma \vdash_{\mathsf{PTS}} x\!:\!T}$$

$$\frac{\Gamma\ \mathsf{wf} \quad (s,s') \in \mathcal{A}}{\Gamma \vdash_{\mathsf{PTS}} s\!:\!s'} \qquad \frac{\Gamma \vdash_{\mathsf{PTS}} U\!:\!s_1 \quad \Gamma, (x:U) \vdash_{\mathsf{PTS}} T\!:\!s_2 \quad (s_1,s_2,s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathsf{PTS}} \Pi x^U.T\!:\!s_3}$$

$$\frac{\Gamma \vdash_{\mathsf{PTS}} \Pi x^U.T\!:\!s \quad \Gamma, (x:U) \vdash_{\mathsf{PTS}} t\!:\!T}{\Gamma \vdash_{\mathsf{PTS}} \lambda x^U.t\!:\!\Pi x^U.T} \qquad \frac{\Gamma \vdash_{\mathsf{PTS}} t\!:\!\Pi x^U.T \quad \Gamma \vdash_{\mathsf{PTS}} u\!:\!U}{\Gamma \vdash_{\mathsf{PTS}} t\ u\!:\!\{{}^{u}\!/\!{}_{x}\}T}$$

$$\frac{\Gamma \vdash_{\mathsf{PTS}} t\!:\!U \quad \Gamma \vdash_{\mathsf{PTS}} V\!:\!s \quad U \longleftrightarrow^*_\beta V}{\Gamma \vdash_{\mathsf{PTS}} t\!:\!V}$$

Figure 7: Typing rules of a PTS

We now extend the translations to environments:

$\mathcal{A}(\emptyset) = []$ $\qquad$ $\mathcal{B}(\emptyset) = []$

$\mathcal{A}(\Gamma, (x:T)) = \mathcal{A}(\Gamma), (x:\mathcal{A}(T))$ $\qquad$ $\mathcal{B}(\Gamma, (x:A)) = \mathcal{B}(\Gamma), (x:\mathcal{B}(A))$

Now note that the simulations in Section 2 imply:

Corollary 4.2 (Equational theories)**.**
$t \longleftrightarrow^*_\beta u$ *if and only if* $\mathcal{A}(t) \longleftrightarrow^* \mathcal{A}(u)$
$M \longleftrightarrow^* N$ *if and only if* $\mathcal{B}(M) \longleftrightarrow^*_\beta \mathcal{B}(N)$ □

Preservation of typing is proved by induction on the typing derivations:

Theorem 4.3 (Preservation of typing 1)**.**
(1) *If* $\Gamma \vdash_{\mathsf{PTS}} t\!:\!T$ *then* $\mathcal{A}(\Gamma) \vdash \mathcal{A}(t)\!:\!\mathcal{A}(T)$
(2) *If* $(\Gamma \vdash_{\mathsf{PTS}} t_i\!:\!\{{}^{t_{i-1}}\!/\!{}_{x_{i-1}}\}\cdots\{{}^{t_1}\!/\!{}_{x_1}\}T_i)_{i=1\ldots n}$
and $\mathcal{A}(\Gamma) \vdash \mathcal{A}(\Pi x_1{}^{T_1}.\ldots\Pi x_n{}^{T_n}.T)\!:\!s$
then $\mathcal{A}(\Gamma); \mathcal{A}(\Pi x_1{}^{T_1}.\ldots\Pi x_n{}^{T_n}.T) \vdash \mathcal{A}(t_1\ldots t_n)\!:\!\mathcal{A}(\{{}^{t_n}\!/\!{}_{x_n}\}\cdots\{{}^{t_1}\!/\!{}_{x_1}\}T)$
(3) *If* Γ *wf then* $\mathcal{A}(\Gamma)$ *wf* □

Theorem 4.4 (Preservation of typing 2)**.**
(1) *If* $\Gamma \vdash M\!:\!A$ *then* $\mathcal{B}(\Gamma) \vdash_{\mathsf{PTS}} \mathcal{B}(M)\!:\!\mathcal{B}(A)$
(2) *If* $\Gamma; B \vdash l\!:\!C$ *then* $\mathcal{B}(\Gamma), y:\mathcal{B}(B) \vdash_{\mathsf{PTS}} \mathcal{B}^y(l)\!:\!\mathcal{B}(C)$ *for any fresh* y
(3) *If* Γ *wf then* $\mathcal{B}(\Gamma)$ *wf* □

4.2. **Equivalence of Strong Normalisation.**

Theorem 4.5. *A PTSC given by the sets* $\mathcal{S}$*,* $\mathcal{A}$*, and* $\mathcal{R}$ *is strongly normalising if and only if the corresponding PTS given by the same sets is.*

Proof. Assume that the PTSC is strongly normalising, and let us consider a well-typed t of the corresponding PTS, i.e. $\Gamma \vdash_{\mathsf{PTS}} t{:}T$ for some Γ, T. By Theorem 4.3, $\mathcal{A}(\Gamma) \vdash \mathcal{A}(t){:}\mathcal{A}(T)$ so $\mathcal{A}(t) \in \mathsf{SN}$. Now by Theorem 2.5, any reduction sequence starting from t maps to a reduction sequence of at least the same length starting from $\mathcal{A}(t)$, but those are finite.

Now assume that the PTS is strongly normalising and that $\Gamma \vdash M : A$ in the corresponding PTSC. By subject reduction, any N such that $M \longrightarrow^* N$ satisfies $\Gamma \vdash N : A$ and any sub-term P (resp. sub-list l) of any such N is also typable. By Theorem 4.4, for any such P (resp. l), $\mathcal{B}(P)$ (resp. $\mathcal{B}^y(l)$) is typable in the PTS, so it is strongly normalising by assumption.

We now refine the first-order encoding of any such P and l (as defined in Section 1), emulating the technique of Bloo and Geuvers [BG99].

Accordingly, we refine the first-order signature from Section 1 by *labelling* the symbols $\mathsf{cut}^t(\_,\_)$ and $\mathsf{sub}^t(\_,\_)$ with all strongly normalising terms t of a PTS, thus generating an infinite signature. The precedence relation is refined as follows

$$\star \prec \mathsf{i}(\_) \prec \mathsf{ii}(\_,\_) \prec \mathsf{cut}^t(\_,\_) \prec \mathsf{sub}^t(\_,\_)$$

but we also set $\mathsf{sub}^t(\_,\_) \prec \mathsf{cut}^{t'}(\_,\_)$ whenever $t' \longrightarrow^+{}_\beta\, t$. The precedence is still well-founded, so the induced (lpo) is also still well-founded (definitions and results can be found in [KL80]). The refinement of the encoding is given in Fig 8. An induction on terms shows that reductions decrease the lpo. $\square$

$\mathcal{T}(s)$		$= \star$
$\mathcal{T}(\lambda x^A.M)$	$= \mathcal{T}(\Pi x^A.M)$	$= \mathsf{ii}(\mathcal{T}(A), \mathcal{T}(M))$
$\mathcal{T}(x\ l)$		$= \mathsf{i}(\mathcal{T}(l))$
$\mathcal{T}(M\ l)$		$= \mathsf{cut}^{\mathcal{B}(M\ l)}(\mathcal{T}(M), \mathcal{T}(l))$
$\mathcal{T}(\langle M/x\rangle N)$		$= \mathsf{sub}^{\mathcal{B}(\langle M/x\rangle N)}(\mathcal{T}(M), \mathcal{T}(N))$
$\mathcal{T}([])$		$= \star$
$\mathcal{T}(M \cdot l)$		$= \mathsf{ii}(\mathcal{T}(M), \mathcal{T}(l))$
$\mathcal{T}(l@l')$		$= \mathsf{ii}(\mathcal{T}(l), \mathcal{T}(l'))$
$\mathcal{T}(\langle M/x\rangle N)$		$= \mathsf{sub}^{\mathcal{B}(\langle M/x\rangle l)}(\mathcal{T}(M), \mathcal{T}(l))$

Figure 8: First-order encoding

Examples of strongly normalising PTS are the *Calculus of Constructions* [CH88], on which the proof-assistant Coq is based [Coq] (but it also uses inductive types and local definitions), as well as the other systems of Barendregt's Cube, for all of which we now have a corresponding PTSC that can be used for proof-search.

5. Proof-search

Proof-search considers as inputs an environment Γ and a type A, and the output, if successful, will be a term M such that $\Gamma \vdash M{:}A$, moreover one in normal form. When we search for a list l such that $\Gamma; B \vdash l{:}C$, the type B in the stoup is also an input. Henceforth, such a term type A or list type C will be called simply a *goal*.

The inference rules now need to be *syntax-directed*, that is determined by the shape of the goal (or of the type in the stoup), and the proof-search system (PS, for short) is then obtained by optimising appeals to the conversion rules, yielding the presentation given in

Fig. 9. The incorporation of the conversion rules into the other rules is similar to that of the Constructive Engine in natural deduction [Hue89, vBJMP94]; however that algorithm was designed for type synthesis, for which the inputs and outputs are not the same as in proof-search, as mentioned in the introduction.

$$\frac{D \longleftrightarrow^* C}{\Gamma; D \vdash_{\mathsf{PS}} [\,]:C}\,\mathsf{axiom} \qquad \frac{D \longrightarrow^* \Pi x^A.B \quad \Gamma \vdash_{\mathsf{PS}} M:A \quad \Gamma; \langle M/x\rangle B \vdash_{\mathsf{PS}} l:C}{\Gamma; D \vdash_{\mathsf{PS}} M\cdot l:C}\,\Pi\mathsf{L}$$

$$\frac{C \longrightarrow^* s_3 \quad (s_1, s_2, s_3) \in R \quad \Gamma \vdash_{\mathsf{PS}} A:s_1 \quad \Gamma, (x:A) \vdash_{\mathsf{PS}} B:s_2}{\Gamma \vdash_{\mathsf{PS}} \Pi x^A.B:C}\,\Pi\mathsf{wf}$$

$$\frac{C \longrightarrow^* s' \quad (s, s') \in \mathcal{A}}{\Gamma \vdash_{\mathsf{PS}} s:C}\,\mathsf{sorted} \qquad \frac{(x:A) \in \Gamma \quad \Gamma; A \vdash_{\mathsf{PS}} l:C}{\Gamma \vdash_{\mathsf{PS}} x\ l:C}\,\mathsf{Select}_x$$

$$\frac{C \longrightarrow^* \Pi x^A.B \quad \Gamma, (x:A) \vdash_{\mathsf{PS}} M:B}{\Gamma \vdash_{\mathsf{PS}} \lambda x^A.M:C}\,\Pi\mathsf{R}$$

Figure 9: Rules for Proof-search

Note one small difference from [LDM06]: we do not, in rule $\Pi\mathsf{R}$, require that A be a normal form. As in [LDM06], soundness and completeness hold, but because of this difference, we get *quasi-normal forms* rather than normal forms.

Definition 5.1 (Quasi-normal form)**.** A term (or a list) is a *quasi-normal form* if all its redexes are within type annotations of λ-abstractions, e.g. A in $\lambda x^A.M$.

Notice that, as we are searching for (quasi-)normal forms, there are *no* cut-rules in PS. However, in PTSC even terms in normal form may need instances of the cut-rule in their typing derivation. This is because, in contrast to logics where well-formedness of formulae is pre-supposed (such as first-order logic, where cut is admissible), PTSC checks well-formedness of types. For instance in rule $\Pi\mathsf{L}$ of PTSC a type which is not normalised ($\langle M/x\rangle B$) occurs in the stoup of the third premiss, so cuts might be needed to type it inside the derivation.

We conjecture that if we modify rule $\Pi\mathsf{L}$ by now requiring in the stoup of its third premiss *a normal form* to which $\langle M/x\rangle B$ reduces, then any typable normal form can be typed with a cut-free derivation. However, this would make rule $\Pi\mathsf{L}$ more complicated and, more importantly, we do not need such a conjecture to hold in order to perform proof-search.

In contrast, system PS avoids this problem by obviating such type-checking constraints altogether, because types are the input of proof-search, and should therefore be checked before starting search. This is the spirit of the type-checking proviso in the following soundness theorem.

PS is sound and complete in the following sense:

Theorem 5.2.
(1) *(Soundness) Provided* $\Gamma \vdash A\!:\!s$*, if* $\Gamma \vdash_{\mathsf{PS}} M\!:\!A$ *then* $\Gamma \vdash M\!:\!A$ *and* M *is a quasi-normal form.*
(2) *(Completeness) If* $\Gamma \vdash M\!:\!A$ *and* M *is a quasi-normal form, then we can derive* $\Gamma \vdash_{\mathsf{PS}} M\!:\!A$*.*

Proof. Both proofs are done by induction on typing derivations, with similar statements for list typing. For Soundness, the type-checking proviso is verified every time we need the induction hypothesis. For Completeness, the following lemma is required (and also proved inductively): given $A \longleftrightarrow^* A'$, $B \longleftrightarrow^* B'$ and $C \longleftrightarrow^* C'$, if $\Gamma \vdash_{\mathsf{PS}} M : A$ then $\Gamma \vdash_{\mathsf{PS}} M : A'$, and if $\Gamma; B \vdash_{\mathsf{PS}} l\!:\!C$ then $\Gamma; B' \vdash_{\mathsf{PS}} l\!:\!C'$. □

Note that neither part of the theorem relies on the unsolved problem of *expansion postponement* [vBJMP94, Pol98]. Indeed, as indicated above PS *does not* check types. When recovering a full derivation tree from a PS one by the soundness theorem, expansions and cuts might be introduced at any point, arising from the derivation of the type-checking proviso.

Basic proof-search can be done in PS simply by

- reducing the goal, or the type in the stoup;
- depending on its shape, trying to apply one of the inference rules bottom-up; and
- recursively calling the process on the new goals (called *sub-goals*) corresponding to each premiss.

However, some degree of non-determinism is to be expected in proof-search. Such non-determinism is already present in natural deduction, but the sequent calculus version conveniently identifies where it occurs exactly.

There are three potential sources of such non-determinism:

- The choice of a variable x for applying rule Select_x, knowing only Γ and B (this corresponds in natural deduction to the choice of the head-variable of the proof-term). Not every variable of the environment will work, since the type in the stoup will eventually have to be unified with the goal, so we still need backtracking.
- When the goal reduces to a Π-type, there is an overlap between rules $\Pi\mathsf{R}$ and Select_x; similarly, when the type in the stoup reduces to a Π-type, there is an overlap between rules $\Pi\mathsf{L}$ and axiom. Both overlaps disappear when Select_x is restricted to the case when the goal does not reduce to a Π-type (and sequents with stoups never have a goal reducing to a Π-type). This corresponds to looking only for η-long normal forms in natural deduction. This restriction also brings the derivations in LJT (and in our PTSC) closer to the notion of uniform proofs. Further work includes the addition of η to the notion of conversion in PTSC.
- When the goal reduces to a sort s, three rules can be applied (in contrast to the first two points, this source of non-determinism does not already appear in the propositional case).

Such classification is often called *"don't care"* non-determinism in the case of the choice to apply an invertible rule and *"don't know"* non-determinism when the choice identifies a potential backtracking point.

Don't know non-determinism can be in fact quite constrained by the need to eventually unify the stoup with the goal, as an example in Section 7 below illustrates. Indeed, the dependency created by a Π-type forces the searches for proofs of the two premisses of rule $\Pi\mathsf{L}$ to be sequentialised in a way that might prove inefficient: the proof-term produced for the first premiss, selected among others at random, might well lead to the failure to solve the second premiss, leading to endless backtracking.

Hence, there is much to be gained by postponing the search for a proof of the first premiss and trying to solve the second with incomplete inputs. This might not terminate with success or failure but will send back constraints that may be useful in helping to solve the first premiss with the correct proof-term. "Helping" could just be giving some information to orient and speed-up the search for the right proof-term, but it could well define it completely (saving numerous attempts with proof-terms that will lead to failure). Unsurprisingly, these constraints are produced by the axiom rule as *unification* constraints.

In Coq [Coq], the proof-search tactic `apply x` can be decomposed into the bottom-up application of Select_x followed by a series of bottom-up applications of $\Pi\mathsf{L}$ and finally axiom, but it either postpones the solution of sub-goals or automatically solves them from the unification attempt, often avoiding obvious back-tracking.

In the next section we use the framework with meta-variables we have introduced to capture this behaviour in an extended sequent calculus.

6. Using meta-variables for proof-search

We now use the meta-variables in $\mathsf{PTSC}\alpha$ to delay the solution of sub-goals created by the application of rules such as $\Pi\mathsf{L}$. In this way, the extension from PTSC to $\mathsf{PTSC}\alpha$ supports not only an account of tactics such as `apply x` of Coq, but also the specification of algorithms for type inhabitant enumeration and unification. It provides the search-trees that such algorithms have to explore. Our approach has two main novelties in comparison with similar approaches (in the setting of natural deduction) by Dowek [Dow93] and Muñoz [Muñ01].

The first main novelty is that the search-tree is made of the inference rules of sequent calculus and its exploration is merely the root-first construction of a derivation tree; this greatly simplifies the understanding and the description of what such algorithms do.

The second main novelty is the avoidance of the complex phenomenon known as *r-splitting* that features in traditional inhabitation and unification algorithms (e.g. [Dow93]). In natural deduction, lists of arguments are not first-class objects; hence, when choosing a head variable in the construction of a λ-term, one also has to anticipate how many arguments it will be applied to (with polymorphism, there could be infinitely many choices). This anticipation can require a complex analysis of the sorting relations during a single search step and result in an infinitely branching search-tree whose exploration requires interleaving techniques. This is avoided by the use of meta-variables for lists of unknown length, which allows the choice of a head variable without commitment to the number of its arguments.

In contrast to Section 4, where we confined our attention to the *ground* terms of $\mathsf{PTSC}\alpha$ and their relation to the corresponding PTS, here we consider the full language of *open* terms, representing incomplete proofs and partially solved goals. Correspondingly, *(open) environments* are now lists of pairs, denoted $(x : A)$, where x is a variable and A is a (possibly open) term (while *ground environments* only feature ground terms). Ground terms and environments are the eventual targets of successful proof-search, with all meta-variables instantiated. We further consider a new environment Σ that contains the sub-goals that remain to be proved:

Definition 6.1 (Goal environment, constraint, solved constraint, substitution)**.**

- A *goal environment* Σ is a list of:
  - Triples of the form $\Gamma \vdash \alpha : A$, declaring the meta-variable α and called *(term-)goals*, where A is an open term and Γ is an open environment.
  - 4-tuples of the form $\Gamma; B \vdash \beta : A$, declaring the meta-variable β and called *(list-)goals*, where A and B are open terms and Γ is an open environment.
  - Triples of the form $A \stackrel{\Gamma}{=} B$, called *constraints*, where Γ is an open environment and A and B are open terms.

  Goals of a goal environment are required to declare distinct meta-variables.
- A constraint is *solved* if it is of the form $A \stackrel{\Gamma}{=} B$ where A and B are ground and $A \longleftrightarrow^* B$.
- A goal environment is *solved* if it contains no term or list goals and consists only of solved constraints.
- A *substitution* is a finite function σ that maps a meta-variable for term (resp. list), of arity n, to a closed *higher-order* term (resp. list) of arity n, that is to say, a term (resp. list) under a series of n bindings that capture (at least) its free variables (e.g. $x.y.M$ with $\mathsf{FV}(M) \subseteq \{x, y\}$).[7]

  Such a series of bindings can be provided by a typing environment Γ, e.g. $\mathsf{Dom}(\Gamma).M$ (which is a useful notation when e.g. $\Gamma \vdash M : A$).

  As usual, substitutions σ are built up from individual bindings of the form $(\alpha \mapsto x_1 \dots x_n.M)$ by concatenation σ, σ', where bindings in σ' override those in σ.
- The application of a substitution to terms and lists is defined by induction on these. Only the base cases are interesting:

  If $\sigma(\alpha) = x_1 \dots x_n.M$, then $\sigma(\alpha(N_1, \dots, N_n))$ is the x'-normal form[8] of
  $$\langle \sigma(N_1)/x_1 \rangle \dots \langle \sigma(N_n)/x_n \rangle M$$
  (with the usual capture-avoiding conditions).

  Similarly, if $\sigma(\beta) = x_1 \dots x_n.l$, then $\sigma(\beta(N_1, \dots, N_n))$ is the x'-normal form of
  $$\langle \sigma(N_1)/x_1 \rangle \dots \langle \sigma(N_n)/x_n \rangle l$$
  The application of a substitution to an environment is the straightforward extension of the above.

For instance on the example of Section 1.1, for an *actual* term M with $\mathsf{FV}(M) = \{x, y\}$ and $\sigma(\alpha) = x.y.M$, we have that $\sigma(\alpha(N, P))$ is the x'-normal form of $\langle \sigma(N)/x \rangle \langle \sigma(P)/y \rangle M$.

The reason why we x'-normalise the instantiation of meta-variables is that if M is already x'-normal then $(\alpha \mapsto x_1 \dots x_n.M)(\alpha(y_1\ [], \dots, y_n\ []))$ really *is* a renaming of M (and also an x'-normal form). This ensures that only normal forms are output by our system for proof-search, which we can more easily relate to PS.

We now introduce this system, called PE for *Proof Enumeration*, which can be seen as an extension of PS to open terms.

Definition 6.2 (An inference system PE for proof enumeration)**.**
The inference rules for system PE, in Fig. 10, manipulate three kinds of statement:

- The first two are of the form $\Gamma \vdash M : A \mid \Sigma$ and $\Gamma; B \vdash l : C \mid \Sigma$.

[7] This uses a standard notation that can be found in e.g. [Ter03], Ch. 11.

[8] Which exists because x' is convergent even on untyped terms, by Corollary 1.3.

$$\frac{\Gamma = x_1 : A_1, \ldots, x_n : A_n}{\Gamma; D \vdash_{\mathsf{PE}} \beta(x_1\ [], \ldots, x_n\ []) : C \mid (\Gamma; D \vdash\ \beta : C)}\ \mathsf{Claim}_\beta$$

$$\frac{}{\Gamma; D \vdash_{\mathsf{PE}} [] : C \mid D \stackrel{\Gamma}{=} C}\ \mathsf{axiom}$$

$$\frac{D \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma_1 \quad \Gamma; \langle M/x \rangle B \vdash_{\mathsf{PE}} l : C \mid \Sigma_2}{\Gamma; D \vdash_{\mathsf{PE}} M \cdot l : C \mid \Sigma_1, \Sigma_2}\ \Pi\mathsf{L}$$

$$\frac{\Gamma = x_1 : A_1, \ldots, x_n : A_n}{\Gamma \vdash_{\mathsf{PE}} \alpha(x_1\ [], \ldots, x_n\ []) : C \mid (\Gamma \vdash\ \alpha : C)}\ \mathsf{Claim}_\alpha$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s \quad (s', s) \in \mathcal{A}}{\Gamma \vdash_{\mathsf{PE}} s' : C \mid \emptyset}\ \mathsf{sorted}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s \quad (s_1, s_2, s) \in \mathcal{R} \quad \Gamma \vdash_{\mathsf{PE}} A : s_1 \mid \Sigma_1 \quad \Gamma, x : A \vdash_{\mathsf{PE}} B : s_2 \mid \Sigma_2}{\Gamma \vdash_{\mathsf{PE}} \Pi x^A.B : C \mid \Sigma_1, \Sigma_2}\ \Pi\mathsf{wf}$$

$$\frac{(x : A) \in \Gamma \quad \Gamma; A \vdash_{\mathsf{PE}} l : C \mid \Sigma'}{\Gamma \vdash_{\mathsf{PE}} x\ l : C \mid \Sigma'}\ \mathsf{Select}_x$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma, x : A \vdash_{\mathsf{PE}} M : B \mid \Sigma'}{\Gamma \vdash_{\mathsf{PE}} \lambda x^A.M : C \mid \Sigma'}\ \Pi\mathsf{R}$$

$$\frac{\Gamma; B \vdash_{\mathsf{PE}} l : C \mid \Sigma'' \qquad \Sigma, \Sigma'', (\beta \mapsto \mathsf{Dom}(\Gamma).l)(\Sigma') \Longrightarrow_{\mathsf{PE}} \sigma_\Sigma, \sigma_{\Sigma''}, \sigma_{\Sigma'}}{\Sigma, (\Gamma; B \vdash\ \beta : C), \Sigma' \Longrightarrow_{\mathsf{PE}} \sigma_\Sigma, (\beta \mapsto \mathsf{Dom}(\Gamma).(\sigma_\Sigma, \sigma_{\Sigma''})(l)), \sigma_{\Sigma'}}\ \mathsf{Solve}_\beta$$

$$\frac{\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma'' \qquad \Sigma, \Sigma'', (\alpha \mapsto \mathsf{Dom}(\Gamma).M)(\Sigma') \Longrightarrow_{\mathsf{PE}} \sigma_\Sigma, \sigma_{\Sigma''}, \sigma_{\Sigma'}}{\Sigma, (\Gamma \vdash\ \alpha : A), \Sigma' \Longrightarrow_{\mathsf{PE}} \sigma_\Sigma, (\alpha \mapsto \mathsf{Dom}(\Gamma).(\sigma_\Sigma, \sigma_{\Sigma''})(M)), \sigma_{\Sigma'}}\ \mathsf{Solve}_\alpha$$

$$\frac{\Sigma \text{ is solved}}{\Sigma \Longrightarrow_{\mathsf{PE}} \emptyset}\ \mathsf{Solved}$$

Figure 10: Proof-term enumeration $\vdash_{\mathsf{PE}}$

- The third kind of statement is of the form $\Sigma \Longrightarrow \sigma$, where
  - Σ is a goal environment;
  - σ is a substitution as defined above.

In the bottom part of the figure we use the notational convention that a substitution denoted σ_Σ has the meta-variables of the goal environment Σ as its domain.

Derivability in PE of the three kinds of statement is denoted respectively by $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$, $\Gamma; B \vdash_{\mathsf{PE}} l : C \mid \Sigma$ and $\Sigma \Longrightarrow_{\mathsf{PE}} \sigma$.

The statements $\Gamma \vdash M : A \mid \Sigma$ and $\Gamma; B \vdash l : C \mid \Sigma$ have the same intuitive meaning as the corresponding statements in system PS, but note the extra goal environment Σ, which represents the list of sub-goals and constraints that have been produced by proof-search and that remain to be solved. Thus, the inputs of proof enumeration are Γ and A (and Γ, B and C for the second kind of statement) and the outputs are a term M (or list l) and goal environment Σ. Statements of PS are in fact particular cases of these statements with Σ being always solved.

In contrast, in a statement of the form $\Sigma \Longrightarrow \sigma$, Σ is the list of goals to solve, together with the constraints that the solutions must satisfy. It is the input of proof enumeration and σ is meant to be its solution, i.e. the output.

Now we prove that PE is *sound*. For that we need the following notion:

Definition 6.3 (Solution)**.** We define the property σ *is a solution of a goal environment* Σ, by induction on the length of Σ.

- σ is a solution of $\emptyset$.
- If σ is a solution of Σ and
$$x_1 : \sigma(A_1), \ldots, x_n : \sigma(A_n) \vdash_{\mathsf{PS}} (\sigma(\alpha))(x_1\ [], \ldots, x_n\ []) : \sigma(C)$$
then σ is a solution of $\Sigma, (x_1 : A_1, \ldots, x_n : A_n \vdash \alpha : C)$.
- If σ is a solution of Σ and
$$x_1 : \sigma(A_1), \ldots, x_n : \sigma(A_n); \sigma(D) \vdash_{\mathsf{PS}} (\sigma(\beta))(x_1\ [], \ldots, x_n\ []) : \sigma(C)$$
then σ is a solution of $\Sigma, (x_1 : A_1, \ldots, x_n : A_n; D \vdash \beta : C)$.
- If σ is a solution of Σ and
$$\sigma(D) \longleftrightarrow^* \sigma(C)$$
then σ is a solution of $\Sigma, D \stackrel{\Gamma}{=} C$.

For soundness we also need the following lemma:

Lemma 6.4. *Suppose that $\sigma(M)$ and $\sigma(l)$ are ground.*
(1) *If $M \longrightarrow_{\mathsf{B}x'} N$ then $\sigma(M) \longrightarrow^*_{\mathsf{B}x} \sigma(N)$.*
(2) *If $l \longrightarrow_{\mathsf{B}x'} l'$ then $\sigma(l) \longrightarrow^*_{\mathsf{B}x} \sigma(l')$.*

Proof. By simultaneous induction on the derivation of the reduction step, checking all rules for the base case of root reduction. □

Theorem 6.5 (Soundness)**.** *Suppose σ is a solution of Σ.*
(1) *If $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$ then $\sigma(\Gamma) \vdash_{\mathsf{PS}} \sigma(M) : \sigma(A)$.*
(2) *If $\Gamma; B \vdash_{\mathsf{PE}} l : C \mid \Sigma$ then $\sigma(\Gamma); \sigma(B) \vdash_{\mathsf{PS}} \sigma(l) : \sigma(C)$.*

Proof. By induction on derivations. □

Corollary 6.6. *If $\Sigma \Longrightarrow_{\mathsf{PE}} \sigma$ then σ is a solution of Σ.*

Proof. By induction on the derivation, using Theorem 6.5. □

System PE is *complete* in the following sense:

Theorem 6.7 (Completeness)**.**
(1) *If* $\Gamma \vdash_{\mathsf{PS}} M : A$ *then* $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$ *for some solved* Σ*.*
(2) *If* $\Gamma; B \vdash_{\mathsf{PS}} l : C$ *then* $\Gamma; B \vdash_{\mathsf{PE}} l : C \mid \Sigma$ *for some solved* Σ*.*

Proof. By induction on derivations. The rules of PE generalise those of PS. □

In fact, completeness of the full system PE is not surprising, since it is quite general. In particular, nothing is said about when the process should decide to abandon the current goal and start working on another one. Hence we should be interested in completeness of particular *strategies* dealing with that question. For instance:

- We can view the system PS as supporting the strategy of eagerly solving sub-goals as soon as they are created, never delaying them with the sub-goal environment.
- The algorithm for proof enumeration in [Dow93] would correspond here to the "lazy" strategy that always abandons the sub-goal generated by rule $\Pi\mathsf{L}_{\mathsf{PS}}$, but this in fact enables unification constraints to guide the solution of this sub-goal later, so in that case laziness is probably more efficient than eagerness. This is probably what should be chosen for automated theorem proving.
- Mixtures of the two strategies can also be considered and could be the basis of interactive theorem proving. Indeed in some cases the user's input might be more efficient than the automated algorithm, and rule $\Pi\mathsf{L}_{\mathsf{PS}}$ would be a good place to ask whether the user has any clue to solve the sub-goal (since it could help solving the rest of the unification). If he or she has none, then by default the algorithm might abandon the sub-goal and leave it for later.

  In Coq, the tactic `apply x` does something similar: it tries to automatically solve the sub-goals that interfere with the unification constraint (leaving the other ones for later, visible to the user), but, if unification fails, it is always possible for the user to use the tactic and give explicitly the proof-term to make it work. However, such an input is not provided in proof synthesis mode in Coq and the user really has to give it fully, since the tactic will fail if unification fails. In PE, the unification constraint can remain partially solved.

All these behaviours can be simulated in PE, which is therefore a useful framework for the study of proof-search strategies in type theory and for comparison with the work of Jojgov [GJ02], McBride [McB00] and Delahaye [Del01].

7. Example: commutativity of conjunction

We now give an example of proof-search (first introduced in [LDM06] without using meta-variables) in the PTSC equivalent to System F, i.e. the one given by the sets:

$$\mathcal{S} = \{\star, \square\},\ \mathcal{A} = \{(\star, \square)\},\ \text{and}\ \mathcal{R} = \{(\star,\star), (\square,\star)\}$$

For brevity, we omit types on λ-abstractions, abbreviate $x\ []$ as x for any variable x and simplify $\langle N/x \rangle P$ to P when $x \notin \mathsf{FV}(P)$. We also write $A \wedge B$ in place of its System F representation as $\Pi Q^{\star}.(A {\rightarrow} (B {\rightarrow} Q)) {\rightarrow} Q$.

Proof-search in system PS would result in the following derivation:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi_B}{\Gamma \vdash_{\mathsf{PS}} N_B : B} \qquad \dfrac{\dfrac{\pi_A}{\Gamma \vdash_{\mathsf{PS}} N_A : A} \qquad \dfrac{}{\Gamma; Q \vdash_{\mathsf{PS}} [] : Q}\,\mathsf{axiom}}{\Gamma; A \to Q \vdash_{\mathsf{PS}} N_A \cdot [] : Q}\,\Pi\mathsf{L}}{\Gamma; B \to (A \to Q) \vdash_{\mathsf{PS}} N_B \cdot N_A \cdot [] : Q}\,\Pi\mathsf{L}}{\Gamma \vdash_{\mathsf{PS}} y\ N_B \cdot N_A \cdot [] : Q}\,\mathsf{Select}_y}{A : \star, B : \star \vdash_{\mathsf{PS}} \lambda x.\lambda Q.\lambda y.y\ N_B \cdot N_A \cdot [] : (A \wedge B) \to (B \wedge A)}\,\Pi\mathsf{R}
$$

where $\Gamma = A : \star, B : \star, x : A \wedge B, Q : \star, y : B \to (A \to Q)$, and π_A is the following derivation ($N_A = x\ A \cdot (\lambda x'.\lambda y'.x') \cdot []$):

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma; \star \vdash_{\mathsf{PS}} [] : \star}\,\mathsf{axiom}}{\Gamma \vdash_{\mathsf{PS}} A : \star}\,\mathsf{Select}_A \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x' : A, y' : B; A \vdash_{\mathsf{PS}} [] : A}\,\mathsf{axiom}}{\Gamma, x' : A, y' : B \vdash_{\mathsf{PS}} x' : A}\,\mathsf{Select}_{x'}}{\Gamma \vdash_{\mathsf{PS}} \lambda x'.\lambda y'.x' : A \to (B \to A)}\,\Pi\mathsf{R} \qquad \dfrac{}{\Gamma; A \vdash_{\mathsf{PS}} [] : A}\,\mathsf{axiom}}{\Gamma; (A \to (B \to A)) \to A \vdash_{\mathsf{PS}} (\lambda x'.\lambda y'.x') \cdot [] : A}\,\Pi\mathsf{L}}{\Gamma; A \wedge B \vdash_{\mathsf{PS}} A \cdot (\lambda x'.\lambda y'.x') \cdot [] : A}\,\Pi\mathsf{L}}{\Gamma \vdash_{\mathsf{PS}} x\ A \cdot (\lambda x'.\lambda y'.x') \cdot [] : A}\,\mathsf{Select}_x
$$

Similarly, π_B has a derivation ($N_B = x\ B \cdot (\lambda x'.\lambda y'.y') \cdot []$) with an analogous conclusion $\Gamma \vdash_{\mathsf{PS}} x\ B \cdot (\lambda x'.\lambda y'.y') \cdot [] : B$.

We now reconsider the above example in the light of system PE. It illustrates the need for delaying the search for a proof of the first premiss of rule $\Pi\mathsf{L}$. Let

$$
\begin{array}{ll}
\Gamma & = A : \star, B : \star, x : A \wedge B, Q : \star, y : B \to A \to Q \\
\alpha_A(\Gamma) & = \alpha_A(A, B, x, Q, y) \\
\alpha_B(\Gamma) & = \alpha_B(A, B, x, Q, y) \\
M' & = \lambda x.\lambda Q.\lambda y.y\ \alpha_B(\Gamma) \cdot \alpha_A(\Gamma) \cdot [] \\
\Sigma & = (\Gamma \vdash \alpha_B : B), (\Gamma \vdash \alpha_A : A), (Q \stackrel{\Gamma}{=} Q)
\end{array}
$$

We get the PE-derivation below:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \alpha_B(\Gamma) : B \mid (\Gamma \vdash \alpha_B : B)} \qquad \dfrac{\dfrac{}{\Gamma \vdash \alpha_A(\Gamma) : A \mid (\Gamma \vdash \alpha_A : A)} \qquad \dfrac{}{\Gamma; Q \vdash [] : Q \mid (Q \stackrel{\Gamma}{=} Q)}}{\Gamma; A \to Q \vdash \alpha_A(\Gamma) \cdot [] : Q \mid (\Gamma \vdash \alpha_A : A), (Q \stackrel{\Gamma}{=} Q)}}{\Gamma; B \to A \to Q \vdash \alpha_B(\Gamma) \cdot \alpha_A(\Gamma) \cdot [] : Q \mid \Sigma}}{\Gamma \vdash y\ \alpha_B(\Gamma) \cdot \alpha_A(\Gamma) \cdot [] : Q \mid \Sigma}}{A : \star, B : \star \vdash M' : (A \wedge B) \to (B \wedge A) \mid \Sigma} \qquad \dfrac{\dots}{\Sigma \Longrightarrow \sigma_\Sigma}}{(A : \star, B : \star \vdash \alpha : (A \wedge B) \to (B \wedge A)) \Longrightarrow (\alpha \mapsto \sigma_\Sigma(M'))}
$$

where $\sigma_\Sigma = (\alpha_B \mapsto \mathsf{Dom}(\Gamma).N_B, \alpha_A \mapsto \mathsf{Dom}(\Gamma).N_A)$ is the solution to be obtained from the right premiss.

In the above derivation, we have systematically abandoned the sub-goals and recorded them for later. The only choice we made was that of the head-variable y, because it led to the production of the (solved) unification constraint $(Q \stackrel{\Gamma}{=} Q)$.

We now continue the proof-search with the right premiss, solving the two sub-goals $(\Gamma \vdash \alpha_B : B)$ and $(\Gamma \vdash \alpha_A : A)$ that have been delayed. For instance, we can now decide to solve $(\Gamma \vdash \alpha_A : A)$, which will eventually produce the binding $\alpha_A \mapsto \mathsf{Dom}(\Gamma).N_A$ with $N_A = x\ A \cdot (\lambda x'y'.x') \cdot []$, as follows:

$$\dfrac{\dfrac{}{\Gamma \vdash \alpha_1(\Gamma) : \star \mid \Sigma_1} \quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash \alpha_1'(\Gamma') : \alpha_1(\Gamma) \mid \Sigma_1'}}{\Gamma \vdash \lambda x'y'.\alpha_1'(\Gamma') : A \to B \to \alpha_1(\Gamma) \mid \Sigma_1'} \quad \dfrac{}{\Gamma; \alpha_1(\Gamma) \vdash [] : A \mid \Sigma_1''}}{\Gamma; (A \to B \to \alpha_1(\Gamma)) \to \alpha_1(\Gamma) \vdash (\lambda x'y'.\alpha_1'(\Gamma')) \cdot [] : A \mid \Sigma_1', \Sigma_1''}}{\dfrac{\dfrac{\Gamma; A \wedge B \vdash \alpha_1(\Gamma) \cdot (\lambda x'y'.\alpha_1'(\Gamma')) \cdot [] : A \mid \Sigma_1, \Sigma_1', \Sigma_1''}{\Gamma \vdash x\ \alpha_1(\Gamma) \cdot (\lambda x'y'.\alpha_1'(\Gamma')) \cdot [] : A \mid \Sigma_1, \Sigma_1', \Sigma_1''} \qquad D}{\Sigma \Longrightarrow (\alpha_B \mapsto \mathsf{Dom}(\Gamma).N_B, \alpha_A \mapsto \mathsf{Dom}(\Gamma).x\ A \cdot (\lambda x'y'.x') \cdot [])}}$$

where

$$\begin{array}{ll}
\alpha_1(\Gamma) & = \alpha_1(A, B, x, Q, y) \\
\Sigma_1 & = (\Gamma \vdash \alpha_1 : \star) \\
\Gamma' & = \Gamma, x' : A, y' : B \\
\alpha_1'(\Gamma) & = \alpha_1'(A, B, x, Q, y, x', y') \\
\Sigma_1' & = (\Gamma' \vdash \alpha_1' : \alpha_1(\Gamma)) \\
\Sigma_1'' & = (\alpha_1(\Gamma) \stackrel{\Gamma}{=} A) \\
\sigma & = (\alpha_B \mapsto \mathsf{Dom}(\Gamma).N_B, \quad \alpha_1 \mapsto \mathsf{Dom}(\Gamma).A, \quad \alpha_1' \mapsto \mathsf{Dom}(\Gamma').x')
\end{array}$$

and D is a sub-derivation whose conclusion is as follows:

$$\dfrac{\cdots}{(\Gamma \vdash \alpha_B : B), \Sigma_1, \Sigma_1', \Sigma_1'', (Q \stackrel{\Gamma}{=} Q) \Longrightarrow \sigma}$$

In the above derivation, we have also abandoned the generated sub-goals. Again we made one committing choice: that of the head-variable x, which led to the unification constraint $\alpha_1(\Gamma) \stackrel{\Gamma}{=} A$. Any other choice of head-variable would have led to a unification constraint with no solution. Here, this fact (and the subsequent choice of x) can be mechanically noticed by a simple syntactic check.

We now continue the proof-search with the right premiss. We can decide to solve $(\Gamma \vdash \alpha_B : B)$, $(\Gamma \vdash \alpha_1 : \star)$, or $(\Gamma' \vdash \alpha_1' : \alpha_1(\Gamma))$. The order in which we solve $(\Gamma \vdash \alpha_B : B)$ has little importance (the structure is similar to that of the derivation above), but clearly we cannot solve $(\Gamma' \vdash \alpha_1' : \alpha_1(\Gamma))$ before we know $\alpha_1(\Gamma)$. Hence, we need to solve $(\Gamma \vdash \alpha_1 : \star)$ first, which will produce $\alpha_1 \mapsto \mathsf{Dom}(\Gamma).A$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma; \star \vdash [] : \star \mid \star \stackrel{\Gamma}{=} \star}}{\Gamma \vdash A\ [] : \star \mid \star \stackrel{\Gamma}{=} \star} \quad \dfrac{\cdots}{(\Gamma \vdash \alpha_B : B), (\star \stackrel{\Gamma}{=} \star), (\Gamma' \vdash \alpha_1' : A), (A \stackrel{\Gamma}{=} A), (Q \stackrel{\Gamma}{=} Q) \Longrightarrow \sigma'}}{(\Gamma \vdash \alpha_B : B), (\Gamma \vdash \alpha_1 : \star), (\Gamma' \vdash \alpha_1' : \alpha_1(\Gamma)), (\alpha_1(\Gamma) \stackrel{\Gamma}{=} A), (Q \stackrel{\Gamma}{=} Q) \Longrightarrow \sigma}$$

where $\sigma' = (\alpha_B \mapsto \mathsf{Dom}(\Gamma).N_B, \quad \alpha'_1 \mapsto \mathsf{Dom}(\Gamma').x')$.

In this derivation we had to inhabit $\star$. This is a fundamental step of the proof, even when expressed with ground terms (in system PS) as above. Here, having delayed the solution of sub-goals, we are now able to infer the correct inhabitation, directly from the unification constraint $(\alpha_1(\Gamma) \stackrel{\Gamma}{=} A)$ which we have generated previously. Our delaying mechanism thus avoids many situations in which the correct choice for inhabiting a type has to be guessed in advance, anticipating the implicit constraints that such a choice will have to satisfy at some point. This is hardly mechanisable and thus leads to numerous backtrackings.

Finally we proceed to the right premiss by solving $(\Gamma' \vdash \alpha'_1 : A)$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma'; A \vdash [] : A \mid A \stackrel{\Gamma'}{=} A}}{\Gamma' \vdash x'\ [] : A \mid A \stackrel{\Gamma'}{=} A} \qquad \dfrac{\dots}{(\Gamma \vdash \alpha_B : B), (\star \stackrel{\Gamma}{=} \star), (A \stackrel{\Gamma'}{=} A), (A \stackrel{\Gamma}{=} A), (Q \stackrel{\Gamma}{=} Q) \Longrightarrow (\alpha_B(\Gamma) \mapsto N_B)}}{(\Gamma \vdash \alpha_B : B), (\star \stackrel{\Gamma}{=} \star), (\Gamma' \vdash \alpha'_1 : A), (A \stackrel{\Gamma}{=} A), (Q \stackrel{\Gamma}{=} Q) \Longrightarrow \sigma'}$$

In this derivation we had to inhabit A. Again we made one committing choice: that of the head-variable x', which led to the unification constraint $A \stackrel{\Gamma'}{=} A$. Again, any other choice of head-variable would have led to obvious failure, a fact which can be mechanically noticed by a simple syntactic check.

We can then proceed with $(\Gamma \vdash \alpha_B : B)$, in a way very similar to that for $(\Gamma \vdash \alpha_A : A)$. We get eventually $N_B = x\ B \cdot (\lambda x'y'.y') \cdot []$.

Putting it all together, we have used system PE to produce the following proof of the commutativity of conjunction:

$$A : \star, B : \star \vdash \lambda xQy.y\ (x\ B \cdot (\lambda x'y'.y') \cdot []) \cdot (x\ A \cdot (\lambda x'y'.x') \cdot []) \cdot [] : (A \wedge B) \rightarrow (B \wedge A)$$

The system has mechanically inferred the relevant choices of the head-variables structuring the proof-term, by finite checks and using the unification constraints generated by delaying the solution of sub-goals.

Conclusion and Further Work

In this paper we have developed a framework that serves as a good theoretical basis for proof-search in type theory.

Proof-search tactics in natural deduction depart from the simple bottom-up application of the typing rules; thus their readability and usage become more complex, as illustrated in proof-assistants such as Coq. Just as in propositional logic [DP99a], permutation-free sequent calculi can be a useful theoretical approach to study and design such tactics, in the hope of improving semi-automated reasoning.

Following these ideas, we have defined a parameterised formalism giving a sequent calculus for each PTS. It comprises a syntax, a rewrite system and typing rules. In contrast to previous work, the syntax of both types and proof-terms of PTSCα is in sequent calculus style, thus avoiding implicit or explicit conversions to natural deduction [GR03c, PD98]. We have given a direct proof, by simulation, of confluence for each PTSCα.

We have established a strong correspondence with natural deduction (regarding both logic and strong normalisation), when restricted to the ground terms PTSC of a given PTSCα. These results and their proofs were formalised in Coq [Sil09]. We can give as examples the corners of Barendregt's λ-cube, for which we now have an elegant theoretical framework for proof-search: We have shown how to deal with conversion rules so that basic proof-search tactics are simply the root-first application of the typing rules.

These ideas have then been extended, in the calculi $\mathsf{PTSC}\alpha$, by the use of meta-variables to formalise the notion of incomplete proofs, and their theory has been studied. The approach differs from [Muñ01] both in that we use sequent calculus rules, which match proof-search tactics, and in that our system simulates β-reduction.

We have shown that, in particular, the explicit use of meta-variables avoids the phenomenon of *r-splitting* and allows for more flexibility in proof-search, where sub-goals can be tackled in the order that is most suitable for each situation. Such a flexibility avoids some of the need for "guess-work" in proof-search, and formalises some mechanisms of proof-search tactics in proof assistants. This approach has been illustrated by the example of commutativity of conjunction.

Our system does not commit to specific search strategies *a priori*, so that it can be used as a general framework to investigate such strategies, as discussed at the end of Section 6. This could reflect various degrees of user interaction in proof-search.

Ongoing work includes the incorporation of some of these ideas into the redesign of the Coq proof engine [Coq]. It also includes the treatment of η-conversion, a feature that is currently lacking in the PTS-based system Coq. We expect that, by adding η-*expansion* to our system, our approach to proof-search can be related to that of *uniform proofs* in logic programming.

Further work includes studying direct proofs of strong normalisation (such as Kikuchi's for propositional logic [Kik04]), and dealing with inductive types such as those used in Coq. Their specific proof-search tactics should also clearly appear in sequent calculus. Finally, given the importance of sequent calculi for classical logic, it would be interesting to build classical Pure Type Sequent Calculi.

Acknowledgements The authors are grateful to Delia Kesner, Gilles Dowek, Hugo Herbelin, Arnaud Spiwack, Vincent Siles, Alex Simpson and David Pym for their helpful remarks and comments, and for pointing out important items of related work.

References

[And92] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[Bar91] H. P. Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991.

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.

[Ber88] S. Berardi. Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Department of Computer Science, CMU, and Dipartimento di Matematica, Universitá di Torino, 1988.

[BG99] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211(1-2):375–395, 1999.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2–3):95–120, 1988.

[Coq] The Coq Proof Assistant. Available at `http://coq.inria.fr/`

[Daa80] D. v. Daalen. *The Language Theory of Automath*. PhD thesis, Eindhoven University of Technology, 1980. Automath Technical Report AUT-073.

[Del01] D. Delahaye. *Conception de langages pour décrire les preuves et les automatisations dans les outils d'aide à la preuve: une étude dans le cadre du système* Coq. PhD thesis, Université Pierre et Marie Curie (Paris 6), 2001.

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. LKQ and LKT: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Proceedings of the Workshop on Advances in Linear Logic*, volume 222 of *London Math. Society Lecture Note Series*, pages 211–224. Cambridge University Press, 1995.

[Dow93] G. Dowek. A complete proof synthesis method for type systems of the cube. *Journal of Logic and Computation*, 3(3):287–315, 1993.

[DP99a] R. Dyckhoff and L. Pinto. Proof search in constructive logics. In *Sets and proofs (Leeds, 1997)*, pages 53–65. Cambridge University Press, 1999.

[DP99b] R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212(1–2):141–155, 1999.

[DU03] R. Dyckhoff and C. Urban. Strong normalization of Herbelin's explicit substitution calculus with substitution propagation. *Journal of Logic and Computation*, 13(5):689–706, 2003.

[Gen35] G. Gentzen. Investigations into logical deduction. In *Gentzen collected works*, pages 68–131. Ed M. E. Szabo, North Holland, (1969), 1935.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, 1972.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[GJ02] H. Geuvers and G. I. Jojgov. Open proofs and open terms: A basis for interactive logic. In J. C. Bradfield, editor, *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic (CSL'02)*, volume 2471 of *Lecture Notes in Computer Science*, pages 537–552. Springer-Verlag, 2002.

[GR03a] F. Gutiérrez and B. C. Ruiz. A cut-free sequent calculus for Pure Type Systems verifying the structural rules of Gentzen/Kleene. In M. Leuschel, editor, *Revised Selected Papers from the 12th International Workshop on Logic Based Program Synthesis and Transformation*, volume 2664 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, 2003.

[GR03b] F. Gutiérrez and B. C. Ruiz. Expansion postponement via cut elimination in sequent calculi for Pure Type Systems. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of *Lecture Notes in Computer Science*, pages 956–968. Springer-Verlag, 2003.

[GR03c] F. Gutiérrez and B. Ruiz. Cut elimination in a class of sequent calculi for Pure Type Systems. In R. de Queiroz, E. Pimentel, and L. Figueiredo, editors, *Proceedings of the 10th Workshop on Logic, Language, Information and Computation (WOLLIC'03)*, volume 84 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.

[Her94] H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th International Workshop , CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1994.

[Her95] H. Herbelin. *Séquents qu'on calcule*. Thèse de doctorat, Université Paris 7, 1995.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings of the 2nd Annual IEEE Symposium on Logic in Computer Science (LICS'87)*, pages 194–204. IEEE Computer Society Press, 1987.

[HOL] The HOL system. Available at `http://www.cl.cam.ac.uk/research/hvg/HOL/`

[Hue76] G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \dots, \omega$. Thèse d'état, Université Paris 7, 1976.

[Hue89] G. Huet. The constructive engine. *World Scientific Publishing*, Commemorative Volume for Gift Siromoney, 1989.

[Kha90] Z. Khasidashvili. Expression reduction systems. In *Proceedings of the IN Vekua Institute of Applied Mathematics*, volume 36, 1990.

[Kik04] K. Kikuchi. A direct proof of strong normalization for an extended Herbelin's calculus. In Y. Kameyama and P. J. Stuckey, editors, *Proceedings of the 7th International Symposium on Functional and Logic Programming (FLOPS'04)*, volume 2998 of *Lecture Notes in Computer Science*, pages 244–259. Springer-Verlag, 2004.

[KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. 1980. Handwritten paper, University of Illinois. .

[KL05] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In J. Giesl, editor, *Proceedings of the 16th International Conference on Rewriting Techniques and Applications(RTA'05)*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer-Verlag, 2005.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*, volume 1 of *Bibliotheca Mathematica*. North-Holland, 1952.

[Klo80] J.-W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, 1980. PhD Thesis.

[Kri] J.-L. Krivine. Un interpréteur du λ-calcul. Unpublished note. Available at `http://www.pps.jussieu.fr/~krivine/`

[LDM06] S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory. In Z. Esik, editor, *Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic (CSL'06)*, volume 4207 of *Lecture Notes in Computer Science*, pages 441–455. Springer-Verlag, 2006.

[Len06] S. Lengrand. *Normalisation & Equivalence in Proof Theory & Type Theory*. PhD thesis, Université Paris 7 & University of St Andrews, 2006.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. Technical Report ECS-LFCS-92-211, School of Informatics, University of Edinburgh, 1992. Available at `http://www.dcs.ed.ac.uk/home/lego/html/papers.html`

[McB00] C. McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, Edinburgh University, 2000.

[McK97] J. McKinna. A rational reconstruction of LEGO, 1997. CARG Seminar, Durham University.

[MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Muñ01] C. Muñoz. Proof-term synthesis on dependent-type systems via explicit substitutions. *Theor. Comput. Sci.*, 266(1-2):407–440, 2001.

[PD98] L. Pinto and R. Dyckhoff. Sequent calculi for the normal terms of the $\Lambda\Pi$ and $\Lambda\Pi\Sigma$ calculi. In D. Galmiche, editor, *Proceedings of the CADE-15 Workshop on Proof Search in Type-Theoretic Languages*, volume 17 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998.

[Plo87] G. Plotkin. Towards search spaces for the Edinburgh Logical Framework. In A. Avron, R. Harper, F. Honsell, I. Mason, and G. Plotkin, editors, *Proceedings of the Workshop on General Logic*, pages 169–181. LFCS, Edinburgh University, 1987. ECS-LFCS-88-52.

[Pol98] E. Poll. Expansion Postponement for Normalising Pure Type Systems. *Journal of Functional Programming*, 8(1):89–96, 1998.

[Pra65] D. Prawitz. Natural deduction. a proof-theoretical study. In *Acta Universitatis Stockholmiensis*, volume 3. Almqvist & Wiksell, 1965.

[PW91] D. Pym and L. Wallen. Proof-search in the $\Lambda\Pi$-calculus. In *Logical frameworks*, pages 309–340. Cambridge University Press, 1991.

[Pym95] D. J. Pym. A note on the proof theory of the $\Lambda\Pi$-calculus. *Studia Logica*, 54(2):1992–30, 1995.

[Sil09] V. Silès. Formalisation of pure type sequent calculi, 2009. Available at `http://www.lix.polytechnique.fr/~vsiles/coq/formalisation.html`

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[Ter89] J. Terlouw. Een nadere bewijstheoretische analyse van GSTTs. 1989. Manuscript (in Dutch), University of Nijmegen, The Netherlands.

[vBJMP94] B. van Benthem Jutting, J. McKinna, and R. Pollack. Checking Algorithms for Pure Type Systems. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

Subject Reduction

Definition 1. We write $\Gamma \vdash^{\star} M\!:\!A$ (resp. $\Gamma; B \vdash^{\star} l\!:\!C$) whenever we can derive $\Gamma \vdash M\!:\!A$ (resp. $\Gamma; B \vdash l\!:\!C$) and the last rule is not a conversion rule.

The following Lemma is easily derived by induction on the typing tree:

Lemma 2 (Generation Lemma)**.**

(1) (a) *If* $\Gamma \vdash_{\mathsf{PTSC}} s\!:\!C$ *then there is* s' *such that* $\Gamma \vdash^{\star} s\!:\!s'$ *with* $C \longleftrightarrow^{*} s'$.
  (b) *If* $\Gamma \vdash_{\mathsf{PTSC}} \Pi x^A.B\!:\!C$ *then there is* s *such that* $\Gamma \vdash^{\star} \Pi x^A.B\!:\!s$ *with* $C \longleftrightarrow^{*} s$.
  (c) *If* $\Gamma \vdash_{\mathsf{PTSC}} \lambda x^A.M\!:\!C$ *then*
  there is B *such that* $C \longleftrightarrow^{*} \Pi x^A.B$ *and* $\Gamma \vdash^{\star} \lambda x^A.M\!:\!\Pi x^A.B$.
  (d) *If* $\Gamma \vdash_{\mathsf{PTSC}} \langle M/x\rangle N\!:\!C$ *then there is* C' *such that* $\Gamma \vdash^{\star} \langle M/x\rangle N\!:\!C'$ *with* $C \longleftrightarrow^{*} C'$.
  (e) *If* M *is not of the above forms and* $\Gamma \vdash_{\mathsf{PTSC}} M\!:\!C$, *then* $\Gamma \vdash^{\star} M\!:\!C$.

(2) (a) *If* $\Gamma; B \vdash_{\mathsf{PTSC}} [\,]\!:\!C$ *then* $B \longleftrightarrow^{*} C$.
  (b) *If* $\Gamma; D \vdash_{\mathsf{PTSC}} M\cdot l\!:\!C$ *then*
  there are A, B *such that* $D \longleftrightarrow^{*} \Pi x^A.B$ *and* $\Gamma; \Pi x^A.B \vdash^{\star} M\cdot l\!:\!C$.
  (c) *If* $\Gamma; B \vdash_{\mathsf{PTSC}} \langle M/x\rangle l\!:\!C$ *then are* B', C' *such that*
  $\Gamma; B' \vdash^{\star} \langle M/x\rangle l\!:\!C'$ *with* $C \longleftrightarrow^{*} C'$ *and* $B \longleftrightarrow^{*} B'$.
  (d) *If* l *is not of the above forms and* $\Gamma; D \vdash_{\mathsf{PTSC}} l\!:\!C$ *then* $\Gamma; D \vdash^{\star} l\!:\!C$.

Proof. Straightforward induction on the typing tree. □

Remark 3. The following rule is derivable, using a conversion rule:

$$\frac{\Gamma \vdash_{\mathsf{PTSC}} Q\!:\!A \qquad \Gamma, (x:A), \Delta \vdash_{\mathsf{PTSC}} M\!:\!C \quad \Delta' \vdash_{\mathsf{PTSC}} \langle Q/x\rangle C\!:\!s \quad \Gamma, \langle Q/x\rangle \Delta \sqsubseteq \Delta'}{\Delta' \vdash_{\mathsf{PTSC}} \langle Q/x\rangle M\!:\!\langle Q/x\rangle C}$$

Proving subject reduction relies on the following properties of $\longrightarrow_{\mathsf{Bx}}$:

Lemma 4.

- *Two distinct sorts are not convertible.*
- *A* Π*-construct is not convertible to a sort.*
- $\Pi x^A.B \longleftrightarrow^{*} \Pi x^D.E$ *if and only if* $A \longleftrightarrow^{*} D$ *and* $B \longleftrightarrow^{*} E$.
- *If* $y \notin FV(P)$, *then* $P \longleftrightarrow^{*} \langle N/y\rangle P$.
- $\langle M/y\rangle\langle N/x\rangle P \longleftrightarrow^{*} \langle\langle M/y\rangle N/x\rangle\langle M/y\rangle P$ *(provided* $x \notin FV(M)$*).*

Proof. The first three properties are a consequence of the confluence of the rewrite system (Corollary 2.9). The last two rely on the fact that the system xsubst is terminating, so that only the case when P is an xsubst-normal form remains to be checked, which is done by structural induction. □

Using all of the results above, subject reduction can be proved:

Theorem 5 (Subject reduction in a PTSC)**.**

(1) *If* $\Gamma \vdash_{\mathsf{PTSC}} M\!:\!X$ *and* $M \longrightarrow_{\mathsf{Bx}} M'$, *then* $\Gamma \vdash_{\mathsf{PTSC}} M'\!:\!X$
(2) *If* $\Gamma; Y \vdash_{\mathsf{PTSC}} l\!:\!Z$ *and* $l \longrightarrow_{\mathsf{Bx}} l'$, *then* $\Gamma; Y \vdash_{\mathsf{PTSC}} l'\!:\!Z$

Proof. By simultaneous induction on the typing tree. For every rule, if the reduction takes place within a sub-term that is typed by one of the premisses of the rule (e.g. the conversion rules), then we can apply the induction hypothesis on that premiss. In particular, this takes care of the cases where the last typing rule is a conversion rule.

So it now suffices to look at the root reductions. For lack of space we often do not display some minor premisses in following derivations, but we mention them before or after. We also drop the subscript PTSC from derivable statements.

B $(\lambda x^A.N)\ (P\cdot l_1) \longrightarrow (\langle P/x\rangle N)\ l_1$

By the Generation Lemma, 1.(c) and 2.(b), there exist B, D, E such that:

$$\frac{\dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x^A.N : C} \qquad \dfrac{\Gamma \vdash P : D \quad \Gamma; \langle P/x\rangle E \vdash l_1 : X}{\Gamma; C \vdash P\cdot l_1 : X}}{\Gamma \vdash^\star (\lambda x^A.N)\ (P\cdot l_1) : X}$$

with $\Pi x^A.B \longleftrightarrow^* C \longleftrightarrow^* \Pi x^D.E$. Therefore, $A \longleftrightarrow^* D$ and $B \longleftrightarrow^* E$. Moreover, $\Gamma \vdash A : s_A$, $\Gamma, x : A \vdash B : s_B$ and Γ wf. Hence, we obtain $\Gamma \vdash \langle P/x\rangle B : s_B$, so:

$$\frac{\dfrac{\dfrac{\Gamma \vdash P : D}{\Gamma \vdash P : A} \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \langle P/x\rangle N : \langle P/x\rangle B} \qquad \dfrac{\Gamma; \langle P/x\rangle E \vdash l_1 : X}{\Gamma; \langle P/x\rangle B \vdash l_1 : X}}{\Gamma \vdash (\langle P/x\rangle N\ l_1) : X}$$

with $\langle P/x\rangle B \longleftrightarrow^* \langle P/x\rangle E$.

As A1 $(N\cdot l_1)@l_2 \longrightarrow N\cdot(l_1@l_2)$

By the Generation Lemma 2.(b), there are A and B such that $Y \longleftrightarrow^* \Pi x^A.B$ and:

$$\frac{\dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma \vdash N : A \quad \Gamma; \langle N/x\rangle B \vdash l_1 : C}{\Gamma; Y \vdash N\cdot l_1 : C} \qquad \Gamma; C \vdash l_2 : Z}{\Gamma; Y \vdash^\star (N\cdot l_1)@l_2 : Z}$$

Hence,

$$\frac{\Gamma \vdash Y : s_Y \qquad \dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma \vdash N : A \quad \dfrac{\Gamma; \langle N/x\rangle B \vdash l_1 : C \quad \Gamma; C \vdash l_2 : Z}{\Gamma; \langle N/x\rangle B \vdash l_1@l_2 : Z}}{\Gamma; \Pi x^A.B \vdash N\cdot(l_1@l_2) : Z}}{\Gamma; Y \vdash N\cdot(l_1@l_2) : Z}$$

A2 $[]@l_1 \longrightarrow l_1$

By the Generation Lemma 2.(a), we have $A \longleftrightarrow^* Y$ and

$$\frac{\Gamma; Y \vdash [] : A \quad \Gamma; A \vdash l_1 : Z}{\Gamma; Y \vdash^\star []@l_1 : Z}$$

Since $\Gamma \vdash Y : s_Y$, we obtain

$$\frac{\Gamma; A \vdash l_1 : Z}{\Gamma; Y \vdash l_1 : Z}$$

A3 $(l_1@l_2)@l_3 \longrightarrow l_1@(l_2@l_3)$

By the Generation Lemma 2.(d),

$$\frac{\dfrac{\Gamma; Y \vdash l_1 : B \quad \Gamma; B \vdash l_2 : A}{\Gamma; Y \vdash^\star l_1@l_2 : A} \qquad \Gamma; A \vdash l_3 : Z}{\Gamma; Y \vdash^\star (l_1@l_2)@l_3 : Z}$$

Hence,

$$\dfrac{\Gamma; Y \vdash l_1 : B \quad \dfrac{\Gamma; B \vdash l_2 : A \quad \Gamma; A \vdash l_3 : Z}{\Gamma; B \vdash l_2 @ l_3 : Z}}{\Gamma; Y \vdash l_1 @ (l_2 @ l_3) : Z}$$

Bs B1 $N\ [] \longrightarrow\ N$

$$\dfrac{\Gamma \vdash N : A \quad \Gamma; A \vdash [] : X}{\Gamma \vdash^{\star} N\ [] : X}$$

By the Generation Lemma 2.(a), we have $A \longleftrightarrow^{*} X$.
Since $\Gamma \vdash X : s_X$, we obtain

$$\dfrac{\Gamma \vdash N : A}{\Gamma \vdash N : X}$$

B2 $(x\ l_1)\ l_2 \longrightarrow\ x\ (l_1 @ l')$
By the Generation Lemma 1.(e),

$$\dfrac{\dfrac{\Gamma; A \vdash l_1 : B \quad (x : A) \in \Gamma}{\Gamma \vdash^{\star} x\ l : B} \quad \Gamma; B \vdash l_2 : X}{\Gamma \vdash^{\star} (x\ l_1)\ l_2 : X}$$

Hence,

$$\dfrac{(x : A) \in \Gamma \quad \dfrac{\Gamma; A \vdash l_1 : B \quad \Gamma; B \vdash l_2 : X}{\Gamma; A \vdash l_1 @ l_2 : X}}{\Gamma \vdash x\ (l_1 @ l_2) : X}$$

B3 $(N\ l_1)\ l_2 \longrightarrow\ N\ (l_1 @ l_2)$
By the Generation Lemma 1.(e),

$$\dfrac{\dfrac{\Gamma \vdash N : A \quad \Gamma; A \vdash l_1 : B}{\Gamma \vdash^{\star} N\ l_1 : B} \quad \Gamma; B \vdash l_2 : X}{\Gamma \vdash^{\star} (N\ l_1)\ l_2 : X}$$

Hence,

$$\dfrac{\Gamma \vdash N : A \quad \dfrac{\Gamma; A \vdash l_1 : B \quad \Gamma; B \vdash l_2 : X}{\Gamma; A \vdash l_1 @ l_2 : X}}{\Gamma \vdash N\ (l_1 @ l_2) : X}$$

Cs We have a redex of the form $\langle Q/y \rangle R$ typed by:

$$\dfrac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash R : X' \quad \Delta', \langle Q/y \rangle \Delta \sqsubseteq \Gamma\ \mathsf{wf}}{\Gamma \vdash^{\star} \langle Q/y \rangle R : X}$$

with either $X = X' \in \mathcal{S}$ or $X = \langle Q/y \rangle X'$.
In the latter case, $\Gamma \vdash X : s_X$ for some $s_X \in \mathcal{S}$. We also have $\Gamma\ \mathsf{wf}$.

Let us consider each rule:

C1 $\langle Q/y\rangle\lambda x^A.N \longrightarrow\ \lambda x^{\langle Q/y\rangle A}.\langle Q/y\rangle N$
$R = \lambda x^A.N$
By the Generation Lemma 1.(b), there is s_3 such that $C \longleftrightarrow^* s_3$ and:

$$\dfrac{\dfrac{\Delta', y:E, \Delta \vdash A : s_1 \qquad \Delta', y:E, \Delta, x:A \vdash B : s_2}{\Delta', y:E, \Delta \vdash \Pi x^A.B : C} \qquad \Delta', y:E, \Delta, x:A \vdash N : B}{\Delta', y:E, \Delta \vdash \lambda x^A.N : X'}$$

with $(s_1, s_2, s_3) \in \mathcal{R}$ and $X' \equiv \Pi x^A.B$. Therefore, $X' \notin \mathcal{S}$, and as a consequence $X = \langle Q/y\rangle X' \longleftrightarrow^* \langle Q/y\rangle \Pi x^A.B \longleftrightarrow^* \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B$. We have:

$$\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta \vdash A : s_1}{\Gamma \vdash \langle Q/y\rangle A : s_1}$$

Hence, $\Gamma, x : \langle Q/y\rangle A$ wf and $\Delta', \langle Q/y\rangle\Delta, x : \langle Q/y\rangle A \sqsubseteq \Gamma, x : \langle Q/y\rangle A$, so:

$$\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta, x:A \vdash B : s_2}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_2}$$

so that $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s_3$ and

$$\dfrac{\dfrac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta, x:A \vdash N : B}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle N : \langle Q/y\rangle B}}{\Gamma \vdash \lambda x^{\langle Q/y\rangle A}.\langle Q/y\rangle N : \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B} \qquad X \longleftrightarrow^* \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B}{\Gamma \vdash \lambda x^{\langle Q/y\rangle A}.\langle Q/y\rangle N : X}$$

C2 $\langle Q/y\rangle(y\ l_1) \longrightarrow\ Q\ \langle Q/y\rangle l_1$
$R = y\ l_1$
By the Generation Lemma 1.(e), $\Delta', y : E, \Delta; E \vdash l_1 : X'$. Now notice that $y \notin FV(E)$, so $\langle Q/y\rangle E \longleftrightarrow^* E$ and $\Delta' \vdash E : s_E$. Also, $\Delta' \sqsubseteq \Gamma$, so

$$\dfrac{\dfrac{\Delta' \vdash Q : E}{\Gamma \vdash Q : E} \qquad \dfrac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta; E \vdash l_1 : X'}{\Gamma; \langle Q/y\rangle E \vdash \langle Q/y\rangle l_1 : X} \qquad \dfrac{\Delta' \vdash E : s_E}{\Gamma \vdash E : s_E}}{\Gamma; E \vdash \langle Q/y\rangle l_1 : X}}{\Gamma \vdash Q\ \langle Q/y\rangle l_1 : X}$$

C3 $\langle Q/y\rangle(x\ l_1) \longrightarrow\ x\ \langle Q/y\rangle l_1$
$R = x\ l_1$
By the Generation Lemma 1.(e), $\Delta', y : E, \Delta; A \vdash l_1 : X'$ with $(x : A) \in \Delta', \Delta$. Let B be the type of x in Γ. We have

$$\dfrac{\dfrac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta; A \vdash l_1 : X'}{\Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_1 : X} \qquad \Gamma \vdash B : s_B}{\Gamma; B \vdash \langle Q/y\rangle l_1 : X}}{\Gamma \vdash x\ \langle Q/y\rangle l_1 : X}$$

Indeed, if $x \in \mathsf{Dom}(\Delta)$ then $B \longleftrightarrow^* \langle Q/y\rangle A$, otherwise $B \longleftrightarrow^* A$ with $y \notin FV(A)$, so in each case $B \longleftrightarrow^* \langle Q/y\rangle A$. Besides, Γ wf so $\Gamma \vdash B : s_B$.

C4 $\langle Q/y\rangle(N\ l_1) \longrightarrow\ \langle Q/y\rangle N\ \langle Q/y\rangle l_1$
$R = N\ l_1$
By the Generation Lemma 1.(e),

$$\frac{\Delta', y : E, \Delta \vdash N : A \quad \Delta', y : E, \Delta; A \vdash l_1 : X'}{\Delta', y : E, \Delta \vdash^\star N\ l_1 : X'}$$

Also, we have

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash A : s_A}{\Gamma \vdash \langle Q/y\rangle A : s_A}$$

Hence,

$$\frac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash N : A}{\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A} \qquad \dfrac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta; A \vdash l_1 : X'}{\Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_1 : X}}{\Gamma \vdash \langle Q/y\rangle N\ \langle Q/y\rangle l_1 : X}$$

C5 $\langle Q/y\rangle \Pi x^A.B \longrightarrow\ \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B$
$R = \Pi x^A.B$
By the Generation Lemma 1.(b), there exists s_3 such that $X' \longleftrightarrow^* s_3$ and:

$$\frac{\Delta', y : E, \Delta \vdash A : s_1 \qquad \Delta', y : E, \Delta, x : A \vdash B : s_2}{\Delta', y : E, \Delta \vdash \Pi x^A.B : X'}$$

with $(s_1, s_2, s_3) \in \mathcal{R}$.

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash A : s_1}{\Gamma \vdash \langle Q/y\rangle A : s_1}$$

Hence, $\Gamma, x : \langle Q/y\rangle A$ wf and $\Delta', \langle Q/y\rangle\Delta, x : \langle Q/y\rangle A \sqsubseteq \Gamma, x : \langle Q/y\rangle A$, so we obtain:

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta, x : A \vdash B : s_2}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_2}$$

and hence that $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s_3$.
Now if $X' \in \mathcal{S}$, then $X = X' = s_3$ and we are done.
Otherwise $X = \langle Q/y\rangle X' \longleftrightarrow^* \langle Q/y\rangle s_3 \longleftrightarrow^* s_3$, and we conclude using a conversion rule (because $\Gamma \vdash X : s_X$).

C6 $\langle Q/y\rangle s \longrightarrow\ s$ and $R = s$. By the Generation Lemma 1.(a), we obtain $X' \longleftrightarrow^* s'$ for some s' with $(s, s') \in \mathcal{A}$. Since Γ wf, we obtain $\Gamma \vdash s : s'$. If $X' \in \mathcal{S}$, then $X = X' = s'$ and we are done. Otherwise $X = \langle Q/y\rangle X' \longleftrightarrow^* \langle Q/y\rangle s' \longleftrightarrow^* s'$ and we conclude using a conversion rule (because $\Gamma \vdash X : s_X$).

Ds We have a redex of the form $\langle Q/y\rangle l_1$ typed by:

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta; Y' \vdash l_1 : Z' \quad \Delta', \langle Q/y\rangle\Delta \sqsubseteq \Gamma \text{ wf}}{\Gamma; Y \vdash^\star \langle Q/y\rangle l_1 : Z}$$

with $Z = \langle Q/y\rangle Z'$ and $Y = \langle Q/y\rangle Y'$. We also have Γ wf, $\Gamma \vdash Y : s_Y$ and $\Gamma \vdash Z : s_Z$.
Let us consider each rule:

D1 $\langle Q/y\rangle[] \longrightarrow []$
$l_1 = []$
By the Generation Lemma 2.(a), $Y' \longleftrightarrow^* X'$, so $Y \longleftrightarrow^* X$.

$$\cfrac{\cfrac{\Gamma \vdash Y : s_Y}{\Gamma; Y \vdash [] : Y} \qquad \Gamma \vdash X : s_X}{Y \vdash [] : X}$$

D2 $\langle Q/y\rangle(N \cdot l_2) \longrightarrow (\langle Q/y\rangle N) \cdot (\langle Q/y\rangle l_2)$
$l_1 = N \cdot l_2$
By the Generation Lemma 2.(b), there are A, B such that $Y' \longleftrightarrow^* \Pi x^A.B$ and:

$$\frac{\Delta', y : E, \Delta \vdash \Pi x^A.B : s \quad \Delta', y : E, \Delta \vdash N : A \quad \Delta', y : E, \Delta; \langle N/x\rangle B \vdash l_2 : Z'}{\Delta', y : E, \Delta; \Pi x^A.B \vdash^\star l_1 : Z'}$$

From $\Delta', y : E, \Delta; \langle N/x\rangle B \vdash l_2 : Z'$ we obtain

$$\Gamma; \langle Q/y\rangle\langle N/x\rangle B \vdash \langle Q/y\rangle l_2 : Z$$

From $\Delta', y : E, \Delta \vdash N : A$ we obtain $\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A$.
From $\Delta', y : E, \Delta \vdash \Pi x^A.B : s$ part (b) of the Generation Lemma 1 allows us to conclude $\Delta', y : E, \Delta \vdash A : s_A$ and $\Delta', y : E, \Delta, x : A \vdash B : s_B$. Hence we obtain

$$\frac{\Delta', y : E, \Delta \vdash A : s_A}{\Gamma \vdash \langle Q/y\rangle A : s_A}$$

and thus $\Gamma, x : \langle Q/y\rangle A$ wf and then

$$\frac{\Delta', y : E, \Delta, x : A \vdash B : s_B}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_B}$$

From that we obtain both $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s$ and
$\Gamma \vdash \langle\langle Q/y\rangle N/x\rangle\langle Q/y\rangle B : s_B$.
Note that $\Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B \longleftrightarrow^* \langle Q/y\rangle \Pi x^A.B \longleftrightarrow^* \langle Q/y\rangle Y' = Y$. We obtain

$$\cfrac{\cfrac{\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A \qquad \cfrac{\Gamma; \langle Q/y\rangle\langle N/x\rangle B \vdash \langle Q/y\rangle l_2 : Z}{\Gamma; \langle\langle Q/y\rangle N/x\rangle\langle Q/y\rangle B \vdash \langle Q/y\rangle l_2 : Z}}{\Gamma; \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B \vdash (\langle Q/y\rangle N) \cdot (\langle Q/y\rangle l_2) : Z}}{\Gamma; Y \vdash (\langle Q/y\rangle N) \cdot (\langle Q/y\rangle l_2) : Z}$$

D3 $\langle Q/y\rangle(l_2 @ l_3) \longrightarrow (\langle Q/y\rangle l_2) @ (\langle Q/y\rangle l_3)$
$l_1 = l_2 @ l_3$
By the Generation Lemma 2.(d),

$$\frac{\Delta', y : E, \Delta; Y' \vdash l_2 : A \quad \Delta', y : E, \Delta; A \vdash l_3 : Z'}{\Delta', y : E, \Delta; Y' \vdash^\star l_2 @ l_3 : Z'}$$

Hence,

$$\frac{\Gamma; Y \vdash \langle Q/y\rangle l_2 : \langle Q/y\rangle A \qquad \Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_3 : Z}{\Gamma; Y \vdash (\langle Q/y\rangle l_2) @ (\langle Q/y\rangle l_3) : Z}$$

□