

A Sequent Calculus for Type Theory

Stéphane Lengrand^{1,2}, Roy Dyckhoff¹, and James McKinna¹

¹ School of Computer Science, University of St Andrews, Scotland

² PPS, Université Paris 7, France

{sl, rd, james}@dcs.st-and.ac.uk

Abstract. Based on natural deduction, Pure Type Systems (PTS) can express a wide range of type theories. In order to express proof-search in such theories, we introduce the Pure Type Sequent Calculi (PTSC) by enriching a sequent calculus due to Herbelin, adapted to proof-search and strongly related to natural deduction.

PTSC are equipped with a normalisation procedure, adapted from Herbelin's and defined by local rewrite rules as in Cut-elimination, using explicit substitutions. It satisfies Subject Reduction and it is confluent. A PTSC is logically equivalent to its corresponding PTS, and the former is strongly normalising if and only if the latter is.

We show how the conversion rules can be incorporated inside logical rules (as in syntax-directed rules for type checking), so that basic proof-search tactics in type theory are merely the root-first application of our inference rules.

Keywords: Type theory, PTS, sequent calculus, proof-search, strong normalisation.

1 Introduction

In this paper, we apply to the framework of *Pure Type Systems* [Bar92] the insights into the relationship between sequent calculus and natural deduction as developed in previous papers by Herbelin [Her94, Her95], the second author and others [DP99b, PD00, DU03].

In sequent calculus the proof-search space is often the cut-free fragment, since the latter usually satisfies the subformula property. Herbelin's sequent calculus LJT has the extra advantage of being closer to natural deduction, in that it is permutation-free, and it makes proof-search more deterministic than a Gentzen-style sequent calculus. This makes LJT a natural formalism to organise proof-search in intuitionistic logic [DP99a], and, its derivations being close to the notion of uniform proofs, LJT can be used to describe proof-search in pure Prolog and some of its extensions [MNPS91]. The corresponding term assignment system also expresses the intimate details of β -normalisation in λ -calculus in a form closer to abstract (stack-based) machines for reduction (such as Krivine's [Kri]).

The framework of *Pure Type Systems* (PTS) [Bar92] exploits and generalises the Curry-Howard correspondence, and accounts for many systems already existing, starting with *Barendregt's Cube*. Proof assistants based on them, such

as the `Coq` system [Coq] or the `Lego` system [LP92], feature interactive proof construction methods using proof-search tactics. Primitive tactics display an asymmetry between introduction rules and elimination rules of the underlying natural deduction calculus: the tactic `Intro` corresponds to the right-introduction rule for the Π -type (whether in natural deduction or in sequent calculus), but the tactics `Apply` in `Coq` or `Refine` in `Lego` are much closer (in spirit) to the left-introduction of Π -types (as in sequent calculus) than to elimination rules of natural deduction.

Although encodings from natural deduction to sequent calculus and vice-versa have been widely studied [Gen35, Pra65, Zuc74], the representation in sequent calculus of type theories is relatively undeveloped compared to the literature about type theory in natural deduction. An interesting approach to Pure Type Systems using sequent calculus is in [GR03]. Nevertheless, only the typing rules are in a sequent calculus style, whereas the syntax is still in a natural deduction style: in particular, proofs are denoted by λ -terms, the structure of which no longer matches the structure of proofs.

However, proofs in sequent calculus *can* be denoted by terms; for instance, a construction $M \cdot l$, representing a list of terms with head M and tail l , is introduced in [Her94, Her95] to denote the left-introduction of implication (in the sequent calculus LJT):

$$\frac{\Gamma \vdash M : A \quad \Gamma ; B \vdash l : C}{\Gamma ; A \rightarrow B \vdash M \cdot l : C}$$

This approach is extended to the corner of the Cube with dependent types and type constructors in [PD00], but types are still built with λ -terms, so the system extensively uses conversion functions from sequent calculus to natural deduction and back.

With such term assignment systems, cut-elimination can be done by means of a rewrite system, cut-free proofs being thus denoted by terms in normal form. In type theory, not only is the notion of proof-normalisation/cut-elimination interesting on its own, but it is even necessary to define the notion of typability, as soon as types depend on terms.

In this paper we enrich Herbelin's sequent calculus LJT into a collection of systems called *Pure Type Sequent Calculi* (PTSC), capturing the traditional PTS, with the hope to improve the understanding of implementation of proof systems based on PTS in respect of:

- having a direct analysis of the basic tactics, which could then be moved into the kernel, rather than requiring a separate type-checking layer for correctness,
- opening the way to improve the basic system with an approach closer to abstract machines to express reductions, both in type-checking *and* in execution (of extracted programs),
- studying extensions to systems involving inductive types/families (such as the Calculus of Inductive Constructions).

Inspired by the fact that, in type theory, implication and universal quantification are just a dependent product, we modify the inference rule above to obtain the left-introduction rule for a Π -type in a PTSC:

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

We use here explicit substitutions, whose natural typing rule are cuts [BR95]. From our system a version with implicit substitutions can easily be derived, but this does not allow cuts on an arbitrary formula of a typing environment Γ . Also, explicit substitutions allow the definition of a normalisation procedure by local (small-step) rewrite rules in the spirit of Gentzen's cut-elimination.

Derivability of sequents in a PTSC is denoted by \vdash , while derivability in a PTS is denoted by \vdash_{PTS} . We establish the logical equivalence between a PTSC and its corresponding PTS by means of type-preserving encodings. We also prove that the former is strongly normalising if and only if the latter is. The proof is based on mutual encodings that allow the normalisation procedure of one formalism to be simulated by that of the other. Part of the proof also uses a technique by Bloo and Geuvers [BG99], introduced to prove strong normalisation properties of an explicit substitution calculus and later used in [DU03].

In order to show the convenience for proof-search of the sequent calculus approach, we then present a system that is syntax-directed for proof-search, by incorporating the conversion rules into the typing rules that correspond to term constructions. This incorporation is similar to the constructive engine of [Hue89], but different in that proof search takes Γ and A as inputs and produces a (normal) term M such that $\Gamma \vdash M : A$, while the constructive engine takes Γ and M as inputs and produces A . Derivability in the proof-search system is denoted by \vdash_{PS} .

Section 2 presents the syntax of a PTSC and gives the rewrite rules for normalisation. Section 3 gives the typing system with the parameters specifying the PTSC, and a few properties are stated such as Subject Reduction. Section 4 establishes the correspondence between a PTSC and its corresponding PTS, from which we derive confluence. Section 5 presents the strong normalisation result. Section 6 discusses proof-search in a PTSC.

2 Syntax and Operational Semantics of a PTSC

The syntax of a PTSC depends on a given set \mathcal{S} of sorts, written s, s', \dots , and a denumerable set \mathcal{X} of variables, written x, y, z, \dots . The set \mathcal{T} of *terms* (denoted M, N, P, \dots) and the set \mathcal{L} of *lists* (denoted l, l', \dots) are inductively defined as

$$\begin{aligned} M, N, A, B &::= \Pi x^A . B \mid \lambda x^A . M \mid s \mid x \mid l \mid M \mid \langle M/x \rangle N \\ l, l' &::= [] \mid M \cdot l \mid l @ l' \mid \langle M/x \rangle l \end{aligned}$$

$\Pi x^A . M$, $\lambda x^A . M$, 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 terms and lists as well as α -conversion. The set of

free variables of a term M (resp. a list l) is denoted $\text{FV}(M)$ (resp. $\text{FV}(l)$). We use Barendregt's convention that no variable is free and bound in a term in order to avoid variable capture when reducing it. Let $A \rightarrow B$ denote $\Pi x^A.B$ when $x \notin \text{FV}(B)$.

This syntax is an extension of Herbelin's $\bar{\lambda}$ [Her95] (with type annotations on λ -abstractions). Lists are used to represent series of arguments of a function, the terms $x l$ (resp. $M l$) representing the application of x (resp. M) to the list of arguments l . Note that a variable alone is not a term, it has to be applied to a list, possibly the empty list, denoted \square . The list with head M and tail l is denoted $M \cdot l$, with a typing rule corresponding to the left-introduction of Π -types (c.f. Section 3). Successive applications give rise to the concatenation of lists, denoted $l @ l'$, and $\langle M/x \rangle N$ and $\langle M/x \rangle l$ are explicit substitution operators 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, which is adapted from [DU03] and shown in Fig. 1. Side-conditions to avoid variable capture can be inferred from the reduction rules and are ensured by Barendregt's convention. Confluence of the system is proved in section 4. More intuition about Herbelin's calculus, its syntax and operational semantics is given in [Her95].

$$\begin{array}{l}
 \text{B} \quad (\lambda x^A.M) (N \cdot l) \longrightarrow (\langle N/x \rangle M) l \\
 \\
 \left. \begin{array}{l}
 \text{B1} \quad M \square \longrightarrow M \\
 \text{B2} \quad (x l) l' \longrightarrow x (l @ l') \\
 \text{B3} \quad (M l) l' \longrightarrow M (l @ l') \\
 \text{A1} \quad (M \cdot l') @ l \longrightarrow M \cdot (l' @ l) \\
 \text{A2} \quad \square @ l \longrightarrow l \\
 \text{A3} \quad (l @ l') @ l'' \longrightarrow l @ (l' @ l'') \\
 \text{C1} \quad \langle P/y \rangle \lambda x^A.M \longrightarrow \lambda x^{\langle P/y \rangle A} . \langle P/y \rangle M \\
 \text{C2} \quad \langle P/y \rangle (y l) \longrightarrow P \langle P/y \rangle l \\
 \text{C3} \quad \langle P/y \rangle (x l) \longrightarrow x \langle P/y \rangle l \quad \text{if } x \neq y \\
 \text{C4} \quad \langle P/y \rangle (M l) \longrightarrow \langle P/y \rangle M \langle P/y \rangle l \\
 \text{C5} \quad \langle P/y \rangle \Pi x^A.B \longrightarrow \Pi x^{\langle P/y \rangle A} . \langle P/y \rangle B \\
 \text{C6} \quad \langle P/y \rangle s \longrightarrow s \\
 \text{D1} \quad \langle P/y \rangle \square \longrightarrow \square \\
 \text{D2} \quad \langle P/y \rangle (M \cdot l) \longrightarrow (\langle P/y \rangle M) \cdot (\langle P/y \rangle l) \\
 \text{D3} \quad \langle P/y \rangle (l @ l') \longrightarrow (\langle P/y \rangle l) @ (\langle P/y \rangle l')
 \end{array} \right\} \begin{array}{l}
 \times \\
 \text{xsubst:}
 \end{array}
 \end{array}$$

Fig. 1. Reduction Rules

We denote by \longrightarrow_G the contextual closure of the reduction relation defined by any system G of rewrite rules (such as $\text{B}, \text{xsubst}, \times$). 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, \times from Fig. 1.

A simple polynomial interpretation shows that system \times is terminating. If we add rule B , then the system fails to be terminating unless we only consider terms that are typed in a particular typing system.

3 Typing System and Properties

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 5.

Definition 1 (Environments)

- Environments are lists of pairs 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:

$$\begin{aligned} \text{Dom}(\square) &= \emptyset & \text{Dom}(\Gamma, (x : A)) &= \text{Dom}(\Gamma) \cup \{x\} \\ \langle P/y \rangle(\square) &= \square & \langle P/y \rangle(\Gamma, (x : A)) &= \langle P/y \rangle\Gamma, (x : \langle P/y \rangle A) \end{aligned}$$

- We define the following inclusion relation between environments:
 $\Gamma \sqsubseteq \Delta$ if for all $(x : A) \in \Gamma$, there is $(x : B) \in \Delta$ with $A \longleftarrow^* B$

The inference rules in Fig. 2 inductively define the derivability of three kinds of judgement: some of the form $\Gamma \text{ wf}$, some of the form $\Gamma \vdash M : A$ and some of the form $\Gamma; B \vdash l : A$. In the latter case, B is said to be in the *stoup* of the sequent. Side-conditions are used, such as $(s_1, s_2, s_3) \in \mathcal{R}$, $x \notin \text{Dom}(\Gamma)$, $A \longleftarrow^* B$ or $\Delta \sqsubseteq \Gamma$, and we use the abbreviation $\Delta \sqsubseteq \Gamma \text{ wf}$ for $\Delta \sqsubseteq \Gamma$ and $\Gamma \text{ wf}$.

Since the substitution of a variable in an environment affects the rest of the environment (which could depend on the variable), the two rules for explicit substitutions Cut_2 and Cut_4 must have a particular shape that is admittedly complex: thinning (Lemma 3) is built-in by allowing a controlled change of environment. This may appear artificial, but simpler versions that we have tried failed the thinning property. More generally, typing rules for explicit substitutions in type theory are known to be a tricky issue (see for instance [Blo01]), often leading to the failure of subject reduction (Theorem 1). The rules here are sound in that respect, but more elegant alternatives are still to be investigated, possibly by enriching the structure of environments as in [Blo01].

The case analysis for C' in the rule Cut_4 is only necessary for Lemma 1.2 to hold in the presence of top sorts (untyped sorts), and is avoided in [Blo01] by not using explicit substitutions for types in sequents. Here we were appealed by the uniformity of using them everywhere, the use of implicit substitutions for C' and the stoup of the third premiss of III being only a minor variant.

There are three conversion rules conv_r , conv'_r , and conv_l in order to deal with the two kinds of judgements and, for one of them, convert the type in the stoup. The lemmas of this section are proved by induction on typing derivations:

$\frac{}{\boxed{\text{wf}}} \text{empty} \qquad \frac{\Gamma \vdash A:s \quad x \notin \text{Dom}(\Gamma)}{\Gamma, (x:A) \text{ wf}} \text{extend}$
$\frac{\Gamma \vdash A:s}{\Gamma; A \vdash \boxed{\text{wf}}:A} \text{axiom} \qquad \frac{\Gamma \vdash \Pi x^A.B:s \quad \Gamma \vdash M:A \quad \Gamma; \langle M/x \rangle B \vdash l:C}{\Gamma; \Pi x^A.B \vdash M.l:C} \text{PII}$
$\frac{\Gamma; C \vdash l:A \quad \Gamma \vdash B:s \quad A \longleftarrow^* B}{\Gamma; C \vdash l:B} \text{conv}'_r \qquad \frac{\Gamma; A \vdash l:C \quad \Gamma \vdash B:s \quad A \longleftarrow^* B}{\Gamma; B \vdash l:C} \text{conv}'_l$
$\frac{\Gamma; C \vdash l':A \quad \Gamma; A \vdash l:B}{\Gamma; C \vdash l'@l:B} \text{Cut}_1$ $\frac{\Gamma \vdash P:A \quad \Gamma, (x:A), \Delta; B \vdash l:C \quad \Gamma, \langle P/x \rangle \Delta \sqsubseteq \Delta' \text{ wf}}{\Delta'; \langle P/x \rangle B \vdash \langle P/x \rangle l: \langle P/x \rangle C} \text{Cut}_2$
$\frac{\Gamma \text{ wf} \quad (s, s') \in \mathcal{A}}{\Gamma \vdash s:s'} \text{sorted} \qquad \frac{\Gamma \vdash A:s_1 \quad \Gamma, (x:A) \vdash B:s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^A.B:s_3} \text{PIIwf}$
$\frac{\Gamma \vdash \Pi x^A.B:s \quad \Gamma, (x:A) \vdash M:B}{\Gamma \vdash \lambda x^A.M:\Pi x^A.B} \text{PIr} \qquad \frac{\Gamma; A \vdash l:B \quad (x:A) \in \Gamma}{\Gamma \vdash x l:B} \text{Select}_x$ $\frac{\Gamma \vdash M:A \quad \Gamma \vdash B:s \quad A \longleftarrow^* B}{\Gamma \vdash M:B} \text{conv}_r$
$\frac{\Gamma \vdash M:A \quad \Gamma; A \vdash l:B}{\Gamma \vdash M l:B} \text{Cut}_3$ $\frac{\Gamma \vdash P:A \quad \Gamma, (x:A), \Delta \vdash M:C \quad \Gamma, \langle P/x \rangle \Delta \sqsubseteq \Delta' \text{ wf}}{\Delta' \vdash \langle P/x \rangle M:C'} \text{Cut}_4$ <p style="text-align: center; margin-top: 5px;">where either $(C' = C \in \mathcal{S})$ or $C \notin \mathcal{S}$ and $C' = \langle P/x \rangle C$</p>

Fig. 2. Typing rules of a PTSC

Lemma 1 (Properties of typing judgements). *If $\Gamma \vdash M:A$ (resp. $\Gamma; B \vdash l:C$) then $\text{FV}(M) \subseteq \text{Dom}(\Gamma)$ (resp. $\text{FV}(l) \subseteq \text{Dom}(\Gamma)$), and the following judgements 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 1 (Properties of well-formed environments)

1. If $\Gamma, x:A, \Delta$ wf then $\Gamma \vdash A:s$ for some $s \in \mathcal{S}$ with $x \notin \text{Dom}(\Gamma) \cup \text{Dom}(\Delta)$
and $\text{FV}(A) \subseteq \text{Dom}(\Gamma)$ (and in particular $x \notin \text{FV}(A)$)
2. If Γ, Δ wf then Γ wf.

Lemma 2 (Weakening). Suppose Γ, Γ' wf and $\text{Dom}(\Gamma') \cap \text{Dom}(\Delta) = \emptyset$.

1. If $\Gamma, \Delta \vdash M:B$ then $\Gamma, \Gamma', \Delta \vdash M:B$.
2. If $\Gamma, \Delta; C \vdash l:B$, then $\Gamma, \Gamma', \Delta; C \vdash l:B$.
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, change its order, and convert the types inside, as long as it remains well-formed:

Lemma 3 (Thinning). Suppose $\Gamma \sqsubseteq \Gamma'$ wf.

1. If $\Gamma \vdash M:B$ then $\Gamma' \vdash M:B$.
2. If $\Gamma; C \vdash l:B$, then $\Gamma'; C \vdash l:B$.

Using all of the results above, Subject Reduction can be proved (see [LDM]).

Theorem 1 (Subject Reduction in a PTSC)

1. If $\Gamma \vdash M:A$ and $M \longrightarrow M'$, then $\Gamma \vdash M':A$
2. If $\Gamma; A \vdash l:B$ and $l \longrightarrow l'$, then $\Gamma; A \vdash l':B$

4 Correspondence with Pure Type Systems

There is a logical correspondence between a PTSC given by the sets \mathcal{S} , \mathcal{A} and \mathcal{R} and the PTS given by the same sets.

We briefly recall the syntax and semantics of the PTS. The terms have the following syntax:

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

which is equipped with the β -reduction rule $(\lambda x^v.t) u \longrightarrow_\beta t\{x = u\}$, in which the substitution is implicit, i.e. is a meta-operation.

The terms are typed by the typing rules in Fig. 3, which depend on the sets \mathcal{S} , \mathcal{A} and \mathcal{R} . PTS are confluent and satisfy subject reduction and thinning [Bar92].

In order to encode the syntax of a PTS into that of a PTSC, it is convenient to re-express the syntax of a PTS with a grammar closer to that of a PTSC as follows:

$$\begin{aligned} w ::= s \mid \Pi x^T.U \mid \lambda x^T.t \\ t, u, v, T, U, V, \dots ::= w \mid x \vec{t} \mid w u \vec{t} \end{aligned}$$

$\frac{}{\square \text{ wf}}$	$\frac{\Gamma \vdash_{\text{PTS}} T:s \quad x \notin \text{Dom}(\Gamma)}{\Gamma, (x:T) \text{ wf}}$	$\frac{\Gamma \text{ wf} \quad (x:T) \in \Gamma}{\Gamma \vdash_{\text{PTS}} x:T}$
$\frac{\Gamma \text{ wf} \quad (s,s') \in \mathcal{A}}{\Gamma \vdash_{\text{PTS}} s:s'}$	$\frac{\Gamma \vdash_{\text{PTS}} U:s_1 \quad \Gamma, (x:U) \vdash_{\text{PTS}} T:s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\text{PTS}} \Pi x^U.T:s_3}$	
$\frac{\Gamma \vdash_{\text{PTS}} \Pi x^U.V:s \quad \Gamma, (x:U) \vdash_{\text{PTS}} t:V}{\Gamma \vdash_{\text{PTS}} \lambda x^U.t:\Pi x^U.V}$		$\frac{\Gamma \vdash_{\text{PTS}} t:\Pi x^T.U \quad \Gamma \vdash_{\text{PTS}} u:T}{\Gamma \vdash_{\text{PTS}} t u:U\{x=u\}}$
$\frac{\Gamma \vdash_{\text{PTS}} t:U \quad \Gamma \vdash_{\text{PTS}} V:s \quad U \longleftrightarrow_{\beta}^* V}{\Gamma \vdash_{\text{PTS}} t:V}$		

Fig. 3. Typing rules of a PTS

$\begin{aligned} \mathbf{h}_w(s) &= s \\ \mathbf{h}_w(\Pi x^v.v') &= \Pi x^{\mathbf{h}(v)}. \mathbf{h}(v') \\ \mathbf{h}_w(\lambda x^v.t) &= \lambda x^{\mathbf{h}(v)}. \mathbf{h}(t) \\ \mathbf{h}(w) &= \mathbf{h}_w(w) \\ \mathbf{h}(x \vec{t}) &= x \mathbf{h}_l(\vec{t}) \\ \mathbf{h}(w u \vec{t}) &= \mathbf{h}_w(w) (\mathbf{h}(u) \cdot \mathbf{h}_l(\vec{t})) \\ \mathbf{h}_l(\vec{\emptyset}) &= \square \\ \mathbf{h}_l(\vec{u}_1 \dots \vec{u}_i) &= \mathbf{h}(u_1) \cdot \mathbf{h}_l(\vec{u}_2 \dots \vec{u}_i) \end{aligned}$	$\begin{aligned} \mathbf{k}(\Pi x^A.B) &= \Pi x^{\mathbf{k}(A)}. \mathbf{k}(B) \\ \mathbf{k}(\lambda x^A.M) &= \lambda x^{\mathbf{k}(A)}. \mathbf{k}(M) \\ \mathbf{k}(s) &= s \\ \mathbf{k}(x l) &= \mathbf{k}^z(l)\{z=x\} \quad z \text{ fresh} \\ \mathbf{k}(M l) &= \mathbf{k}^z(l)\{z=\mathbf{k}(M)\} \quad z \text{ fresh} \\ \mathbf{k}(\langle P/x \rangle M) &= \mathbf{k}(M)\{x=\mathbf{k}(P)\} \\ \mathbf{k}^y(\square) &= y \\ \mathbf{k}^y(M \cdot l) &= \mathbf{k}^z(l)\{z=y \mathbf{k}(M)\} \quad z \text{ fresh} \\ \mathbf{k}^y(l @ l') &= \mathbf{k}^z(l')\{z=\mathbf{k}^y(l)\} \quad z \text{ fresh} \\ \mathbf{k}^y(\langle P/x \rangle l) &= \mathbf{k}^y(l)\{x=\mathbf{k}(P)\} \end{aligned}$
--	--

From a PTS to a PTSC

From a PTSC to a PTS

Fig. 4. Mutual encodings between a PTS and a PTSC

where \vec{t} represents a list of “ t -terms” of arbitrary length. The grammar is sound and complete with respect to the usual one presented at the beginning of the section, and it has the advantage of isolating redexes in one term construction in a way similar to a PTSC.

Given in the left-hand side of Fig. 4, the encoding into the corresponding PTSC, is threefold: \mathbf{h}_w applies to “ w -terms”, \mathbf{h} applies to “ t -terms”, and \mathbf{h}_l to lists of “ t -terms”. The right-hand side of Fig. 4 shows the encoding from a PTSC to a PTS. We prove the following theorem by induction on t and M :

Theorem 2 (Properties of the encodings)

1. $\mathbf{h}(t)$ is always an x -normal form, and $\langle \mathbf{h}(t)/x \rangle \mathbf{h}(u) \longrightarrow_x^* \mathbf{h}(u\{x=t\})$.
2. $\mathbf{k}(\mathbf{h}(t)) = t$

3. If M is an x -normal form, then $M = h(k(M))$
4. $M \longrightarrow^*_x h(k(M))$
5. If $t \longrightarrow_\beta u$ then $h(t) \longrightarrow^+ h(u)$
6. If $M \longrightarrow N$ then $k(M) \longrightarrow^*_\beta k(N)$

Now we use Theorem 2 to prove the confluence of PTSC and the equivalence of the equational theories.

Corollary 2 (Confluence). \longrightarrow_x and \longrightarrow are confluent.

Proof. We use the technique of simulation: consider two reduction sequences starting from a term in a PTSC. They can be simulated through k by β -reductions, and since a PTS is confluent, we can close the diagram. Now the lower part of the diagram can be simulated through h back in the PTSC, which closes the diagram there as well. The diagrams can be found in [LDM]. \square

Corollary 3 (Equational theories)

- $t \longleftarrow^*_\beta u$ if and only if $h(t) \longleftarrow^* h(u)$
 $M \longleftarrow^* N$ if and only if $k(M) \longleftarrow^*_\beta k(N)$

Regarding typing, we first define the following operations on environments:

$$\begin{aligned} h(\emptyset) &= \emptyset & k(\emptyset) &= \emptyset \\ h(\Gamma, (x : v)) &= h(\Gamma), (x : h(v)) & k(\Gamma, (x : A)) &= k(\Gamma), (x : k(A)) \end{aligned}$$

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

Theorem 3 (Preservation of typing 1)

1. If $\Gamma \vdash_{PTS} t : T$ then $h(\Gamma) \vdash h(t) : h(T)$
2. If $(\Gamma \vdash_{PTS} t_i : T_i \{x_1 = t_1\} \dots \{x_{i-1} = t_{i-1}\})_{i=1 \dots n}$
and $h(\Gamma) \vdash h(\Pi x_1^{T_1} \dots \Pi x_n^{T_n}. T) : s$
then $h(\Gamma); h(\Pi x_1^{T_1} \dots \Pi x_n^{T_n}. T) \vdash h_l(t_1 \dots t_n) : h(T \{x_1 = t_1\} \dots \{x_n = t_n\})$
3. If Γ wf then $h(\Gamma)$ wf

Theorem 4 (Preservation of typing 2)

1. If $\Gamma \vdash M : A$ then $k(\Gamma) \vdash_{PTS} k(M) : k(A)$
2. If $\Gamma; B \vdash l : A$ then $k(\Gamma), y : k(B) \vdash_{PTS} k^y(l) : k(A)$ for a fresh y
3. If Γ wf then $k(\Gamma)$ wf

5 Equivalence of Strong Normalisation

Theorem 5. A PTSC given by the sets \mathcal{S} , \mathcal{A} , and \mathcal{R} is strongly normalising if and only if the 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. $\Gamma \vdash_{PTS} t : T$ for some Γ, T . By Theorem 3, $h(\Gamma) \vdash h(t) : h(T)$ so $h(t) \in \text{SN}$. Now by Theorem 2, any reduction sequence starting from t maps to a reduction sequence of at least the same length starting from $h(t)$, but those are finite.

Now assume that the PTS is strongly normalising and that $\Gamma \vdash M : A$ in the corresponding PTSC. We shall now apply Bloo and Geuvers' technique from [BG99]. 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, for any such P (resp. l), $k(P)$ (resp. $k^y(l)$) is typable in the PTS, so it is strongly normalising by assumption and we denote by $\sharp k(P)$ (resp. $\sharp k^y(l)$) the length of the longest β -reduction sequence reducing it.

We now encode any such P and l into a first-order syntax given by the following ordered infinite signature:

$$\star \prec i(_) \prec ii(_, _) \prec cut^n(_, _) \prec sub^n(_, _)$$

for all integers n . Moreover, we set $sub^n(_, _) \prec cut^m(_, _)$ if $n < m$. The order is well-founded, and the *lexicographic path ordering* (lpo) that it induces on the first-order terms is also well-founded (definitions and results can be found in [KL80]). The encoding is given in Fig 5. An induction on terms shows that reduction decreases the lpo. \square

$\mathcal{T}(s)$	$= \star$	
$\mathcal{T}(\lambda x^A.M)$	$= \mathcal{T}(\Pi x^A.M) = ii(\mathcal{T}(A), \mathcal{T}(M))$	
$\mathcal{T}(x\ l)$	$= i(\mathcal{T}(l))$	
$\mathcal{T}(M\ l)$	$= cut^{\sharp k(M\ l)}(\mathcal{T}(M), \mathcal{T}(l))$	
$\mathcal{T}(\langle M/x \rangle N)$	$= sub^{\sharp k(\langle M/x \rangle N)}(\mathcal{T}(M), \mathcal{T}(N))$	
$\mathcal{T}(\llbracket \rrbracket)$	$= \star$	
$\mathcal{T}(M.l)$	$= ii(\mathcal{T}(M), \mathcal{T}(l))$	
$\mathcal{T}(l@l')$	$= ii(\mathcal{T}(l), \mathcal{T}(l'))$	
$\mathcal{T}(\langle M/x \rangle l)$	$= sub^{\sharp k^y(\langle M/x \rangle l)}(\mathcal{T}(M), \mathcal{T}(l))$	where y is fresh

Fig. 5. First-order encoding

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

6 Proof-Search

In contrast to propositional logic where cut is an admissible rule of sequent calculus, terms in normal form may need a cut-rule in their typing derivation. For instance in the rule *III*, a type which is not normalised ($\langle M/x \rangle B$) must appear in the stoup of the third premiss, so that cuts might be needed to type it inside the derivation. However if we modify *III* by now using an *implicit* substitution $B\{x = M\}$, normal forms can then be typed not using Cut_2 and Cut_4 , but still using Cut_1 and Cut_3 .

In this section we present a system for proof-search that avoids all cuts, is complete and is sound provided that types are checked independently. In proof-search, the inputs are an environment Γ and a type A , henceforth called *goal*, and the output is a term M such that $\Gamma \vdash M : A$. When we look for a list, the type in the stoup is also an input. The inference rules now need to be directed by the shape of the goal (or of the type in the stoup), and the proof-search system (PS, for short) can be obtained by optimising the use of the conversion rules as shown in Fig. 6. The incorporation of the conversion rules into the other rules is similar to that of the constructive engine in natural deduction [Hue89, JMP94]; however the latter was designed for type synthesis, for which the inputs and outputs are not the same as in proof-search, as mentioned in the introduction.

$\frac{A \longleftrightarrow^* A' \quad \text{axiom}_{\text{PS}} \quad \frac{D \longrightarrow^* \Pi x^A.B \quad \Gamma \vdash_{\text{PS}} M : A \quad \Gamma; \langle M/x \rangle B \vdash_{\text{PS}} l : C}{\Gamma; D \vdash_{\text{PS}} M \cdot l : C} \text{IIl}_{\text{PS}}}{\Gamma; A \vdash_{\text{PS}} [] : A'} \text{II}_{\text{PS}}$
$\frac{C \longrightarrow^* s_3 \quad (s_1, s_2, s_3) \in R \quad \Gamma \vdash_{\text{PS}} A : s_1 \quad \Gamma, (x : A) \vdash_{\text{PS}} B : s_2}{\Gamma \vdash_{\text{PS}} \Pi x^A.B : C} \text{IIwf}_{\text{PS}}$
$\frac{C \longrightarrow^* s' \quad (s, s') \in \mathcal{A}}{\Gamma \vdash_{\text{PS}} s : C} \text{sorted}_{\text{PS}} \quad \frac{(x : A) \in \Gamma \quad \Gamma; A \vdash_{\text{PS}} l : B}{\Gamma \vdash_{\text{PS}} x l : B} \text{Select}_x$
$\frac{C \longrightarrow^* \Pi x^A.B \quad A \text{ is a normal form} \quad \Gamma, (x : A) \vdash_{\text{PS}} M : B}{\Gamma \vdash_{\text{PS}} \lambda x^A.M : C} \text{IIr}_{\text{PS}}$

Fig. 6. Rules for Proof-search

Notice than in PS there are *no* cut-rules. Indeed, even though in the original typing system cuts are required in typing derivations of normal forms, they only occur to check that types are well-typed themselves. Here we removed those type-checking constraints, relaxing the system, because types are the input of proof-search, and they would be checked before starting the search. PS is sound and complete in the following sense:

Theorem 6

1. (Soundness) Provided $\Gamma \vdash A : s$, if $\Gamma \vdash_{\text{PS}} M : A$ then $\Gamma \vdash M : A$ and M is a normal form.
2. (Completeness) If $\Gamma \vdash M : A$ and M is a normal form, then $\Gamma \vdash_{\text{PS}} M : A$.

Proof. Both proofs are done by induction on typing derivations, with similar statements for lists. 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): assuming $A \longleftrightarrow^* A'$ and $B \longleftrightarrow^* B'$, if $\Gamma \vdash_{\text{PS}} M : A$ then $\Gamma \vdash_{\text{PS}} M : A'$, and if $\Gamma; B \vdash_{\text{PS}} l : A$ then $\Gamma; B' \vdash_{\text{PS}} l : A'$.

□

Note that neither part of the theorem relies on the unsolved problem of *expansion postponement* [JMP94, Pol98]. Indeed, 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, coming from the derivation of the type-checking proviso.

The condition that A is in normal form in rule Πr_{PS} is not problematic for completeness: whether or not the PTSC is strongly normalising, such a normal form is given as the type annotation of the λ -abstraction, in the term M of the hypothesis of completeness. On the other hand, the condition allows the soundness theorem to state that all terms typable in system PS are normal forms. Without it, terms would be in normal forms but for their type annotations in λ -abstractions.

Basic proof-search can be done in the proof-search system simply by reducing the goal, or the type in the stoup, and then, depending on its shape, trying to apply one of the inference rules bottom-up.

There are three points of non-determinism in proof-search:

- 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 back-tracking.
- When the goal reduces to a Π -type, there is an overlap between rules Πr_{PS} and Select_x ; similarly, when the type in the stoup reduces to a Π -type, there is an overlap between rules Πl_{PS} and axiom_{PS} . 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).

The non-determinism is already present in natural deduction, but the sequent calculus version conveniently identifies where it occurs exactly.

We now give the example of a derivation in PS. We consider the PTSC equivalent to system F , i.e. the one given by the sets:

$\mathcal{S} = \{\text{Type}, \text{Kind}\}$, $\mathcal{A} = \{(\text{Type}, \text{Kind})\}$, and $\mathcal{R} = \{(\text{Type}, \text{Type}), (\text{Kind}, \text{Type})\}$.

For brevity we omit the types on λ -abstractions, we abbreviate $x \square$ as x for any variable x and simplify $\langle N/x \rangle P$ as P when $x \notin \text{FV}(P)$. We also write $A \wedge B$ for $\Pi Q^{\text{Type}}.(A \rightarrow (B \rightarrow Q)) \rightarrow Q$. Trying to find a term M such that $A : \text{Type}, B : \text{Type} \vdash M : (A \wedge B) \rightarrow (B \wedge A)$, we get the PS-derivation below:

$$\frac{\frac{\frac{\pi_B}{\Gamma \vdash_{\text{PS}} N_B : B} \quad \frac{\frac{\pi_A}{\Gamma \vdash_{\text{PS}} N_A : A} \quad \frac{}{\Gamma; Q \vdash_{\text{PS}} [] : Q} \text{axiom}_{\text{PS}}}{\Gamma; A \rightarrow Q \vdash_{\text{PS}} N_A \cdot [] : Q} \text{II}_{\text{PS}}}{\Gamma; B \rightarrow (A \rightarrow Q) \vdash_{\text{PS}} N_B \cdot N_A \cdot [] : Q} \text{III}_{\text{PS}}}{\Gamma \vdash_{\text{PS}} y N_B \cdot N_A \cdot [] : Q} \text{Select}_y}{A : \text{Type}, B : \text{Type} \vdash_{\text{PS}} \lambda x. \lambda Q. \lambda y. y N_B \cdot N_A \cdot [] : (A \wedge B) \rightarrow (B \wedge A)} \text{IIr}_{\text{PS}}$$

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

$$\frac{\frac{\frac{\frac{\Gamma; x' : A, y' : B; A \vdash_{\text{PS}} [] : A}{\Gamma; x' : A, y' : B \vdash_{\text{PS}} x' : A}}{\Gamma \vdash_{\text{PS}} \lambda x'. \lambda y'. x' : A \rightarrow (B \rightarrow A)} \quad \frac{}{\Gamma; A \vdash_{\text{PS}} [] : A}}{\Gamma; \langle A/Q \rangle (A \rightarrow (B \rightarrow Q)) \rightarrow Q \vdash_{\text{PS}} (\lambda x'. \lambda y'. x') \cdot [] : A}}{\Gamma; A \wedge B \vdash_{\text{PS}} A \cdot (\lambda x'. \lambda y'. x') \cdot [] : A}}{\Gamma \vdash_{\text{PS}} x A \cdot (\lambda x'. \lambda y'. x') \cdot [] : A}$$

and π_B is the derivation similar to π_A ($N_B = x B \cdot (\lambda x'. \lambda y'. y') \cdot []$) with conclusion $\Gamma \vdash_{\text{PS}} x B \cdot (\lambda x'. \lambda y'. y') \cdot [] : B$.

This example shows how the non-determinism of proof-search is sometimes quite constrained by the need to eventually unify the type in the stoup with the goal. For instance in π_A (resp. π_B), the resolution of $\Gamma \vdash Q : \text{Type}$ by A (resp. B) could be inferred from the unification in the right-hand side branch.

In Coq [Coq], the proof-search tactic `apply x` can be decomposed into the bottom-up application of `Selectx` followed by a series of bottom-up applications of `IIPS` and finally `axiomPS`, but it either delays the resolution of sub-goals or automatically solves them from the unification attempt, often avoiding obvious back-tracking.

In order to mimic even more closely this basic tactic, delaying the resolution of sub-goals can be done by using meta-variables, to be instantiated later with the help of the unification constraint. By extending PTSC with meta-variables, we can go further and express a sound and complete algorithm for type inhabitant enumeration (similar to Dowek's [Dow93] and Muñoz's [Mun01] in natural deduction) simply as the bottom-up construction of derivation trees in sequent calculus.

Proof-search tactics in natural deduction simply depart from the simple bottom-up application of the typing rules, so that their readability and usage would be made more complex. Just as in propositional logic [DP99a], sequent calculi can be a useful theoretical approach to study and design those tactics, in the hope to improve semi-automated reasoning in proof-assistants such as Coq.

7 Conclusion and Further Work

We have defined a parameterised formalism that gives 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 PTSC is in a sequent-calculus style, thus avoiding the use of implicit or explicit conversions to natural deduction [GR03, PD00].

A strong correspondence with natural deduction has been established (regarding both the logic and the strong normalisation), and we derive from it the confluence of each PTSC. We can give as examples the corners of Barendregt's 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.

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, the latter is also more elegant than natural deduction to express classical logic, so it would be interesting to build classical Pure Type Sequent Calculi.

References

- [Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Hand. Log. Comput. Sci.*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.
- [BG99] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoret. Comput. Sci.*, 211(1-2):375–395, 1999.
- [Blo01] R. Bloo. Pure type systems with explicit substitution. *Math. Structures in Comput. Sci.*, 11(1):3–19, 2001.
- [BR95] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *Computing Science in the Netherlands (CSN '95)*, pages 62–72, Koninklijke Jaarbeurs, Utrecht, 1995.
- [CH88] T. Coquand and G. Huet. The calculus of constructions. *Inf. Comput.*, 76(2–3):95–120, 1988.
- [Coq] The Coq Proof Assistant. <http://coq.inria.fr/>.
- [Dow93] G. Dowek. A complete proof synthesis method for type systems of the cube. *J. Logic Comput.*, 1993.
- [DP99a] R. Dyckhoff and L. Pinto. Proof search in constructive logics. In *Sets and proofs (Leeds, 1997)*, pages 53–65. Cambridge Univ. Press, Cambridge, 1999.
- [DP99b] R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoret. Comput. Sci.*, 212(1–2):141–155, 1999.
- [DU03] R. Dyckhoff and C. Urban. Strong normalization of Herbelin's explicit substitution calculus with substitution propagation. *J. Logic Comput.*, 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.

- [GR03] 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, *ENTCS*, volume 84. 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 Int. Work., CSL '94*, volume 933 of *LNCS*, pages 61–75. Springer, 1994.
- [Her95] H. Herbelin. *Séquents qu'on calcule*. PhD thesis, Université Paris 7, 1995.
- [Hue89] G. Huet. The constructive engine. *World Scientific Publishing*, Commemorative Volume for Gift Siromoney, 1989.
- [Kik04] K. Kikuchi. A direct proof of strong normalization for an extended Herbelin's calculus. In Y. Kameyama and P. J. Stuckey, editors, *Proc. of the 7th Int. Symp. on Functional and Logic Programming (FLOPS'04)*, volume 2998 of *LNCS*, pages 244–259. Springer-Verlag, 2004.
- [KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. Handwritten paper, University of Illinois, 1980.
- [Kri] J.-L. Krivine. Un interpréteur du λ -calcul. available at <http://www.pps.jussieu.fr/~krivine/>.
- [LDM] S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory - longer version. available at <http://www.pps.jussieu.fr/~lengrand/Work/Reports/Proofs.ps>.
- [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.
- [MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl Logic*, 51:125–157, 1991.
- [Mun01] C. Munoz. Proof-term synthesis on dependent-type systems via explicit substitutions. *Theor. Comput. Sci.*, 266(1-2):407–440, 2001.
- [PD00] L. Pinto and R. Dyckhoff. Sequent calculi for the normal terms of the $\lambda\Pi$ and $\lambda\Pi\Sigma$ calculi. In D. Galmiche, editor, *ENTCS*, volume 17. Elsevier, 2000.
- [Pol98] E. Poll. Expansion Postponement for Normalising Pure Type Systems. *J. Funct. Programming*, 8(1):89–96, 1998.
- [Pra65] D. Prawitz. Natural deduction. a proof-theoretical study. In *Acta Universitatis Stockholmiensis*, volume 3. Almqvist & Wiksell, 1965.
- [JMP94] 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 *LNCS*. Springer-Verlag, 1994.
- [Zuc74] J. Zucker. The correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–156, 1974.