

On Zucker's isomorphism for LJ and its extension to Pure Type Systems

Felix Joachimski, May 5, 2003

joachski@mathematik.uni-muenchen.de

Mathematisches Institut, Ludwig-Maximilians-Universität München
Theresienstrasse 39, 80333 München, Germany

Abstract. It is shown how the sequent calculus LJ can be embedded into a simple extension of the λ -calculus by generalized applications, called AJ. The reduction rules of cut elimination and normalization can be precisely correlated, if explicit substitutions are added to AJ. The resulting system AJ_□ is proved strongly normalizing, thus showing strong normalization for Gentzen's cut elimination steps. This refines previous results by Zucker, Pottinger and Herbelin on the isomorphism between natural deduction and sequent calculus.

The concept of generalized applications extends to Pure Type Systems, so that in particular sequent calculus analogues for all systems of the λ -cube arise. Cut elimination and strong β -normalization are shown to be equivalent for all Pure Type Systems.

Introduction

In the structural proof theory of intuitionistic logic, the most puzzling rule of the sequent calculus is

$$(l) \frac{\Phi \vdash A \quad \Phi, B \vdash C}{\Phi, A \rightarrow B \vdash C}$$

Although it is quite appropriate (yet not efficient) for proof search purposes, it somehow resists the otherwise beautiful correspondence between sequent calculus and natural deduction and leads to technical difficulties when comparing the notions of cut elimination on the one side and normalization on the other (see [Pra65, Zuc74, Pot77]).

From an intuitive natural deduction perspective, the left-sided rules of sequent calculus work on the leaves of derivations —the assumptions—, while the right-sided rules, just as all natural deduction rules, pertain the derivation root only. From the more formal (Curry-Howard) viewpoint of derivations as λ -terms, the left-sided rules require substitution rather than the basic term constructors. For the instance of (l) this reads¹

$$(l) \frac{\Gamma \vdash r : A \quad \Gamma, x : B \vdash s : C}{\Gamma, y : A \rightarrow B \vdash s[x := yr] : C}$$

¹ Prawitz was the first to formulate this embedding, albeit not with derivation terms. In its λ -calculus formulation it is also discussed in [DP98, Sch99, BG00, VW01].

Considering only cut-free sequent calculus derivations — i.e., constructed by means of (l) and

$$(r) \frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x r : A \rightarrow B} \quad (v) \frac{}{\Gamma, x : A \vdash x : A} \quad (x) \frac{\Gamma, x : A, y : A \vdash r : B}{\Gamma, x : A \vdash r[y := x] : B}$$

— it is a striking fact that the corresponding inductively defined set of cut-free λ -terms

$$A \supset \Lambda_{\text{cf}} \ni r, s = s[x := yr] \mid \lambda x r \mid x \mid r[x := y]$$

exactly captures the β -normal terms of the λ -calculus, given by the grammar

$$A \supset \text{NF}_\beta \ni r, s ::= x\bar{r} \mid \lambda x r,$$

although the decomposition of NF_β into Λ_{cf} -clauses is not unique.

Using general substitutions, the exact correspondence on cut-free/normal derivations can be extended to derivations with the cut-rule

$$(c) \frac{\Gamma, x : A \vdash r : B \quad \Gamma \vdash s : A}{\Gamma \vdash r[x := s] : B}$$

Quite fascinating, the advent of explicit substitutions in the λ -calculus allowed Herbelin to even formally relate (some of) Gentzen's [Gen35] reduction rules for cut elimination to the reduction rules for an explicit substitution calculus [Her94] and in this way prove them weakly normalizing, thus continuing the long tradition of interpreting sequent calculus derivations as a subset of substituted λ -terms.

Yet substitutions are notoriously cumbersome to define and work with, in particular the explicit variants, not the least, because their reduction behaviour is quite awkward, if permutation of substitutions is allowed [Mel95], as is required for Gentzen's reduction steps. It has thus been argued that a more restricted extension of λ -calculus is necessary that encompasses sequent-calculus-like rules and retains customary metatheoretic properties. The calculus ΛJ represents such an attempt: it extends the grammar of cut-free LJ-derivations — hence the name — and that of the λ -calculus. Its reformulation of the (l) -rule can be understood as the term-based counterpart of von Plato's generalized \rightarrow -elimination [vP01]

$$(a) \frac{\Gamma \vdash r : A \rightarrow B \quad \Gamma \vdash s : A \quad \Gamma, z : B \vdash t : C}{\Gamma \vdash r(s, z.t) : C}$$

As previously shown, untyped ΛJ with its reduction rules

$$(\lambda x r)(s, z.t) \rightarrow_{\beta_j} t[z := r[x := s]], \quad r(s, z.t)(s', z'.t') \rightarrow_{\pi} r(s, z.t(s', z'.t'))$$

enjoys confluence and standardization [JM00] and in the simply typed form is strongly normalizing [JM03]. In [Mat00], an intersection type system for ΛJ has

been devised and shown to precisely capture the set of strongly normalizing terms; in [Mat01] the calculus is used to obtain a particularly strong form of interpolation theorem.

In this article, we make the correlation of the reduction behavior of λJ and λLJ explicit. More precisely, it is shown that Gentzen’s original reduction rules (fitted to a calculus without weakening and exchange) are verbally the same as those of λJ with simple (non permutating) explicit substitutions. From this we derive strong cut elimination, i.e., the termination of Gentzen’s reductions in arbitrary order, building on

- a new strong normalization proof for typed λJ , which reduces the statement to strong normalization of typed λ ,
- a new strong normalization proof for the typed λ -calculus with explicit substitutions, which can easily be extended to λJ with explicit substitutions.

Furthermore, the generalization of the application rule is extended to arbitrary Pure Type Systems to provide an abstract framework for sequent-calculus-like arguments in dependent type systems, thus improving previous results in the literature [BG00, GR02b].

Outline of the contents. Section 1 reviews the definitions of λJ and the λ -calculus with and without explicit substitutions, giving a summary of the previously established correlations.

Section 2 gives a new proof of strong normalization for the simply-typed λ -calculus with non-compositional explicit substitutions.

Section 3 recalls the calculus λJ and provides the equivalence results between λJ and λLJ , including a new proof of strong normalization of λJ without and with explicit substitutions. This directly leads to strong cut elimination for λJ .

Section 4 extends these considerations to Pure Type Systems, providing a definition of generalized application and the equivalence proof of strong normalization for the calculi with and without generalized application, thus establishing cut elimination for all normalizing Pure Type Systems.

Notation. We use \rightarrow_θ to denote the term closure of a given elementary reduction rule \rightarrow_θ . $\rightarrow_{\theta\theta'}$ is used for the union of \rightarrow_θ and $\rightarrow_{\theta'}$. We write $r \Downarrow_\theta$ iff r is strongly normalizing w.r.t. \rightarrow_θ , in which case $\#_\theta r$ denotes the height of the θ -reduction tree of r . \rightarrow_θ^* stands for the reflexive, transitive, and \rightarrow_θ^+ for the transitive closure of \rightarrow_θ . \rightarrow_θ^n is the n -fold relational composition of \rightarrow_θ with itself. NF_θ denotes the set of normal forms w.r.t. \rightarrow_θ . If \rightarrow_θ is confluent and weakly normalizing, then r^θ denotes the normal form of r w.r.t. \rightarrow_θ .

In most sections we will fix one particular reduction relation \rightarrow and write $r \Downarrow$ for strong normalizability w.r.t. \rightarrow , using $\#r$ for the reduction tree height.

Acknowledgements. I am very grateful to Ralph Matthes for valuable remarks on early drafts of this article. I also want to thank the people of the Swansea University Computer Science Department for their hospitality during my stay there in February 2003.

1 Sequent calculus

This section recapitulates the definition of the sequent calculus for the implicational fragment of minimal propositional logic and cut elimination therein, as well as the customary embeddings from and to the simply-typed λ -calculus with explicit substitutions.

1.1. LJ. *Formulas* A, B, C are inductively constructed from propositional variables P by $A \rightarrow B$. In *sequents* $\Phi \vdash A$, the metavariable Φ stands for a multiset of formulas, so that there is no need for the structural rule of exchange.

The calculus LJ is given by the rules

$$\begin{array}{ll}
 (l) \frac{\Phi \vdash A \quad \Phi, B \vdash C}{\Phi, A \rightarrow B \vdash C} & (r) \frac{\Phi, A \vdash B}{\Phi \vdash A \rightarrow B} \\
 (c) \frac{\Phi \vdash A \quad \Phi, A \vdash B}{\Phi \vdash B} & (v) \frac{}{\Phi, A \vdash A}
 \end{array}$$

This calculus enjoys *weakening* in the sense that $\Phi \vdash A$ implies $\Phi, B \vdash A$. It also derives *contraction*

$$(x) \frac{\Phi, A, A \vdash B}{\Phi, A \vdash B}$$

using the cut rule as follows

$$(c) \frac{(v) \frac{}{\Phi, A \vdash A} \quad \Phi, A, A \vdash B}{\Phi, A \vdash B}$$

1.2. LJ'. Kleene's variant LJ' of LJ uses a formulation of (l) which requires that the newly introduced formula $A \rightarrow B$ was present in the assumptions of the premises:

$$(l') \frac{\Phi, A \rightarrow B \vdash' A \quad \Phi, A \rightarrow B, B \vdash' C}{\Phi, A \rightarrow B \vdash' C}$$

In this calculus, contraction is admissible without cut. We will use the notation $\Phi \vdash' A$ for derivations in LJ'.

1.3. Cut elimination. The original reduction rules of Gentzen's cut elimination proof for LJ (reformulated to forgo structural rules) are displayed in figure 1. Rather than repeating Gentzen's direct termination proof, we will further analyze these reduction rules as a rewrite system in the following subsections.

1.4. λ -calculus. As has become customary, we use terms of the λ -calculus Λ to denote derivations in LJ. *Terms* are given by the grammar

$$\Lambda \ni r, s, t ::= x \mid \lambda x r \mid r s,$$

$$\begin{array}{c}
\begin{array}{c}
\text{(r)} \frac{A \vdash B}{\vdash A \rightarrow B} \quad \text{(l)} \frac{\vdash A \quad B \vdash C}{A \rightarrow B \vdash C} \\
\text{(c)} \frac{}{\vdash C}
\end{array}
\quad \rightsquigarrow_{\beta} \quad
\begin{array}{c}
\text{(c)} \frac{\vdash A \quad A \vdash B}{\vdash B} \\
\text{(c)} \frac{}{\vdash C}
\end{array}
\\
\\
\begin{array}{c}
\text{(c)} \frac{A \vdash B \quad B \vdash B}{A \vdash B} \quad \rightsquigarrow_{\square} \quad A \vdash B \\
\text{(c)} \frac{\vdash B \quad A, B \vdash A}{A \vdash A} \quad \rightsquigarrow_{\square} \quad A \vdash A
\end{array}
\\
\\
\begin{array}{c}
\text{(c)} \frac{\vdash D \quad D \vdash A \quad D, B \vdash C}{A \rightarrow B \vdash C} \quad \rightsquigarrow_{\square} \quad \text{(l)} \frac{\vdash D \quad D \vdash A}{\vdash A} \quad \text{(c)} \frac{\vdash D \quad D, B \vdash C}{B \vdash C} \\
A \rightarrow B \vdash C
\end{array}
\\
\\
\begin{array}{c}
\text{(c)} \frac{A \rightarrow B \vdash A \rightarrow B}{A \rightarrow B \vdash C} \quad \rightsquigarrow_{\square} \quad \text{(l)} \frac{\vdash A \quad B \vdash C}{A \rightarrow B \vdash C} \\
\text{(c)} \frac{\vdash C}{\vdash A \rightarrow B} \quad \text{(r)} \frac{C, A \vdash B}{C \vdash A \rightarrow B} \\
\vdash A \rightarrow B
\end{array}
\\
\\
\begin{array}{c}
\text{(l)} \frac{\vdash A \quad B \vdash A' \rightarrow B'}{A \rightarrow B \vdash A' \rightarrow B'} \quad \text{(l)} \frac{\vdash A' \quad B' \vdash C'}{A' \rightarrow B' \vdash C'} \\
\text{(c)} \frac{}{A \rightarrow B \vdash C}
\end{array}
\quad \rightsquigarrow_{\pi} \quad
\begin{array}{c}
\text{(w)} \frac{\vdash A' \quad \dots \quad B' \vdash C}{B \vdash A'} \quad \text{(w)} \frac{\dots \quad B, B' \vdash C}{B, B' \vdash C} \\
\text{(l)} \frac{}{B \vdash C} \\
\text{(c)} \frac{B \vdash A' \rightarrow B'}{A \rightarrow B \vdash C}
\end{array}
\end{array}$$

Fig. 1. Gentzen's reduction rules for LJ. In all rules, the part of the context that all premises and the conclusion have in common, is omitted.

where x, y, z denote variables. As usual, we will identify terms that differ only in the renaming of bound variables and let application associate to the left, writing $r\vec{s}$ for $(\dots(rs_1)\dots)s_n$ with a possibly empty vector \vec{s} . $x \in r$ means that x is free in r . We use the dot notation $\lambda x.r$ to omit the parentheses in $\lambda x(r)$, where the range of the dot extends as far as syntactically possible. Substitution is written $r[x := s]$ and straightforwardly defined.

Reduction \rightarrow_β is generated as the term closure of elementary β -reduction

$$(\lambda xr)s \rightarrow_\beta r[x := s].$$

The (strongly normalizing) *simply-typed* λ -calculus $\vec{A} \subset A$ is obtained from A by the following type assignment system:

$$(v) \frac{}{\Gamma, x : A \vdash x : A} \quad (r) \frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda xr : A \rightarrow B} \quad (a) \frac{\Gamma \vdash r : A \rightarrow B \quad \Gamma \vdash s : A}{\Gamma \vdash rs : B}$$

It is easily checked that $\Gamma \vdash r : A$ and $r \rightarrow r'$ imply $\Gamma \vdash r' : A$ (*subject reduction*); this uses the following type assignment for substitution:

$$(c) \frac{\Gamma \vdash s : A \quad \Gamma, x : A \vdash r : B}{\Gamma \vdash r[x := s] : B}$$

1.5. Notations for LJ. The following decoration of the rules of LJ assigns formulas (types) to λ -terms (as usual, Γ stands for a unique assignment of formulas to variables $x_1 : A_1, \dots, x_n : A_n$ and $\Gamma, x : A$ implies that $x \notin \Gamma$):

$$(l) \frac{\Gamma \vdash s : A \quad \Gamma, z : B \vdash t : C}{\Gamma, y : A \rightarrow B \vdash t[z := ys] : C} \quad (r) \frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda xr : A \rightarrow B}$$

$$(c) \frac{\Gamma \vdash s : A \quad \Gamma, x : A \vdash r : B}{\Gamma \vdash r[x := s] : B} \quad (v) \frac{}{\Gamma, x : A \vdash x : A}$$

Using these notations for LJ-derivations, it is straightforward to verify that both LJ and \vec{A} derive the same formulas (*equivalence*), the only interesting case being that of an application rs , which is obtained as $x[x := ys][y := r]$ with new y .²

1.6. Notations for LJ'. These are obtained by the following decoration of (l'):

$$(l') \frac{\Gamma, y : A \rightarrow B \vdash s : A \quad \Gamma, y : A \rightarrow B, z : B \vdash t : C}{\Gamma, y : A \rightarrow B \vdash t[z := ys] : C}$$

1.7. LJ_{cf}. Let LJ_{cf} stand for the system LJ with the cut rule (c) replaced by contraction (x) (see subsection 1.1), writing $\Phi \vdash_{\text{cf}} A$ for cut free derivability. The

² Obviously, this is not the only possibility.

corresponding term calculus of Λ is given inductively³ by

$$\Lambda_{\text{cf}} \ni r, s, t = x \mid \lambda x r \mid r[x := y] \mid r[x := ys],$$

where the last clause is subject to the proviso that $y \notin r, s$.

A simple induction verifies that $\Lambda_{\text{cf}} \subseteq \text{NF}_\beta$, the set of normal forms of Λ , as given by the grammar

$$\text{NF}_\beta \ni r, s, t ::= x\vec{r} \mid \lambda x r.$$

Conversely, every term in NF_Λ can be found in Λ_{cf} , but its deconstruction by the inductive characterization is obviously not unique. For instance $x(yz)$ might be derived as $x'[x' := xy'][y' := yz]$ or as $x'[x' := xy'[y' := yz]]$.

Note that the calculus LJ_{cf} without (x) does not enjoy contraction and is therefore not complete.

1.8. LJ'_{cf} . Similarly, LJ'_{cf} is LJ' without the rule (c). The inductive normal form characterization is

$$\Lambda'_{\text{cf}} \ni r, s, t = x \mid \lambda x r \mid r[x := ys]$$

without further provisos. Rule (l') of LJ'_{cf} can be emulated in LJ_{cf} by (l) and (x), e.g.,

$$(l) \frac{\Gamma, y : A \rightarrow B \vdash A \quad \Gamma, y : A \rightarrow B, z : B \vdash t : C}{\Gamma, y : A \rightarrow B, y' : A \rightarrow B \vdash t[z := y's]} \\ (x) \frac{\Gamma, y : A \rightarrow B \vdash t[z := y's]}{\Gamma, y : A \rightarrow B \vdash t[z := y's][y' := y]}$$

but for any given derivation of LJ'_{cf} one can devise various equivalent LJ_{cf} -derivations, differing only in the position of the contraction rules.

1.9. Cut elimination. Gentzen's Hauptsatz reads

$$\Phi \vdash A \implies \Phi \vdash_{\text{cf}} A.$$

i.e., all sequents of LJ are derivable without the cut rule.

Using the notation system above, there is a trivial way to reduce this problem to normalization of the simply typed λ -calculus: any derivation in LJ is denoted by a simply typed λ -term; as shown in subsection 1.7, its normal form denotes an LJ'_{cf} -derivation, which can be interpreted as a derivation of LJ_{cf} that is normal w.r.t. Gentzen's reduction rules.

The goal of this article, however, is to prove strong cut elimination, i.e., strong normalization of Gentzen's reduction rules. To this end we first recall the explicit substitution approach to notations for LJ.

1.10. Explicit substitutions. In [Her94], explicit substitutions were used to denote derivations of LJ and a weak normalization result for some of Gentzen's

³ Note that this inductive characterization is *not* a grammar, because substitution is not a syntactic constructor in Λ , but a metasyntactic device.

reduction rules obtained. As a preliminary to the later treatment of the calculus ΛJ with explicit substitutions, we recall some basic concepts of explicit substitutions for Λ along the lines of [Her94], yet technically quite different and simpler.

Terms of the calculus Λ_\square are given by the grammar⁴

$$\Lambda_\square \ni r, s, t ::= x \mid rs \mid \lambda xr \mid r[x := s].$$

The reduction relation $\rightarrow := \rightarrow_{\square\beta\alpha p}$ is obtained as the term closure of the following reduction rules (assume $x \neq y$)

$$\begin{array}{ll} x[x := s] \rightarrow_\square s, & (rs)[x := t] \rightarrow_\square r[x := t]s[x := t], \\ x[y := s] \rightarrow_\square x, & (\lambda yr)[x := s] \rightarrow_\square \lambda y(r[x := s]), \quad y \notin s, \\ (\lambda xr)s \rightarrow_{\beta\alpha} r[x := s], & r[x := s][y := t] \rightarrow_p r[y := t][x := s[y := t]]. \end{array}$$

Weak normalization of typed Λ_\square (see the next section for a definition) can be established by reduction to normalization in the typed λ -calculus using the straightforward embedding of Λ_\square into Λ (interpreting explicit as implicit substitutions) and strong normalization of \rightarrow_\square . In the next section we will provide a proof of strong normalization in typed Λ_\square without the reduction \rightarrow_p .

1.11. Typed Λ_\square . Using rule (c) (see subsection 1.5) and the typing rules of $\vec{\Lambda}$, we get the simply typed calculus $\vec{\Lambda}_\square$, where \rightarrow preserves typability and types (*subject reduction*).

1.12. Simulation. In $\vec{\Lambda}_\square$, Gentzen's reduction rules (see figure 1) read (assume x, y, z to be all different)

$$\begin{array}{ll} x[x := s] \rightsquigarrow_\square s, & \\ x[y := s] \rightsquigarrow_\square x, & \\ (\lambda yr)[x := t] \rightsquigarrow_\square \lambda y.r[x := t], & y \notin t, \\ t[z := ys][y := y'] \rightsquigarrow_\square t[z := y's], & y \notin s, t, \\ t[z := ys][x := r] \rightsquigarrow_\square t[x := r][z := y s[x := r]], & \\ t[z := ys][y := \lambda xr] \rightsquigarrow_\beta t[z := r[x := s]], & y \notin s, t, \\ t'[z' := ys'][y := t[z := xs]] \rightsquigarrow_\pi t'[z' := ys'][y := t][z := xs], & y \notin s', t'. \end{array}$$

Note that these reductions are simulated by the ambient notion of reduction \rightarrow^* in Λ_\square , using \rightarrow_p for the difficult last four rules. Unfortunately, it is precisely \rightarrow_p which renders the calculus non-terminating [Mel95]. It is not clear how to devise a general reduction strategy for Λ_\square that would exactly capture Gentzen's rules. For instance, Herbelin's reduction strategy (proved terminating in [Her94]) does not allow to prove simulation in this sense.

⁴ See also the previous footnote; in Λ_\square , substitution $r[x := s]$ becomes a syntactic constructor.

2 Strong normalization for $\vec{\Lambda}_\square$ without \rightarrow_p

In this section we develop a comparably short proof of strong normalization for $\rightarrow := \rightarrow_{\square\beta_\square}$ in the simply-typed system $\vec{\Lambda}_\square$, thus simplifying proofs of analogous theorems in the literature [Her94, BG97].

2.1. \square -normalization. For $r \in \Lambda_\square$ we define its \square -height $\#_\square r$ by recursion on r :

$$\begin{aligned} \#_\square x &:= 1, & \#_\square (r[x := s]) &:= \#_\square r \cdot (\#_\square s + 1), \\ \#_\square (\lambda xr) &:= 1 + \#_\square r, & \#_\square (rs) &:= \#_\square r + \#_\square s + 1. \end{aligned}$$

Proposition 1. $r \rightarrow_\square s \implies \#_\square r > \#_\square s$.

Proof. Simple induction on \rightarrow_\square . The interesting case is that of an elementary reduction $(rr')[x := s] \rightarrow_\square r[x := s] r'[x := s]$:

$$\begin{aligned} \#_\square((rr')[x := s]) &= (\#_\square r + \#_\square r' + 1) \cdot (\#_\square s + 1) \\ &> \#_\square r \cdot (\#_\square s + 1) + \#_\square r' \cdot (\#_\square s + 1) \\ &= \#_\square(r[x := s] r'[x := s]). \quad \square \end{aligned}$$

Thus \rightarrow_\square is strongly normalizing. The \square -normal form $r^\square \in \Lambda$ is given by recursion on $r \in \Lambda_\square$:

$$x^\square := x, \quad (\lambda xr)^\square := \lambda x.r^\square, \quad (rs)^\square := r^\square s^\square, \quad (r[x := s])^\square := r^\square[x := s^\square].$$

Lemma 2.

- (i) r^\square is in \rightarrow_\square -normal form,
- (ii) $r \rightarrow_\square^* r^\square$,
- (iii) $r \rightarrow_\square s \implies r^\square = s^\square$,
- (iv) $r \rightarrow_{\beta_\square} s \implies r^\square \rightarrow_\beta^* s^\square$,
- (v) $\Lambda \ni r \rightarrow_\beta s \implies \Lambda_\square \ni r \rightarrow^+ s$.

Thus \rightarrow_\square is convergent, i.e., confluent and strongly normalizing with normal forms given by $()^\square$.

Corollary 3. \rightarrow is confluent.

Proof. Assume $\Lambda_\square : s \xleftarrow{*} r \rightarrow^* t$.
 By (iii) and (iv) $\Lambda : s^\square \xleftarrow{\beta^*} r^\square \rightarrow_\beta^* t^\square$.
 By confluence of \rightarrow_β $\Lambda : s^\square \rightarrow_\beta^* r' \xleftarrow{\beta^*} t^\square$.
 By (v) $\Lambda_\square : s^\square \rightarrow^* r' \xleftarrow{*} t^\square$.
 Altogether $\Lambda_\square : s \rightarrow_\square^* s^\square \rightarrow^* r' \xleftarrow{*} t^\square \xleftarrow{\square} t$. \square

2.2. Subterm relation. Let $\rightarrow_\triangleright$ denote the term closure of the immediate subterm relation, as given by

$$rs \rightarrow_\triangleright r, s, \quad \lambda xr \rightarrow_\triangleright r.$$

Proposition 4. $\Lambda : r \Downarrow_{\beta} \implies r \Downarrow_{\beta \triangleright}$.

Proof. Straightforward induction on $\#_{\beta} r$ with side induction on r . \square

2.3. Strong normalization. For the rest of this section we restrict all quantifiers to typable terms.

As a mere technical concept we introduce \rightarrow' -reduction to be the reduction relation \rightarrow without the elementary β_{\square} -reduction $(\lambda xr)s \rightarrow_{\beta_{\square}} r[x := s]$. Thus \rightarrow' contains all \square -reductions and all inner β_{\square} -reductions. $r \Downarrow'$ denotes strong normalizability w.r.t. \rightarrow' and $\#' r$ stands for the reduction tree height of r w.r.t. \rightarrow' if $r \Downarrow'$.

Proposition 5. (A) $r \Downarrow \ \& \ s \Downarrow \implies rs \Downarrow'$.

Proof. Obvious, since all reductions w.r.t. \rightarrow' have to occur inside either r or s . \square

Theorem 6. (B) $r \Downarrow \ \& \ s \Downarrow \implies t := r[x := s] \Downarrow'$,
(C) $t \Downarrow' \implies t \Downarrow$.

Proof. Simultaneous induction on $\#_{\beta \triangleright} t^{\square}$, using proposition 4 and strong normalization of $\vec{\Lambda}$.

(B) is shown by side induction on $\#r$ and side side induction on $\#s$. We have to show strong normalizability w.r.t. \rightarrow' for every \rightarrow -reduct of $r[x := s]$.

Case $(rr')[x := s] \rightarrow_{\square} r[x := s]r'[x := s]$. Using the proposition we need to show $r[x := s] \Downarrow$ and $r'[x := s] \Downarrow$. The main induction hypothesis (C) is applicable to $r[x := s]$ and $r'[x := s]$, since $r[x := s]r'[x := s] \triangleright r[x := s], r'[x := s]$, so that we need to show $r[x := s] \Downarrow'$ and $r'[x := s] \Downarrow'$. Both follow from the main induction hypothesis (B).

Case $(\lambda yr)[x := s] \rightarrow_{\square} \lambda y.r[x := s]$. Similar, but easier.

Case $x[x := s] \rightarrow_{\square} s$ and $y[x := s] \rightarrow_{\square} y$. Clear.

Case of an inner reduction $r[x := s] \rightarrow r'[x := s']$. By (iii) and (iv), this at most decreases the main induction measure, but decreases $\#r$ or $\#s$.

(C) is shown by side induction on $\#'$. We need to show strong normalizability of every reduct t' of t .

Case $r \rightarrow' r'$. The side induction hypothesis applies and yields $r' \Downarrow$.

Case $(\lambda xr)s \rightarrow_{\beta_{\square}} r[x := s]$. This reduction is simulated on the \square -normal form:

$$((\lambda xr)s)^{\square} = (\lambda xr^{\square})s^{\square} \rightarrow_{\beta} r^{\square}[x := s^{\square}] = (r[x := s])^{\square}.$$

By the main induction hypothesis (C) for $r[x := s]$ we need to show $r[x := s] \Downarrow'$. This follows from the main induction hypothesis (B) for $r[x := s]$, since $(\lambda xr)s \Downarrow'$ entails $r \Downarrow$ and $s \Downarrow$. \square

Corollary 7. $r \Downarrow$.

Proof. Induction on r . **Case** x . Trivial. **Case** λxr . Simple application of the induction hypothesis. **Case** rs . By induction hypothesis $r \Downarrow$ and $s \Downarrow$, so by (A) $(rs) \Downarrow'$, hence $(rs) \Downarrow$ by (C). **Case** $r[x := s]$. By induction hypothesis $r \Downarrow$ and $s \Downarrow$, so by (B) $r[x := s] \Downarrow'$ and (C) yields $r[x := s] \Downarrow$. \square

3 The calculus λJ

In this section the calculus λJ —a conservative syntactic extension of the λ -calculus introduced and studied in [JM00, JM03]— is used to precisely simulate Gentzen’s reductions of LJ (see subsection 1.12). It indirectly captures some of the reductions in Λ_\square and in this sense λJ can be viewed as a rudimentary explicit substitution calculus.

3.1. Definition. *Terms* of λJ are given by the grammar

$$\lambda J \ni r, s, t ::= x \mid \lambda x r \mid r(s, z.t).$$

The variable z gets bound in t in the generalized application $r(s, z.t)$. Again, we consider terms up to α -equivalence. The definition of substitution $r[x := s]$ is straightforward. We also write R, S, T for expressions of the form $(s, z.t)$, so that generalized application reads rR .

Reduction $\rightarrow := \rightarrow_{\beta_j \pi}$ is generated as the term closure of β_j -reduction and permutation, given by

$$(\lambda x r)(s, z.t) \rightarrow_{\beta_j} t[z := r[x := s]] \quad \text{and} \quad r(s, z.t)T \rightarrow_{\pi} r(s, z.tT).$$

This formulation of \rightarrow_{π} is only compatible with the variable convention if $z \notin T$, so that this has to be implicitly assumed.

We note without proof that \rightarrow is confluent and standardizing, referring the interested reader to [JM00] for details. The set of normal forms is given by the grammar

$$\text{NF}_{\beta_j \pi} \ni r, s, t ::= x \mid x(s, z.t) \mid \lambda x r.$$

3.2. π -normalization. \rightarrow_{π} is strongly normalizing and confluent [JM00]; the π -normal form of a term r^{π} is recursively given by

$$\begin{aligned} x^{\pi} &:= x, \\ (\lambda x r)^{\pi} &:= \lambda x. r^{\pi}, \\ (r(s, z.t))^{\pi} &:= r^{\pi} @ (s^{\pi}, z.t^{\pi}), \end{aligned} \quad \text{using} \quad r @ R := \begin{cases} r'(s, z.t @ R) & \text{if } r = r'(s, z.t), \\ rR & \text{otherwise.} \end{cases}$$

The set of π -normal forms is inductively characterized by the grammar

$$\text{NF}_{\pi} \ni r, s, t ::= x \mid x(s, z.t) \mid (\lambda x r)(s, z.t).$$

For later reference we state a simple commutation property of \rightarrow_{β_j} and $()^{\pi}$:

Lemma 8. $r \rightarrow_{\beta_j} s \implies \exists t. r^{\pi} \rightarrow_{\beta_j} t \rightarrow_{\pi}^* s^{\pi}$.

The proof is an easy induction on \rightarrow_{β_j} , using the following

Proposition 9. $r^{\pi}[x := s^{\pi}] \rightarrow_{\pi}^* (r[x := s])^{\pi}$.

3.3. $\Lambda \subseteq \Lambda\mathbf{J}$. In $\Lambda\mathbf{J}$ we abbreviate $rs := r(s, z.z)$, thus embedding the λ -calculus. Since

$$\Lambda\mathbf{J} : \quad (\lambda xr)s = (\lambda xr)(s, z.z) \rightarrow_{\beta_j} z[z := r[x := s]] = r[x := s],$$

β -reductions of Λ are simulated, using just β_j -reductions in $\Lambda\mathbf{J}$. Normal forms of the form $x\vec{r}$ become

$$x(r_1, z.z)(r_2, z.z) \dots (r_n, z.z).$$

Their π -normal form reads

$$(x\vec{r})^\pi = x(r_1^\pi, z.z)(r_2^\pi, z.z)(\dots z.z(r_n^\pi, z.z) \dots).$$

3.4. $\Lambda\mathbf{J} \subseteq \Lambda$ — first variant. For the converse embedding two variants are possible. The first embedding sets

$$\Lambda : \quad r(s, z.t) := t[z := rs] \quad (*).$$

Note that if $s, t \in \text{NF}_\beta$, then $\Lambda \ni x(s, z.t) = t[z := xs] \in \text{NF}_\beta$, so that $(*)$ preserves normal forms. It is also invariant under permutation, as follows from

$$\begin{aligned} \Lambda : \quad r(s, z.t)(s', z'.t') &= t'[z' := t[z := rs]s'] \\ &= t'[z' := ts'][z := rs] \quad (\dagger) \\ &= r(s, z.t(s', z'.t')), \end{aligned}$$

where the equality (\dagger) is justified by the implicit proviso $z \notin s'$ in our formulation of permutative reduction.

Note, that the embedding does not *simulate* β_j -reductions, because a β_j -redex $(\lambda xr)(s, z.t)$ with $z \notin t$ translates into just t .

Summarizing, $\Lambda\mathbf{J}$ contains many π -equal terms for each β -normal form of Λ , corresponding to the fact that there are many different cut-free sequent calculus derivations for each normal natural deduction proof.

3.5. $\Lambda\mathbf{J} \subseteq \Lambda$ — second variant. The second embedding uses

$$\Lambda : \quad r(s, z.t) := (\lambda zt)(rs) \quad (**),$$

thus creating a β -redex for every generalized application. Obviously, this embedding does not preserve normality. Yet it simulates reductions, as stated by

Lemma 10. $r \rightarrow_{\beta_j} s \implies r \rightarrow_{\beta}^2 s$.

Proof. Induction on \rightarrow_{β_j} . Reduction in subterms is treated by the induction hypothesis in the obvious way. For elementary β_j -reduction we compute in Λ :

$$(\lambda xr)(s, z.t) = (\lambda zt)((\lambda xr)s) \rightarrow_{\beta} (\lambda zt)(r[x := s]) \rightarrow_{\beta} t[z := r[x := s]]. \quad \square$$

This result can be used to derive strong normalizability of terms in ΛJ from their strong normalizability as terms of Λ .

First, we recall a basic property of $\#_\beta$, which has also been dubbed the fundamental lemma of perpetuality [vRSSX99]:

Proposition 11. $x \in r \implies \#_\beta((\lambda x r)s) \leq \#_\beta(r[x := s]) + 1$,
 $x \notin r \implies \#_\beta((\lambda x r)s) \leq \#_\beta r + \#_\beta s + 1$.

Lemma 12. $r \rightarrow_\pi s \implies \#_\beta r \geq \#_\beta s$.

Proof. Induction on \rightarrow_π . The inner reductions are all quite straightforward. The interesting case is that of a permutation

$$r(s, z.t)(s', z'.t') \rightarrow_\pi r(s, z.t(s', z'.t')).$$

In Λ , the two terms read

$$r_0 := (\lambda z't')((\lambda z t)(rs)s') \quad \text{and} \quad r_1 := (\lambda z.(\lambda z't')(ts'))(rs).$$

First assume $z \in t$. Then by the proposition

$$\#_\beta r_1 = 1 + \#_\beta((\lambda z't')(t[z := rs]s')) \leq \#_\beta r_0,$$

since $r_0 \rightarrow_\beta (\lambda z't')(t[z := rs]s')$. If, on the other hand, $z \notin t$, write r' for the normal form of rs . The proposition yields

$$\#_\beta r_1 = 1 + \#_\beta(rs) + \#_\beta((\lambda z't')(ts')) \leq \#_\beta r_0,$$

because $r_0 \rightarrow_{\beta}^{\#_\beta(rs)} (\lambda z't')((\lambda z t)r's') \rightarrow_\beta (\lambda z't')(ts')$. □

Theorem 13. $r^\pi \Downarrow \implies r \Downarrow$.

Proof. Induction on $\# r^\pi$, side induction on $\#_\pi r$.

Case $r \rightarrow_{\beta_j} s$. Then $\exists t. r^\pi \rightarrow_{\beta_j} t \rightarrow_\pi^* s^\pi$ by the commutation property (lemma 8). It follows that $s^\pi \Downarrow$ with $\# s^\pi < \# r^\pi$, so by induction hypothesis $s \Downarrow$.

Case $r \rightarrow_\pi s$. Then $r^\pi = s^\pi$ and by the side induction hypothesis $s \Downarrow$. □

Using the second interpretation of ΛJ in Λ , we thus obtain the

Corollary 14. $r \Downarrow_\beta \implies r \Downarrow$.

Proof. Induction on $\#_\beta r$ and side induction on $\#_\pi r$. So let $r \rightarrow s$.

Case $r \rightarrow_{\beta_j} s$. Then by lemma 8 $\exists t. r^\pi \rightarrow_{\beta_j} t \rightarrow_\pi^* s^\pi$, so by lemma 12

$$\#_\beta s^\pi \leq \#_\beta t < \#_\beta r^\pi \leq \#_\beta r$$

and the induction hypothesis shows $s^\pi \Downarrow$. Therefore $s \Downarrow$ by the theorem.

Case $r \rightarrow_\pi s$. Then by lemma 12 $\#_\beta s \leq \#_\beta r$. Also $\#_\pi s < \#_\pi r$, so either the side or the main induction hypothesis derives $s \Downarrow$. □

3.6. Simply-typed $\Lambda\mathbf{J}$. Using rules (v) and (r) of $\vec{\Lambda}\mathbf{J}$ together with the typing rule for generalized application

$$(a) \frac{\Gamma \vdash r : A \rightarrow B \quad \Gamma \vdash s : A \quad \Gamma, z : B \vdash t : C}{\Gamma \vdash r(s, z.t) : C}$$

we obtain the set of typed $\Lambda\mathbf{J}$ -terms $\vec{\Lambda}\mathbf{J}$. Note that contraction is admissible in $\vec{\Lambda}\mathbf{J}$.

In [JM03], a direct proof of strong normalization for $\vec{\Lambda}\mathbf{J}$ has been given in order to demonstrate how to adapt conventional normalization proofs to permutative conversions, which are prototyped by \rightarrow_π in $\Lambda\mathbf{J}$.

Here we can use the result of the previous subsection, with the advantage that the proof easily extends to more complex type systems.

Corollary 15. *$\vec{\Lambda}\mathbf{J}$ is strongly normalizing.*

3.7. $\mathbf{LJ} \subseteq \Lambda\mathbf{J}$. Given a derivation of $\Gamma \vdash s : A$ and $\Gamma, z : B \vdash t : C$ and $y \notin \Gamma$, we obtain $\Gamma, y : A \rightarrow B \vdash s : A$ and $\Gamma, y : A \rightarrow B, z : B \vdash t : C$ by weakening, so that we can simulate rule (l) by

$$(a) \frac{\Gamma, y : A \rightarrow B \vdash y : A \rightarrow B \quad \Gamma, y : A \rightarrow B \vdash s : A \quad \Gamma, y : A \rightarrow B, z : B \vdash t : C}{\Gamma, y : A \rightarrow B \vdash y(s, z.t) : C}$$

Translated into Λ (using the first variant given above), the term $y(s, z.t)$ reads $t[z := ys]$, which is identical to the term assignment for rule (l) in subsection 1.5.

3.8. $\mathbf{NF}_{\beta_j\pi} = \mathbf{LJ}'_{\text{cf}}$. In the typed setting, identifying the rules (l) and (a) for the case $x(r, z.s)$, the sets $\mathbf{NF}_{\beta_j\pi}$ and \mathbf{LJ}'_{cf} are in a one-to-one-correspondence, hence isomorphic; with respect to \mathbf{LJ} , a normal $\Lambda\mathbf{J}$ -term only determines \mathbf{LJ}_{cf} -derivations up to contractions.

As a corollary, we obtain that the following sets of terms are extensionally equal:

- the set of $\beta_j\pi$ -normal terms of $\Lambda\mathbf{J}$,
- the set of β -normal terms of Λ ,
- the set of cut-free derivations in \mathbf{LJ} , denoted by Λ_{cf} ,
- the set of cut-free derivations in \mathbf{LJ}' , denoted by Λ'_{cf} .

3.9. Gentzen's reductions. The analogy between $\Lambda\mathbf{J}$ and \mathbf{LJ} also extends to Gentzen's reduction steps.

To show this, we first translate the two more complicated reduction steps of cut elimination into $\Lambda\mathbf{J}$:

$$\begin{aligned} (t[z := ys])[y := \lambda xr] &= (y(s, z.t))[y := \lambda xr] \\ &= (\lambda xr)(s, z.t) \quad y \notin s, t \\ &\rightarrow_\beta t[z := r[x := s]]. \end{aligned}$$

$$\begin{aligned}
(t'[z' := ys'])[y := t[z := xs]] &= (y(s', z'.t'))[y := x(s, z.t)] \\
&= x(s, z.t)(s', z'.t') && y \notin s', t' \\
&\rightarrow_{\pi} x(s, z.t(s', z'.t')) \\
&= (t(s', z'.t'))[z := sx] \\
&= t'[z' := ts'] [z := xs] \\
&= t'[z' := t[z := xs]s'] && z \notin s'.
\end{aligned}$$

So we get a precise correspondence between the permutations in LJ and ΛJ , as well as between \rightsquigarrow_{β} and \rightarrow_{β} -reductions.

In order to treat the remaining reductions (which by the current embedding are mapped to equalities), we introduce the straightforward extension of ΛJ by explicit substitutions.

3.10. ΛJ_{\square} . Extend the term grammar by the substitution constructor:

$$\Lambda J_{\square} \ni r, s, t ::= x \mid \lambda x r \mid r(s, z.t) \mid r[x := s].$$

The reduction rules capture just the basic implementation of substitution, without permutation of substitutions (assume $x \neq y$):

$$\begin{aligned}
x[x := s] &\rightarrow_{\square} s, \\
x[y := s] &\rightarrow_{\square} x, \\
(\lambda y r)[x := s] &\rightarrow_{\square} \lambda y. r[x := s], && y \notin s, \\
(r(r', z.r''))[x := s] &\rightarrow_{\square} r[x := s](r'[x := s], z.r''[x := s]), \\
(\lambda x r)(s, z.t) &\rightarrow_{\beta_{\square}} t[z := r[x := s]], \\
r(s, z.t)(s', z'.t') &\rightarrow_{\pi} r(s, z.t(s', z'.t')).
\end{aligned}$$

We write $\vec{\Lambda J}_{\square}$ for the typed system, obtained by adding the cut rule (c) to the type assignment system for ΛJ .

Theorem 16. $\vec{\Lambda J}_{\square}$ is strongly normalizing and confluent.

Proof (sketch). Similar to the strong normalization proof for $\vec{\Lambda}_{\square}$ in subsections 2.1–2.3. We only hint on a few more or less obvious changes.

- All proofs of subsections 2.1 and 2.2 go through without change. In the definition of $\#_{\square}$, the clause for generalized application is

$$\#_{\square}(r(s, z.t)) := \#_{\square} r + \#_{\square} s + \#_{\square} t + 1.$$

- Property (A) reads

$$r \Downarrow, s \Downarrow, t \Downarrow \implies r(s, z.t) \Downarrow.$$

The proof is as before.

- In the proof of theorem 6, the case of a β_{\square} -reduction is treated as follows: assume $(\lambda x r)(s, z.t) \rightarrow_{\beta_{\square}} t[z := r[x := s]]$ with $(\lambda x r)(s, z.t) \Downarrow$. Then $r \Downarrow$, $s \Downarrow$, $t \Downarrow$ and by (B) $r[x := s] \Downarrow$. (C) shows $r[x := s] \Downarrow$, so again by (B) $t[z := r[x := s]] \Downarrow$ and the claim follows by (C). \square

Corollary 17 (Strong Hauptsatz).

Cut reduction in LJ is strongly normalizing and confluent.

Proof. All reductions of LJ are simulated by the respective ones in $\vec{A}\vec{J}_\square$. Reduction in $\vec{A}\vec{J}_\square$ is strongly normalizing and confluent. Its normal forms are those of $\vec{A}\vec{J}$ and are isomorphic to derivations in LJ_{cf} up to contractions. \square

4 Sequent Pure Type Systems

Recently, Gutierrez and Ruiz [GR02b] proposed two variants of an extension to Pure Type Systems of the implicit substitution approach to λ -calculus notations for the sequent calculus. The goal of these attempts is to enable sequent-calculus like arguments in theorem provers that are based on Pure Type Systems, such as Coq. This would be promising, since the rule (l) allows manipulation of leaves of derivation trees, whereas the usual natural deduction rules work on the derivation root only.

Unfortunately, it turns out that neither of the systems in [GR02b, GR02a] extends to really dependent Pure Type Systems, in particular, cut elimination fails in the Calculus of Constructions.

In this section, we introduce generalized application for Pure Type Systems to capture the (l) -rule and use the methods of the previous section to derive the equivalence between cut elimination and strong normalization.

4.1. Raw terms. The grammar of *raw terms* is given by

$$r, s, A, B ::= x \mid rs \mid \lambda x : A.r \mid \Pi x : A.B \mid \mathfrak{s}.$$

Here \mathfrak{s} ranges over a given set of sorts \mathcal{S} . $\lambda x : A.r$ and $\Pi x : A.r$ bind x in r . We adopt most of the notational conventions discussed in subsection 1.4.

β -reduction is given by the term closure of elementary β -reduction:

$$(\lambda x : A.r)s \rightarrow_\beta r[x := s].$$

\rightarrow_β is confluent, as can be seen by a simple adaptation of the usual confluence proof for the untyped λ -calculus.

4.2. Pure Type Systems. A Pure Type System is determined by three data: a set \mathcal{S} of *sorts*, a set of pairs $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ (*axioms*) and a set of triples $\mathcal{R} \subset \mathcal{S}^3$ (*rules*).

A *context* Γ is an expression of the form $x_1 : A_1, \dots, x_n : A_n$ with disjoint \vec{x} (the domain of the context, denoted by $\text{dom}\Gamma$). The empty context will be omitted, if reasonable. The notation $\Gamma, x : A$ is used only if $x \notin \text{dom}\Gamma$.

The rules of Pure Type Systems with β -equality (see figure 2) derive judgments of the form $\Gamma \vdash r : A$ with r a raw term.

A *term* is a raw term typable in this system. From now on r, s, t, A, B, C will range over terms and Γ over *legal contexts*, i.e., over $\{\Gamma \mid \exists r, A. \Gamma \vdash r : A\}$.

$(A) \frac{(C, \mathfrak{s}) \in \mathcal{A}}{\vdash C : \mathfrak{s}}$	$(\beta) \frac{\Gamma \vdash r : A \quad \Gamma \vdash A' : \mathfrak{s} \quad A =_{\beta} A'}{\Gamma \vdash r : A'}$
$(w) \frac{\Gamma \vdash r : A \quad \Gamma \vdash B : \mathfrak{s}}{\Gamma, x : B \vdash r : A}$	$(\lambda) \frac{\Gamma, x : A \vdash r : B \quad \Gamma \vdash \Pi x : A. B : \mathfrak{s}}{\Gamma \vdash \lambda x : A. r : \Pi x : A. B}$
$(v) \frac{\Gamma \vdash A : \mathfrak{s}}{\Gamma, x : A \vdash x : A}$	$(a) \frac{\Gamma \vdash r : \Pi x : A. B \quad \Gamma \vdash s : A}{\Gamma \vdash rs : B[x := s]}$
$(II) \frac{(\mathfrak{s}_1, \mathfrak{s}_2, \mathfrak{s}_3) \in \mathcal{R} \quad \Gamma \vdash A : \mathfrak{s}_1 \quad \Gamma, x : A \vdash B : \mathfrak{s}_2}{\Gamma \vdash \Pi x : A. B : \mathfrak{s}_3}$	

Fig. 2. Rules of Pure Type Systems with β -equality

We refer the reader to [Bar92] for the standard properties of Pure Type Systems, which we will use freely. We explicitly mention the substitution property (writing $\Gamma[x := s]$ for substitution in all types of Γ) and subject reduction:

$$(c) \frac{\Gamma, x : A, \Gamma' \vdash r : B \quad \Gamma \vdash s : A}{\Gamma, \Gamma'[x := s] \vdash r[x := s] : B[x := s]} \quad (r) \frac{\Gamma \vdash r : A \quad r \rightarrow_{\beta} r'}{\Gamma \vdash r' : A}$$

Furthermore, one can prove the generation lemma, which essentially allows to invert rules (a), (λ), (v), (A) and (II) — for the instance of application, it states that

$$\Gamma \vdash rs : B \implies \exists A, B'. \Gamma \vdash r : \Pi x : A. B' \ \& \ \Gamma \vdash s : A \ \& \ B =_{\beta} B'[x := s].$$

4.3. Generalized application. The raw terms for Pure Type Systems with generalized application (abbreviated LPTS) are given by the grammar

$$r, s, A, B ::= x \mid r(s, x : A. z : B.t) \mid \lambda x : A. r \mid \Pi x : A. B \mid \mathfrak{s}.$$

In the generalized application, the variable x is bound in B and required not to occur in t ; z is bound in t .

β_j -reduction is formulated as in AJ:

$$(\lambda x : A. r)(s, x : A'. z : B.t) \rightarrow_{\beta_j} t[z := r[x := s]].$$

Remark the possibility of having different types for x in r and x in B . In the typed LPTS we will have $A =_{\beta_j \pi} A'$ thanks to the generation lemma.

In addition to \rightarrow_{β_j} we have *permutations*:

$$r(s, x : A. z : B.t)S \rightarrow_{\pi} r(s, x : A. z : B.tS) \quad \text{with } S := (s', x' : A'. z' : B'. t')$$

where without loss of generality $x, z \notin S$.

As in λJ , we close these elementary reductions under all term formers and write \rightarrow for the union of \rightarrow_{β_j} and \rightarrow_{π} .

The *type assignment calculus* of LPTS is modified only in the rule (a):

$$(a)^\ddagger \frac{\Gamma \vdash r : \Pi x : A.B \quad \Gamma \vdash s : A \quad \Gamma \vdash \Pi x : A. \Pi z : B. C : \mathfrak{s}}{\Gamma \vdash r(s, x : A. z : B.t) : C}$$

The proviso ‡ is quite important: $z \notin C$ & $x \notin t, C$.

We mention without detailing the proof that the substitution property and the generation lemma hold for LPTS.

4.4. Subject reduction. We only verify subject reduction for elementary β_j -reduction and permutation. Using the generation lemma repeatedly, a typing of the β -redex can be brought into the following form:

$$\frac{\frac{\Gamma, x : A \vdash r : B \quad \Gamma \vdash \Pi x : A. B : \mathfrak{s}}{\Gamma \vdash \lambda x : A. r : \Pi x : A. B} \quad \Gamma \vdash s : A \quad \Gamma, x : A, z : B \vdash t : C}{\Gamma \vdash (\lambda x : A. r)(s, x : A, z : B.t) : C}$$

We use the substitution lemma twice to obtain

$$\frac{\Gamma \quad \Gamma \vdash r[x := s] : B[x := s],}{\Gamma, z : B[x := s] \vdash t[x := s] : C[x := s].}$$

Since $x \notin C, t$ by the proviso ‡ on (a), we get

$$\Gamma, z : B[x := s] \vdash t : C$$

To this we apply the substitution lemma once more and get

$$\Gamma \vdash t[z := r[x := s]] : C[z := r[x := s]].$$

By the proviso ‡ , z does not occur in C , so that we obtain

$$\Gamma \vdash t[z := r[x := s]] : C.$$

As for permutation assume

$$\Gamma \vdash r(s, x : A. z : B.t)S : C \quad \text{and} \quad r(s, x : A. z : B.t)S \rightarrow_{\pi} r(s, x : A, z : B.tS)$$

with $S := (s', x' : A', z' : B'.t')$. By the generation lemma for (a) we get typings as follows:

$$\begin{array}{l} \Gamma \quad \Gamma \quad \Gamma, x : A, z : B \quad \Gamma \quad \Gamma, x' : A', z' : B' \\ \vdash r : \Pi x : A. B, \quad \vdash s : A, \quad \vdash t : \Pi x' : A'. B', \quad \vdash s' : A', \\ \vdash t' : C \end{array}$$

with the following provisos due to restrictions on application and permutation:

- (1) $x \notin t, \Pi x' : A'.B', S, C, x', z', z$
- (2) $z \notin \Pi x' : A'.B', S, C, x, x', z'$
- (3) $x' \notin t', C, x, z, z'$
- (4) $z' \notin C, x, z, x'$.

Using weakening, (1) and (2) we obtain

$$\Gamma, x : A, z : B \vdash s' : A' \quad \text{and} \quad \Gamma, x : A, z : B, x' : A', z' : B' \vdash t' : C$$

So applying rule (a) twice (noting the provisos (1)–(4)) we get

$$\Gamma, x : A, z : B \vdash tS : C \quad \text{and} \quad \Gamma \vdash r(s, x : A.z : B.tS) : C.$$

4.5. Embeddings. We use the analogue of the embedding (**) (see subsection 3.5) to embed LPTS into PTS:

$$PTS : \quad r(s, x : A.z : B.t) := (\lambda x : A.\lambda z : B.t)s(rs).$$

Note that if $r(s, x : A.z : B.t) : C$ is typed with (a), then the two abstractions on the right can be typed $\lambda x : A.\lambda z : B.t : \Pi x : A.\Pi x : B.C$ due to the explicit premise of the rule (a).

It should be remarked that the proviso $x \notin t$ of the typing rule guarantees that the simulation of β -reduction

$$\begin{aligned} (\lambda x : A.r)(s, x : A.z : B.t) &= (\lambda x : A.\lambda z : B.t)s((\lambda x : A.r)s) \\ &\rightarrow_{\beta} (\lambda z : B[x := s].t)((\lambda x : A.r)s) \\ &\rightarrow_{\beta} (\lambda z : B[x := s].t)(r[x := s]) \\ &\rightarrow_{\beta} t[z := r[x := s]] \end{aligned}$$

is possible on well-typed terms.

The converse embedding of PTS into LPTS is straightforward:

$$LPTS : \quad rs := r(s, x : A.z : B.z).$$

This definition involves the introduction of arbitrary types A, B . Given a typing of rs , these types arise from the generation lemma and are thus well-defined up to β -equality.

Combining both embeddings we obtain

Theorem 18 (Equivalence). *LPTS and PTS are equiconsistent.*

4.6. Strong cut elimination. All steps of the strong normalization proof for $\vec{A}\vec{J}$ can be adapted to the setting of LPTS. We only highlight some of the differences to the treatment of section 3.

- The π -normal form (see subsection 3.2) is given by

$$\begin{aligned} x^\pi &:= x, \\ (\lambda x : A.r)^\pi &:= \lambda x : A^\pi.r^\pi, \\ (\Pi x : A.B)^\pi &:= \Pi x : A^\pi.B^\pi, \\ (r(s, x : A.z : B.t))^\pi &:= r^\pi @ (s^\pi, x : A^\pi.z : B^\pi.t^\pi). \end{aligned}$$

The definition of @ is straightforward. The commutation lemma 8 holds without major change in its proof.

- As shown in the previous subsection, the analogue of lemma 10 in subsection 3.5 reads

$$r \rightarrow_{\beta_j} s \implies r \rightarrow_{\beta}^3 s.$$

- In the presence of abstracted types, proposition 11 is reformulated to

$$\begin{aligned} x \in r &\implies \#_{\beta}((\lambda x : A.r)s) \leq \#_{\beta}(r[x := s]) + \#_{\beta}A + 1, \\ x \notin r &\implies \#_{\beta}((\lambda x r)s) \leq \#_{\beta}r + \#_{\beta}s + \#_{\beta}A + 1. \end{aligned}$$

- Lemma 12 remains unchanged; its proof is only marginally more complex, since it has to take into account β -reductions on abstracted types.
- Theorem 13 and corollary 14 are verbally the same as in subsection 3.5.

Corollary 19. *A LPTS enjoys strong cut elimination iff the corresponding PTS is strongly β -normalizing.*

Conclusions

The calculus ΛJ provides a uniform framework for combining the features of natural deduction and sequent calculus, syntactically extending both systems without increase in proof-theoretic strength. Our analysis of the relation between Gentzen's reductions in LJ and those in ΛJ (and a fortiori also in Λ) refines previous work in the field and establishes a precise one-to-one correspondence.

The technical simplicity of ΛJ permits a straightforward extension to Pure Type Systems and leads to a general cut elimination theorem for sequent Pure Type Systems, thus improving on previous results by Gutierrez and Ruiz [GR02b]. This allows to incorporate the left-sided rules into theorem provers that are based on Pure Type Systems and in this way enable the use of bottom-up strategies, without having to resort to substitutions on proof terms.

References

- [Bar92] Henk Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.
- [BG97] Roel Bloo and Herman Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211(1–2):375–395, 1997.

- [BG00] Henk P. Barendregt and Sylvia Ghilezan. Lambda terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10:121–134, 2000.
- [DP98] Roy Dyckhoff and Luis Pinto. Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60:107–118, 1998.
- [Gen35] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210,405–431, 1935.
- [GR02a] Francisco Gutierrez and Blas Ruiz. Cut elimination in a class of sequent calculi for pure type systems. Submitted., 2002.
- [GR02b] Francisco Gutierrez and Blas Ruiz. Sequent calculi for pure type systems. Technical Report 06/02, Dep. de Lenguajes y Ciencias de la Computación, Universidad de Malaga (Spain), 2002.
- [Her94] Hugo Herbelin. A lambda calculus structure isomorphic to Gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Proc. 8th CSL '94 (Kazimierz)*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1994.
- [JM00] Felix Joachimski and Ralph Matthes. Standardization and confluence for a lambda calculus with generalized applications. In Leo Bachmair, editor, *Proc. 11th RTA 2000 (Norwich)*, volume 1833 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2000.
- [JM03] Felix Joachimski and Ralph Matthes. Short proofs of normalization for the simply-typed λ -calculus, permutative conversions and Gödel's *T*. *Archive for Mathematical Logic*, 42(1):59–87, 2003.
- [Mat00] Ralph Matthes. Characterizing strongly normalizing terms for a lambda calculus with generalized applications via intersection types. In J.D.P. Rolim et al., editor, *Proc. ICALP 2000 (Geneva)*, volume 8 of *Proceedings in Informatics*, pages 339–353. Carlton Scientific, 2000.
- [Mat01] Ralph Matthes. Interpolation for natural deduction with generalized eliminations. In R. Kahle, P. Schroeder-Heister, and R. Stärk, editors, *Proc. PTCS 2001 (Dagstuhl)*, volume 2183 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 2001.
- [Mel95] Paul-André Mellies. Typed λ -calculi with explicit substitutions may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. 2nd TLCA 1995 (Edinburgh)*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer, 1995.
- [Pot77] Garrel Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977.
- [Pra65] Dag Prawitz. *Natural deduction, a proof-theoretical study*. Almqvist and Wiksell, Stockholm, 1965.
- [Sch99] Helmut Schwichtenberg. Termination of permutative conversions in intuitionistic Gentzen calculi. *Theoretical Computer Science*, 212(1–2):247–260, 1999.
- [vP01] Jan von Plato. Natural deduction with generalized elimination rules. *Archive of Mathematical Logic*, 40(7):541–567, 2001.
- [vRSSX99] Femke van Raamsdonk, Paula Severi, Morten Heine Sørensen, and Hongwei Xi. Perpetual reductions in lambda-calculus. *Information and Computation*, 149(2):173–225, 1999.
- [VW01] René Vestergaard and Joe Wells. Cut rules and explicit substitutions. *Mathematical Structures in Computer Science*, 11(1):131–168, 2001.
- [Zuc74] Jeffrey Zucker. Correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–156, 1974.