Delayed substitutions

José Espírito Santo*

Departamento de Matemática
Universidade do Minho
Portugal
jes@math.uminho.pt

Abstract. This paper investigates an approach to substitution alternative to the implicit treatment of the λ-calculus and the explicit treatment of explicit substitution calculi. In this approach, substitutions are delayed (but not executed) explicitly. We implement this idea with two calculi, one where substitution is a primitive construction of the calculus, the other where substitutions is represented by a β-redex. For both calculi, confluence and (preservation of) strong normalisation are proved (the latter fails for a related system due to Revesz, as we show). Applications of delayed substitutions are of theoretical nature. The strong normalisation result implies strong normalisation for other calculi, like the computational lambda-calculus, lambda-calculi with generalised applications, or calculi of cut-elimination for sequent calculus. We give an investigation of the computational interpretation of cut-elimination in terms of generation, execution, and delaying of substitutions, paying particular attention to how generalised applications improve such interpretation.

1 Introduction

Explicit substitution calculi were introduced as an improvement of the λ-calculus, capable of modelling the actual implementation of functional languages and symbolic systems [1]. However, other applications of theoretical nature were soon recognized, particularly in proof theory, where λ-calculus also fails to give a computational interpretation to sequent calculus and cut-elimination [4, 5, 14].

The basic idea in explicit substitution calculi is the separation between the generation and the execution of substitution. But this idea is operative only if this execution can be delayed. Of course, the mentioned separation gives the opportunity to do something between the generation and the execution of a substitution. But there are situations, for instance in a syntax like that of the $\lambda\mathtt{x}$-calculus [12], where explicit rules for the delaying of substitution are required.

This paper investigates explicit rules for delaying substitution in a syntax similar to $\lambda\mathtt{x}$. However, a first and immediate observation is that explicit delaying cannot be combined with explicit execution without breaking termination. The situation is even worse if we try to implement substitutions as β-redexes (Revesz's idea [11]). So, the system we study, named $\lambda\mathtt{s}$, separates generation

* The author is supported by FCT via Centro de Matematica, Universidade do Minho.

and execution of substitution, but employs implicit execution. In addition, it has permutation rules for achieving the delaying of substitution.

The calculus $\lambda\mathtt{s}$ enjoys good properties, like confluence and (preservation of) strong normalisation. The circumstance of employing implicit substitution disallows direct applications of $\lambda\mathtt{s}$ to the implementation of computational systems. However, $\lambda\mathtt{s}$ has several theoretical uses. Strong normalisation of $\lambda\mathtt{s}$ implies the same property for several calculi, like the computational lambda-calculus [9] and lambda-calculi with generalised applications [6]. Certainly, future work should exploit the use of the calculus for reasoning about programs. In this paper we emphasize applications to proof theory.

We define a sequent calculus LJ with a simple cut-elimination procedure consisting of 3 reduction rules. Then, we show that $\lambda\mathtt{s}$ gives a computational interpretation to the 3 cut-elimination rules of LJ, precisely as rules for the generation, delaying, and execution of substitution. Strong normalisation of $\lambda\mathtt{s}$ is lifted to LJ. We pay particular attention to how generalised applications [6, 15], when combined with delayed substitutions, improve the mentioned interpretation of LJ.

Notations: Types (=formulas) are ranged over by A, B, C and generated from type variables using the "arrow type" (=implication), written $A \supset B$. Contexts Γ are sets of declarations $x : A$ where each variable is declared at most once. Barendregt's variable convention is adopted. In particular, we take renaming of bound variables for granted. Meta substitution is denoted by $[\_/x]\_$. By a *value* we mean a variable or λ-abstraction in the calculus at hand.

2 Delayed substitutions

Motivation: Recall the syntax of the $\lambda\mathtt{x}$-calculus:

$$M, N, P ::= x \mid \lambda x.M \mid MN \mid \langle N/x\rangle M$$

The variable x is bound in M in $\lambda x.M$ and $\langle N/x\rangle M$. The scope of $\lambda x.\_$ and $\langle N/x\rangle\_$ extends to the right as much as possible. There is a reduction rule

$$(\beta)\ (\lambda x.M)N \to \langle N/x\rangle M$$

that generates substitutions and four rules

$$\begin{array}{ll} (\mathtt{x}_1)\ \langle N/x\rangle x \to N & (\mathtt{x}_3)\ \ \langle N/x\rangle MP \to (\langle N/x\rangle M)\langle N/x\rangle P \\ (\mathtt{x}_2)\ \langle N/x\rangle y \to y,\ y \neq x & (\mathtt{x}_4)\ \langle N/x\rangle \lambda y.M \to \lambda y.\langle N/x\rangle M \end{array}$$

for the explicit execution of substitution. By variable convention, $x \neq y$ and $y \notin N$ in rule $(\mathtt{x}_4)$. Let $\mathtt{x} = \cup_{i=1}^{4} \mathtt{x}_i$.

Suppose we want to reduce $Q_0 = (\lambda x.M)NN'$, where $M = \lambda y.P$. After a β-step, we obtain $Q_1 = (\langle N/x\rangle M)N'$. Substitution $\langle N/x\rangle M$ was generated but not immediately executed. This allows the *delaying* of its execution, if we decide to do something else, e.g. reducing N, M, N', or another term in the

program surrounding Q_1. However, we may very well be interested in delaying the execution of $\langle N/x\rangle M$ in another way, namely by applying immediately M to N'. In $\lambda\mathtt{x}$ this may be achieved in some sense, if a step of the execution of $\langle N/x\rangle M$ is performed, yielding $Q_2 = (\lambda y.P')N'$, where $P' = \langle N/x\rangle P$.

This lack of separation between substitution execution and delaying is unsatisfactory. The delaying of $\langle N/x\rangle M$ in Q_1 can be achieved if we adopt a permutation rule that yields $\langle N/x\rangle MN'$, that is $Q_2' = \langle N/x\rangle(\lambda y.P)N'$. However, we cannot add this permutation to the set of $\mathtt{x}$-rules without breaking termination. Suppose we want to reduce $\langle N/x\rangle M_1M_2$, where M_2 is a pure term (i.e. a term without substitutions) and $x \notin M_2$. Then a cycle is easily generated:

$$\begin{array}{rll}
\langle N/x\rangle M_1M_2 & \to_{\mathtt{x}} (\langle N/x\rangle M_1)\langle N/x\rangle M_2 & (\text{by } \mathtt{x}_3)\\
& \to^*_{\mathtt{x}} (\langle N/x\rangle M_1)[N/x]M_2 & (\text{because } M_2 \text{ is pure})\\
& = (\langle N/x\rangle M_1)M_2 & (\text{because } x \notin M_2)\\
& \to \langle N/x\rangle M_1M_2 & (\text{by permutation})
\end{array}$$

(Here $[N/x]M_2$ denotes meta-substitution.) This is why the calculus of delayed substitutions we introduce next does not have $\mathtt{x}$-rules for explicit, stepwise execution of substitution, but instead a single σ-rule for its implicit execution.

The $\lambda\mathtt{s}$-calculus: The terms of $\lambda\mathtt{s}$ are given by:

$$M, N, P, Q ::= x \mid \lambda x.M \mid MN \mid \langle N/x\rangle M$$

This set of terms is equipped with the following reduction rules:

$$\begin{array}{llll}
(\beta) & (\lambda x.M)N \to \langle N/x\rangle M & (\pi_1) & (\langle N/x\rangle M)P \to \langle N/x\rangle MP\\
(\sigma) & \langle N/x\rangle M \to [N/x]M & (\pi_2) & \langle\langle N/x\rangle P/y\rangle M \to \langle N/x\rangle\langle P/y\rangle M
\end{array}$$

where meta-substitution $[N/x]M$ is defined as expected. In particular

$$[N/x]\langle P/y\rangle M = \langle [N/x]P/y\rangle [N/x]M \ . \tag{1}$$

By variable convention, $x \neq y$ in π_2 and (1). For the same reason, $y \notin N$ in (1).

Let $\pi = \pi_1 \cup \pi_2$. The choice of permutations π_1 and π_2 is pragmatic. These are the permutations appropriate for the applications of the calculus to be shown in this paper. It is natural that, if the applications of the calculus are different, other rules for pulling out substitutions in other contexts are useful and needed.

By a *typable* term $M \in \lambda$ or $M \in \lambda\mathtt{s}$ we mean a term that has a simple type A, given a context Γ assigning types to the free variable of M. This relation is written $\Gamma \vdash M : A$ and generated by the set of usual rules for assigning simple types to variables, abstraction and application (which we omit), plus the typing rule of substitution:

$$\frac{\Gamma \vdash N : A \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash \langle N/x\rangle M : B}$$

Natural relationship with the λ-calculus: The study of the natural interpretation of $\lambda\mathtt{s}$ in λ yields easily a proof of confluence for $\lambda\mathtt{s}$. First, $\lambda\mathtt{s}$ simulates β-reduction.

Proposition 1. *If $M \to_\beta N$ in λ then $M \to_\beta P \to_\sigma N$ in $\lambda\mathtt{s}$, for some P.*

Conversely, let $(\_)^\natural : \lambda\mathtt{s} \to \lambda$ be defined as follows: $x^\natural = x$, $(MN)^\natural = M^\natural N^\natural$, $(\lambda x.M)^\natural = \lambda x.M^\natural$, and $(\langle N/x\rangle M)^\natural = [N^\natural/x]M^\natural$.

Proposition 2. *(1) If $M \to_\beta N$ in $\lambda\mathtt{s}$ then $M^\natural \to_\beta^* N^\natural$ in λ. (2) If $M \to_{\pi\sigma} N$ in $\lambda\mathtt{s}$ then $M^\natural = N^\natural$. (3) In $\lambda\mathtt{s}$, $M \to_\sigma^* M^\natural$.*

Proposition 3 (Confluence). *Let $R \in \{\pi_1, \pi_2, \pi\}$. Then $\to_{\beta\sigma R}$ is confluent in $\lambda\mathtt{s}$.*

Proof: By confluence of β-reduction in λ and Propositions 1 and 2. ■

We will prove strong normalisation of $\lambda\mathtt{s}$ as a corollary to strong normalisation of another calculus of delayed substitutions. The terms of the latter are the ordinary λ-terms, where β-redexes are regarded as substitutions. The first author to develop this idea was G. Revesz, see for instance [11].

Revesz's system: G. Revesz proposed to replace in the λ-calculus the ordinary β-rule and the related calls to meta-substitution by a set of local transformation rules. These local rules correspond to the explicit, stepwise execution of substitution.

$$\begin{array}{ll} (\beta_1)\ (\lambda x.x)N \to N & (\beta_3)\ (\lambda x.\lambda y.M)N \to \lambda y.(\lambda x.M)N \\ (\beta_2)\ (\lambda x.y)N \to y,\ y \neq x & (\beta_4)\quad (\lambda x.MP)N \to (\lambda x.M)N((\lambda x.P)N) \end{array}$$

Let $\mathtt{R} = \cup_{i=1}^4 \beta_i$. By variable convention, $x \neq y$ and $y \notin N$ in β_3. A basic property of Revesz's system is that

$$(\lambda x.M)N \to_{\mathtt{R}}^* [N/x]M \ . \tag{2}$$

Now, we have seen that in a syntax with a primitive substitution construction, we cannot combine explicit substitution execution and delaying without breaking termination. When substitution is represented by β-redexes, the situation is even worse, as substitution execution alone breaks termination.

Theorem 1. *There is a typed λ-term Q such that Q is not $\mathtt{R} - SN$.*

Proof: Let $Q = (\lambda x.(\lambda y.M)N)P$. We underline the successive redexes.

$$\begin{array}{lll} Q & = & \underline{(\lambda x.(\lambda y.M)N)P} \\ & \to_{\beta_4} & \underline{(\lambda x.\lambda y.M)P((\lambda x.N)P)} = Q' \\ & \to_{\beta_3} & \underline{(\lambda y.(\lambda x.M)P)((\lambda x.N)P)} \\ & \to_{\beta_4} & (\lambda y.\lambda x.M)((\lambda x.N)P)\underline{((\lambda y.P)((\lambda x.N)P))} \\ & \to_{\mathtt{R}}^* & (\lambda y.\lambda x.M)((\lambda x.N)P)\underline{[(\lambda x.N)P/y]P} \qquad \text{(by (2))} \\ & = & \underline{(\lambda y.\lambda x.M)((\lambda x.N)P)}P \qquad (\text{as } y \notin P) \\ & \to_{\beta_3} & \underline{(\lambda x.(\lambda y.M)((\lambda x.N)P))}P \\ & \to_{\beta_4} & \underline{(\lambda x.\lambda y.M)P((\lambda x.((\lambda x.N)P))P)} \\ & \to_{\mathtt{R}}^* & (\lambda x.\lambda y.M)P\underline{[P/x]((\lambda x.N)P)} \qquad \text{(by (2))} \\ & = & (\lambda x.\lambda y.M)P((\lambda x.N)P) \qquad (\text{as } x \notin ((\lambda x.N)P)) \\ & = & Q' \qquad \blacksquare \end{array}$$

So, one has to give up the idea of explicitly executing substitution within the syntax of the λ-calculus. But we can do explicit delaying.

Delaying of substitution in the λ-calculus: In the λ-calculus, define $\pi = \pi_1 \cup \pi_2$, where:

$$\begin{array}{ll} (\pi_1) & (\lambda x.M)NP \to (\lambda x.MP)N \\ (\pi_2) & M((\lambda x.P)N) \to (\lambda x.MP)N \end{array}$$

In both redexes, M "wants" to be applied to P, but something forbids this application, namely the fact that one of M or P is inside a β-redex (or "substitution"). The rules rearrange the term so that the "substitution" is delayed. We denote the calculus consisting of β and π as $\lambda[\beta\pi]$. Notice that π_1 is one of Regnier's σ-rules [10].

Proposition 4. *In λ, $\to_{\beta\pi}$ is confluent, but $\to_\pi$ is not.*

Proof: Notice that $(\lambda x.M)NP =_\beta (\lambda x.MP)N$ and $M((\lambda y.P)N) =_\beta (\lambda y.MP)N$. So, confluence of $\to_{\beta\pi}$ follows from confluence of $\to_\beta$. On the other hand, $(\lambda x.M)N((\lambda y.P)Q)$ π-reduces to both $(\lambda x.(\lambda y.MP)Q)N$ and $(\lambda y.(\lambda x.MP)N)Q$, which can easily be two π-nfs. ■

Define $|M|$, the *size* of λ-term M, as follows: $|x| = 1$; $|\lambda x.M| = 1 + |M|$; $|MN| = 1 + |M| + |N|$.

Proposition 5. *In λ, $\to_\pi$ is terminating.*

Proof: The termination of $\to_{\pi_1}$ is in [10]. As to the remaining cases, define $w(M)$, the *weight* of a λ-term M, as follows: $w(x) = 0$; $w(\lambda x.M) = w(M)$; $w(MN) = |N| + w(M) + w(N)$. It holds that, if $M \to_{\pi_1} N$, then $w(M) = w(N)$; and that, if $M \to_{\pi_2} N$, then $w(M) > w(N)$. The proposition now follows. ■

Let M be a λ-term such that M is β-SN. Define $||M||_\beta$ to be the maximal length of β-reduction sequences starting from M.

Proposition 6. *Let $M \to_\pi N$. If M is β-SN, then so is N and $||M||_\beta \geq ||N||_\beta$.*

Proof: For π_1, [10] proves $||M||_\beta = ||N||_\beta$. For π_2, the argument is a slight generalisation of an argument in [7], and uses the fact that, for M, N λ-terms,

$$x \in FV(M) \Rightarrow ||(\lambda x.M)N||_\beta \leq ||[N/x]M||_\beta + 1 \quad (3)$$

$$x \notin FV(M) \Rightarrow ||(\lambda x.M)N||_\beta \leq ||M||_\beta + ||N||_\beta + 1 \ . \quad (4)$$

This is the so called "fundamental lemma of perpetuality"[1]. Let $Q_0 = M((\lambda x.P)N)$ and $Q_1 = (\lambda x.MP)N$. If $x \in P$, then $||Q||_\beta \geq 1 + ||M([N/x]P)||_\beta \geq ||Q_1||_\beta$.

[1] One immediate consequence of this fact is that if (i) $(\lambda x.M)N \notin \beta - SN$ and (ii) $N \in \beta - SN$ when $x \notin FM(M)$, then $[N/x]M \notin \beta - SN$. It is this latter fact that is called "fundamental lemma of perpetuality" in [13].

The first inequality is by $Q_0 \to_\beta M([N/x]P)$ and the second by (3). If $x \notin P$, then $||Q||_\beta \geq 1 + ||N||_\beta + ||MP||_\beta \geq ||Q_1||_\beta$. The first inequality is by $Q_0 \to_\beta^k M((\lambda x.P)N') \to_\beta MP$ (where $k = ||N||_\beta$) and the second by (4). ■

Theorem 2 (SN and PSN). *If $M \in \lambda$ is β-SN (in particular, if M is typable) then M is $\beta\pi$-SN.*

Proof: From Propositions 5 and 6. ■

Sharper relationship with the λ-calculus: Let $(\_)^\sharp : \lambda\mathtt{s} \to \lambda[\beta\pi]$ be defined as follows: $x^\sharp = x$, $MN^\sharp = M^\sharp N^\sharp$, $(\lambda x.M)^\sharp = \lambda x.M^\sharp$, and $(\langle N/x\rangle M)^\sharp = (\lambda x.M^\sharp)N^\sharp$. Hence, mapping $(\_)^\sharp$ "raises" substitutions to β-redexes.

Proposition 7. *(1) If $M \to_\beta N$ in $\lambda\mathtt{s}$ then $M^\sharp = N^\sharp$. (2) If $M \to_{\pi_i} N$ in $\lambda\mathtt{s}$ then $M^\sharp \to_{\pi_i} N^\sharp$ in $\lambda[\beta\pi]$ ($i = 1, 2$). (3) If $M \to_\sigma N$ in $\lambda\mathtt{s}$ then $M^\sharp \to_\beta N^\sharp$ in $\lambda[\beta\pi]$.*

Proposition 8. *In $\lambda\mathtt{s}$, $\to_{\beta\pi}$ is terminating.*

Proof: From termination of $\to_\beta$ in $\lambda\mathtt{s}$, parts 1. and 2. of Proposition 7 and Proposition 5. ■

Proposition 9. *Let $M \in \lambda\mathtt{s}$ and suppose $M^\sharp$ is β-SN. Then M is $\beta\pi\sigma$-SN.*

Proof: From Propositions 7, 8 and 6. ■

Theorem 3 (SN and PSN).

1. *If $M \in \lambda\mathtt{s}$ is typable then M is $\beta\pi\sigma$-SN.*
2. *If $M \in \lambda$ is β-SN then, in $\lambda\mathtt{s}$, M is $\beta\pi\sigma$-SN.*

Proof: 1. Suppose $M \in \lambda\mathtt{s}$ is typable. Then $M^\sharp$ is typable, because $(\_)^\sharp$ preserves typability. Hence $M^\sharp$ is β-SN, by strong normalisation of the simply typed λ-calculus. By Proposition 9, M is $\beta\pi\sigma$-SN.

2. If $M \in \lambda$, then $M^\sharp = M$. Now apply Proposition 9. ■

3 Related calculi

Substitution is a natural interpretation for let-expressions and generalised applications. These interpretations allow strong normalisation of $\lambda\mathtt{s}$ to be transferred to the computational λ-calculus $\lambda_\mathbf{C}$ [8] and to the ΛJ-calculus [6].

Computational λ-calculus: Its terms are given by:

$$M, N, P ::= x \mid \lambda x.M \mid MN \mid \mathsf{let}\, x = N \,\mathsf{in}\, M \ .$$

It was proved in [9] that strong normalisation for Moggi's original reduction rules is a consequence of strong normalisation for the following restricted set of rules (where V stands for a value):

$$\begin{array}{ll}(C_1) & (\lambda x.M)V \to [V/x]M \\ (C_2) & \mathsf{let}\, x = V \,\mathsf{in}\, M \to [V/x]M \\ (C_3) & \mathsf{let}\, x = M \,\mathsf{in}\, x \to M \\ (C_4) & \mathsf{let}\, y = \mathsf{let}\, x = P \,\mathsf{in}\, M \,\mathsf{in}\, N \to \mathsf{let}\, x = P \,\mathsf{in}\, \mathsf{let}\, y = M \,\mathsf{in}\, N \\ (\eta_v) & \lambda x.Vx \to V,\ x \notin V \end{array}$$

Let $C = \cup_{i=1}^{4} C_i$. Checking [9] again one sees that strong normalisation of $\to_C$ (η_v dropped) is sufficient for strong normalisation of $\lambda_{\mathbf{C}}$ with η_v omitted.

Now, instead of mapping the restricted calculus to the linear λ-calculus (as in [9]), we simply interpret into $\lambda\mathtt{s}$, reading $\mathsf{let}\, x = N \,\mathsf{in}\, M$ as $\langle N/x\rangle M$. With this interpretation, C_2 and C_3 are particular cases of σ, C_4 is π_2, and C_1 is β followed by σ. Thus strong normalisation of $\lambda\mathtt{s}$ implies strong normalisation of $\to_C$, and, therefore, of $\lambda_{\mathbf{C}}$ with η_v omitted.

λ-calculus with generalised application: The system ΛJ of [6] is renamed here as $\lambda\mathbf{g}$. Terms of $\lambda\mathbf{g}$ are given by

$$M, N, P ::= x \mid \lambda x.M \mid M(N, x.P) \ .$$

The typing rule for generalized application is

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash M(N, x.P) : C}\ gElim$$

The λg-calculus has two reduction rules:

$$\begin{array}{ll}(\beta) & (\lambda x.M)(N, y.P) \to [[N/x]M/y]P \\ (\pi) & M(N, x.P)(N', y.P') \to M(N, x.P(N', y.P')) \ . \end{array}$$

The natural mapping $(\_)^* : \lambda\mathbf{g} \to \lambda\mathtt{s}$ is given by $x^* = x$, $(\lambda x.M)^* = \lambda x.M^*$, and $(M(N, x.P))^* = \langle M^* N^*/x\rangle P^*$. This mapping gives a strict simulation (one step mapped to one or more steps). Here is the simulation of π.

$$\begin{array}{rcl}(M_0(N_1, x.P_1)(N_2, y.P_2))^* & = & \langle(\langle M_0^* N_1^*/x\rangle P_1^*)N_2^*/y\rangle P_2^* \\ & \to_{\pi_1} & \langle\langle M_0^* N_1^*/x\rangle P_1^* N_2^*/y\rangle P_2^* \\ & \to_{\pi_2} & \langle M_0^* N_1^*/x\rangle\langle P_1^* N_2^*/y\rangle P_2^* \\ & = & (M_0(N_1, x.P_1(N_2, y.P_2)))^* \ . \end{array}$$

So, strong normalisation of $\lambda\mathtt{s}$ implies strong normalisation of $\lambda\mathbf{g}$.

4 Applications to proof theory

Summary of the section: The λ-calculus is the computational interpretation of natural deduction (in the setting of intuitionistic implicational logic). A λ-term is assigned to each natural deduction by the Curry-Howard correspondence,

so that the interpretation of normalisation is β-reduction. There is a traditional assignment $(\_)^\flat$ of λ-terms to sequent calculus derivations, but this assignment fails to give a computational interpretation to the process of cut-elimination. We try an obvious assignment $(\_)^\triangledown$ of $\lambda\mathtt{s}$-terms, but only an optimization $(\_)^\blacktriangledown$ of the latter gives a computational interpretation in terms of generation, execution and delaying of substitution. As a by product, we lift strong normalisation of $\lambda\mathtt{s}$ to sequent calculus. The need for the mentioned optimization is caused by a problem of "imperfect substitution" in sequent calculus, which does not show up when translating sequent calculus with $(\_)^\flat$. A tool that we use in analyzing, and in some sense overcoming, this problem is the calculus $\lambda\mathtt{gs}$, a calculus with generalised application and a primitive substitution construction. Figure 1 shows the systems and mappings studied in this section.

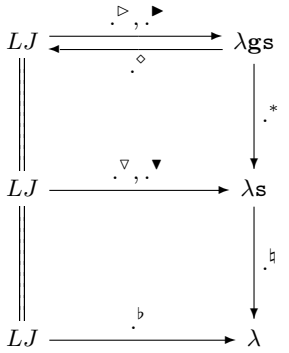

Fig. 1. Sequent calculus and delayed substitution

The calculus LJ: Sequent calculus derivations are represented by:

$$L ::= \mathsf{Axiom}(x) \mid \mathsf{Left}(y, L, (x)L) \mid \mathsf{Right}((x)L) \mid \mathsf{Cut}(L, (y)L)$$

Typing rules are as follows:

$$\frac{}{\Gamma, x : A \vdash \mathsf{Axiom}(x) : A}\ Axiom \qquad \frac{\Gamma \vdash L_1 : A \quad \Gamma, x : A \vdash L_2 : C}{\Gamma \vdash \mathsf{Cut}(L_1, (x)L_2) : C}\ Cut$$

$$\frac{\Gamma \vdash L_1 : A \quad \Gamma, x : B \vdash L_2 : C}{\Gamma, y : A \supset B \vdash \mathsf{Left}(y, L_1, (x)L_2) : C}\ Left \qquad \frac{\Gamma, x : A \vdash L : B}{\Gamma \vdash \mathsf{Right}((x)L) : A \supset B}\ Right$$

where $x \notin \Gamma$ in $Left$, $Right$ and Cut.

Cut elimination in LJ follows the t-protocol of [2]. If a cut is right-permutable, perform its complete, upward, right permutation. This is the first structural step ($S1$). If a cut is not right-permutable, but is left-permutable, perform its complete, upward, left permutation. This is the second structural step ($S2$). If a cut

is neither right-permutable, nor left-permutable, then it is a logical cut (both cut formulas main in the premisses). In that case, apply the logical step of cut-elimination (Log), generating cuts with simpler cut-formula.

Let the predicate $mll(x, L)$ (read "x is main in a linear left introduction L") be defined by: $mll(x, L)$ iff there are L_1, y, L_2 such that $L = \mathsf{Left}(x, L_1, (y)L_2)$ and $x \notin L_1, L_2$. The reduction rules of LJ are as follows (where Axiom, Left, Right and Cut are abbreviated by A, L, R and C, respectively):

$$
\begin{array}{ll}
(S1) & \mathsf{C}(L_1, (x)L_2) \to S1(L_1, x, L_2) \\
(S20) & \mathsf{C}(\mathsf{A}(y), (x)\mathsf{L}(x, L_1', (y')L_2')) \to \mathsf{L}(y, L_1', (y')L_2') \\
(S21) & \mathsf{C}(\mathsf{L}(z, L_1, y, L_2), (x)\mathsf{L}(x, L_1', (y')L_2')) \to \mathsf{L}(z, L_1, y, S2(L_2, x, L_1', y', L_2')) \\
(S22) & \mathsf{C}(\mathsf{C}(L_1, (y)L_2), (x)\mathsf{L}(x, L_1'(y')L_2')) \to \mathsf{C}(L_1, y, S2(L_2, x, L_1', y', L_2')) \\
(Log) & \mathsf{C}(\mathsf{R}((y)L_1), (x)\mathsf{L}(x, L_1', (y')L_2')) \to \mathsf{C}(\mathsf{C}(L_1', (y)L_1), (y')L_2'))
\end{array}
$$

Provisos: in $S1$, not $mll(x, L_2)$; in the remaining rules, $x \notin L_1', L_2'$. We let $S2 = \cup_{i=0}^{2} S2i$. The meta-operations $S1$ and $S2$ are given by:

$$
\begin{array}{rl}
S1(L, x, \mathsf{Axiom}(x)) & = L \\
S1(L, x, \mathsf{Axiom}(y)) & = \mathsf{Axiom}(y),\ y \neq x \\
S1(L, x, \mathsf{Left}(x, L', (z)L'')) & = \mathsf{Cut}(L, (x)\mathsf{Left}(x, S1(L, x, L'), (z)S1(L, x, L'')) \\
S1(L, x, \mathsf{Left}(y, L', (z)L'')) & = \mathsf{Left}(y, S1(L, x, L'), (z)S1(L, x, L'')),\ y \neq x \\
S1(L, x, \mathsf{Right}((y)L')) & = \mathsf{Right}((y)S1(L, x, L')) \\
S1(L, x, \mathsf{Cut}(L', (y)L'')) & = \mathsf{Cut}(S1(L, x, L'), (y)S1(L, x, L''))
\end{array}
$$

$$
\begin{array}{rl}
S2(\mathsf{Axiom}(y), x, L_1', y', L_2') & = \mathsf{Left}(y, L_1', y', L_2') \\
S2(\mathsf{Left}(y, L_1, (z)L_2), x, L_1', y', L_2') & = \mathsf{Left}(y, L_1, (z)S2(L_2, x, L_1', y', L_2')) \\
S2(\mathsf{Right}((y)L'), x, L_1', y', L_2') & = \mathsf{Cut}(\mathsf{Right}((y)L'), (x)\mathsf{Left}(x, L_1', y', L_2')) \\
S2(\mathsf{Cut}(L_1, (y)L_2), x, L_1', y', L_2') & = \mathsf{Cut}(L_1, (y)S2(L_2, x, L_1', y', L_2'))
\end{array}
$$

Traditional assignment: The mapping $(\_)^\flat : LJ \to \lambda$ is defined by

$$
\begin{array}{ll}
\mathsf{Axiom}(x)^\flat = x & \mathsf{Right}((x)L)^\flat = \lambda x.L^\flat \\
\mathsf{Left}(y, L_1, (x)L_2)^\flat = [yL_1^\flat/x]L_2^\flat & \mathsf{Cut}(L_1, (x)L_2)^\flat = [L_1^\flat/x]L_2^\flat
\end{array}
$$

Under this mapping, some parts of cut-elimination are translated as β-reduction, but others receive no interpretation.

Proposition 10. *Let $R = Log$ (resp. $R \in \{S1, S2\}$). If $L_1 \to_R L_2$ in LJ, then $L_1^\flat \to_\beta L_2^\flat$ in λ (resp. $L_1^\flat = L_2^\flat$).*

A first attempt to overcome this situation is to consider the assignment $(\_)^\triangledown : LJ \to \lambda\mathsf{s}$, which generates substitutions where $(\_)^\flat$ calls meta-substitution:

$$
\begin{array}{ll}
\mathsf{Axiom}(x)^\triangledown = x & \mathsf{Right}((x)L)^\triangledown = \lambda x.L^\triangledown \\
\mathsf{Left}(y, L_1, (x)L_2)^\triangledown = \langle yL_1^\triangledown/x\rangle L_2^\triangledown & \mathsf{Cut}(L_1, (x)L_2)^\triangledown = \langle L_1^\triangledown/x\rangle L_2^\triangledown
\end{array}
$$

This mapping reveals a problem that was concealed by $(\_)^\flat$.

"Imperfect substitution" in LJ: Let $L_0 = \mathsf{Cut}(L_1, (x)\mathsf{Left}(x, L_2, (y)L_3))$ and $L_4 = S1(L_1, x, \mathsf{Left}(x, L_2, (y)L_3)) = \mathsf{Cut}(L_1, x, \mathsf{Left}(x, L_2', (y)L_3'))$, where $L_i' = S1(L_1, x, L_i)$, $i = 2, 3$, and x is a free variable of L_2 or L_3. Then $L_0 \to_{S1} L_4$. For $i = 2, 3$, let $P_i = [L_1^\flat/x]L_i^\flat$. Then $L_0^\flat = [L_1^\flat/x][xL_2^\flat/y]L_3^\flat = [L_1^\flat P_2/y]P_3 = L_4^\flat$, the latter equality following by $P_i = L_i'^\flat$, $i = 2, 3$.

Now consider $(\_)^\nabla$ instead. Let $N_i = [L_1^\nabla/x]L_i^\nabla$, $i = 2, 3$. On the one hand $L_0^\nabla = \langle L_1^\nabla/x\rangle\langle xL_2^\nabla/y\rangle L_3^\nabla \to_\sigma [L_1^\nabla/x]\langle xL_2^\nabla/y\rangle L_3^\nabla = \langle L_1^\nabla N_2/y\rangle N_3 = M$, say. On the other hand $L_4^\nabla = \langle L_1^\nabla/x\rangle\langle xL_2'^\nabla/y\rangle L_3'^\nabla$. We would have liked $L_4^\nabla = M$, but L_4^∇ is at least a σ-step behind: $L_4^\nabla \to \langle L_1^\nabla L_2'^\nabla/y\rangle L_3'^\nabla$. Unfortunately, these missing σ-steps propagate recursively, and we can expect $L_i'^\nabla \to_\sigma^+ N_i$.

Why is L_4^∇ a σ-step behind? Because, going back to L_4, $S1$ is an imperfect substitution operator, which did not replace the free, head occurrence of x in $\mathsf{Left}(x, L_2, (y)L_3)$ by L_1. Instead, a cut is generated which $(\_)^\nabla$ translates as the substitution $\langle L_1^\nabla/x\rangle\_$. On the other hand, in $[L_1^\nabla/x]\langle xL_2^\nabla/y\rangle L_3^\nabla$ the free, head occurrence of x in $\langle xL_2^\nabla/y\rangle L_3^\nabla$ is indeed replaced by L_1^∇. These mismatches are not visible if meta-substitution is employed everywhere.

One way out is to extend LJ to a calculus where any term can enter the head position of $\mathsf{Left}(\_, L_2, (y)L_3)$ (see $\lambda\mathtt{gs}$ later on). For now, we optimize $(\_)^\nabla$ by performing the missing σ-steps at "compile time".

Normalisation in $\lambda\mathtt{s}$ versus cut-elimination in LJ: We introduce mapping $(\_)^\blacktriangledown : LJ \to \lambda\mathtt{s}$, which is defined exactly as $(\_)^\nabla$, except for the clause for cuts, which now reads:

$$\mathsf{Cut}(L_1, (x)\mathsf{Left}(x, L_2, (y)L_3))^\blacktriangledown = \langle L_1^\blacktriangledown L_2^\blacktriangledown/y\rangle L_3^\blacktriangledown, \text{ if } x \notin L_2, L_3$$
$$\mathsf{Cut}(L_1, (x)L_2)^\blacktriangledown = \langle L_1^\blacktriangledown/x\rangle L_2^\blacktriangledown, \text{ if } \neg mll(x, L_2)$$

Mapping $(\_)^\blacktriangledown$ has better properties than mapping $(\_)^\nabla$ as to preservation of reduction, but it introduces a typical identification. Suppose $x \notin L_2, L_3$ and let $L_0 = \mathsf{Cut}(z, (x)\mathsf{Left}(x, L_2, (y)L_3))$ and $L_4 = \mathsf{Left}(z, L_2, (y)L_3)$. Notice that $L_0 \to_{S20} L_4$, and that $L_0^\blacktriangledown$ and $L_1^\blacktriangledown$ are the same $\lambda\mathtt{s}$-term of the form $\langle zN_2/y\rangle N_3$.

We now obtain for $\lambda\mathtt{s}$ a result that improves Proposition 10. In order to achieve a good correspondence, we need to introduce in $\lambda\mathtt{s}$ an "eager" version of π, since the structural steps of cut-elimination in LJ perform *complete* permutations of cuts. First, we define certain contexts:

$$\mathcal{S} ::= \langle N/x\rangle[] \mid \langle N/x\rangle\mathcal{S}$$

Each $\mathcal{S}$ is a $\lambda\mathtt{s}$-term with a hole $[]$. $\mathcal{S}[P]$ denotes the result of filling P in the hole of $\mathcal{S}$. Next, "eager" π is defined by

$$(\pi') \qquad \langle\mathcal{S}[V]N/x\rangle P \to \mathcal{S}[\langle VN/x\rangle P] \ ,$$

where V is a value. It is easy to show that each π'-step corresponds to a sequence of one or more π-steps.

Theorem 4 (Computational interpretation of cut-elimination). *Let $R \in \{S1, S21, S22, Log\}$. If $L_1 \to_R L_2$ (resp. $L_1 \to_{S20} L_2$) in LJ, then $L_1^{\blacktriangledown} \to^+_{\beta\pi\sigma} L_2^{\blacktriangledown}$ in $\lambda\mathtt{s}$ (resp. $L_1^{\blacktriangledown} = L_2^{\blacktriangledown}$). In addition, $(\_)^{\blacktriangledown}$ maps a reduction sequence ρ in LJ from L_1 to L_2 to a reduction sequence $\rho^{\blacktriangledown}$ in $\lambda\mathtt{s}$ from $L_1^{\blacktriangledown}$ to $L_2^{\blacktriangledown}$ in a, so to say, structure-preserving way. To each R-step in ρ, there is a corresponding R'-steps in $\rho^{\blacktriangledown}$, where R' is given according to the following table*

R	R'	*computational interpretation*
$S1$	σ	*execution of substitution*
$S21 \cup S22$	π'	*delaying of substitution*
Log	β	*generation of substituion*

Moreover these R'-steps in $\rho^{\blacktriangledown}$ may be interleaved with trivial σ-steps of the form

$$\langle N_1/x\rangle\langle xN_2/y\rangle N_3 \to_\sigma \langle N_1N_2/y\rangle N_3 \quad (x \notin N_2, N_3) \ . \tag{5}$$

The proof is postponed. It is useful to introduce here a new calculus $\lambda\mathbf{gs}$. By studying $\lambda\mathbf{gs}$, we will obtain a proof and two improvements of this theorem.

The $\lambda\mathbf{gs}$-calculus: The $\lambda\mathbf{gs}$-calculus is simultaneously an extension of $\lambda\mathbf{g}$ with a primitive substitution constructor, written $\langle N/x\rangle M$, and an extension of $\lambda\mathtt{s}$ where application is generalised. Reduction rules are as follows:

$$\begin{array}{ll}
(\beta) & (\lambda x.M)(N, y.P) \to \langle\langle N/x\rangle M/y\rangle P \\
(\sigma) & \langle N/x\rangle M \to [N/x]M \\
(\pi_1) & (\langle M/x\rangle N)(N', y.P') \to \langle M/x\rangle(N(N', y.P')) \\
(\pi_2) & \langle\langle M/x\rangle N/y\rangle P \to \langle M/x\rangle\langle N/y\rangle P \\
(\pi_3) & M(N, x.P)(N', y.P') \to M(N, x.P(N', y.P')) \\
(\pi_4) & \langle M(N, x.P)/y\rangle P' \to M(N, x.\langle P/y\rangle P') \ .
\end{array}$$

Meta-substitution in $\lambda\mathbf{gs}$ is defined as expected. In particular, equation (1) holds again. Let $\pi = \cup_{i=0}^4 \pi_i$. A $\lambda g\mathbf{s}$-term is in $\beta\pi\sigma$-normal form iff it is in $\beta\pi_3\sigma$-normal form iff it is a λg-term in $\beta\pi$-normal form iff it has no occurrences of substitution and every occurrence of generalised application in it is of the form $x(N, y, P)$.

Mappings between $\lambda\mathbf{gs}$ and LJ: There is an obvious injection of LJ into $\lambda\mathbf{gs}$. Formally, we define the mapping $(\_)^{\triangleright} : LJ \to \lambda\mathbf{gs}$ as follows:

$$\begin{array}{ll}
\mathsf{Axiom}(x)^{\triangleright} = x & \mathsf{Right}((x)L)^{\triangleright} = \lambda x.L^{\triangleright} \\
\mathsf{Left}(y, L_1, (x)L_2)^{\triangleright} = y(L_1^{\triangleright}, x.L_2^{\triangleright}) & \mathsf{Cut}(L_1, (x)L_2)^{\triangleright} = \langle L_1^{\triangleright}/x\rangle L_2^{\triangleright}
\end{array}$$

Hence, we may regard $\lambda\mathbf{gs}$ as an extension of LJ, where the particular case $y(N, x.P)$ of the generalised application construction plays the role of left introduction. Conversely, $M(N, x.P)$ may be regarded, when mapping back to LJ, as a primitive construction for a particular case of cut (except in the case of M being a variable). The mapping $(\_)^{\diamond} : \lambda\mathbf{gs} \to LJ$ embodies this idea:

$$\begin{array}{ll}
x^{\diamond} = \mathsf{Axiom}(x) & (\lambda x.M)^{\diamond} = \mathsf{Right}((x)M^{\diamond}) \\
(y(N, x.P))^{\diamond} = \mathsf{Left}(y, N^{\diamond}, (x)P^{\diamond}) & (\langle N/x\rangle M)^{\diamond} = \mathsf{Cut}(N^{\diamond}, (x)M^{\diamond}) \\
(M(N, x.P))^{\diamond} = \mathsf{Cut}(M^{\diamond}, (z)\mathsf{Left}(z, N^{\diamond}, (x)P^{\diamond})), \text{ if } M \text{ is not a variable}
\end{array}$$

with $z \notin N, P$ in the last equation.

Lemma 1. *(1) For all $L \in LJ$: (i) $L^{\triangleright\diamond} = L$. (ii) If L is cut-free, then $L^{\triangleright}$ is $\beta\pi\sigma$-normal. (2) For all $M \in \lambda g\mathtt{s}$, if M is $\beta\pi\sigma$-normal, then $M^{\diamond}$ is cut-free and $M^{\diamond\triangleright} = M$.*

Corollary 1. *$(\_)^{\triangleright}$ and $(\_)^{\diamond}$ are mutually inverse bijections between the sets of cut-free terms of LJ and the $\beta\pi\sigma$-normal $\lambda g\mathtt{s}$-terms.*[2]

If cuts are allowed, we cannot expect a bijective correspondence. Suppose $x \notin N_2, N_3$ and N_1 is not a variable. Let $M_1 = \langle N_1/x\rangle(x(N_2, y.N_3))$ and $M_2 = N_1(N_2, y.N_3)$. Then, $M_1^{\diamond}$ and $M_2^{\diamond}$ are the same term L, a cut of the form $\mathsf{Cut}(L_1, (x)\mathsf{Left}(x, L_2, (y)L_3))$. Notice that $M_1 \to_\sigma M_2$, by a *sigma*-step of the restricted form

$$\langle N_1/x\rangle(x(N_2, y.N_3)) \to_\sigma N_1(N_2, y.N_3) \ \ (x \notin N_2, N_3). \tag{6}$$

Now, when mapping cut L back to $\lambda\mathbf{gs}$, there is, so to say, the M_1 option and the M_2 option. Mapping $(\_)^{\triangleright}$ corresponds to the first option, whereas the second option corresponds to a refinement of $(\_)^{\triangleright}$ named $(\_)^{\blacktriangleright}$. This last mapping is defined exactly as $(\_)^{\triangleright}$, except for the case of cuts, which now reads

$$\mathsf{Cut}(L_1, (x)\mathsf{Left}(x, L_2, (y)L_3))^{\blacktriangleright} = L_1^{\blacktriangleright}(L_2^{\blacktriangleright}, y.L_3^{\blacktriangleright}), \text{ if } x \notin L_2, L_3$$
$$\mathsf{Cut}(L_1, (x)L_2)^{\blacktriangleright} = \langle L_1^{\blacktriangleright}/x\rangle L_2^{\blacktriangleright}, \text{ if } \neg mll(x, L_2)$$

In particular, mappings $(\_)^{\triangleright}$ and $(\_)^{\blacktriangleright}$ coincide on cut-free terms.

Mapping $(\_)^{\blacktriangleright}$ has better properties than mapping $(\_)^{\triangleright}$ as to preservation of reduction, but it introduces a typical identification. Suppose $x \notin L_2, L_3$ and let $L_0 = \mathsf{Cut}(z, (x)\mathsf{Left}(x, L_2, (y)L_3))$ and $L_4 = \mathsf{Left}(z, L_2, (y)L_3)$. Then $L_0^{\blacktriangleright}$ and $L_1^{\blacktriangleright}$ are the same $\lambda g\mathtt{s}$-term of the form $z(N_2, y.N_3)$. By the way, $L_0 \to_{S20} L_4$.

Normalisation in $\lambda\mathbf{gs}$ versus cut-elimination in LJ: We now investigate how mappings $(\_)^{\diamond}$ and $(\_)^{\blacktriangleright}$ relate normalisation in $\lambda\mathbf{gs}$ and cut-elimination in LJ. To this end, we only need π_1 and π_3 among the π-rules of $\lambda\mathbf{gs}$. In addition, "eager" versions rules of π_1 and π_3 are required:

$$(\pi_1') \quad (\langle M/x\rangle N)(N', y.P') \to \langle M/x\rangle(N@(N', y, P'))$$
$$(\pi_3') \quad M(N, x.P)(N', y.P') \to M(N, x.P@(N', y, P')) \ ,$$

where $M@(N', y, P')$ is defined by recursion on M as follows: $x@(N', y, P') = x(N', y.P')$; $(\lambda x.M)@(N', y, P') = (\lambda x.M)(N', y.P')$; $(M(N, x.P))@(N', y, P') = M(N, x.P@(N', y, P'))$; and $(\langle M/x\rangle N)@(N', y, P') = \langle M/x\rangle(N@(N', y, P'))$.

Let $\pi' = \pi_1' \cup \pi_2'$. It is easy to see that one π_i'-step $(i = 1, 3)$ corresponds to one or more R-steps, where $R = \pi_1 \cup \pi_3$.

[2] It is well-known that cut-free LJ-terms are not in bijective correspondence with β-normal λ-terms; therefore, they are not in bijective correspondence with $\beta\pi\sigma$-normal $\lambda\mathtt{s}$-terms. In the particular case of $(\_)^{\blacktriangledown}$, terms of the form $\mathsf{Left}(y, L_1, x.L_2)$ are always mapped to a substitution (which is a σ-redex).

Proposition 11. *Let $R \in \{\beta, \pi', \sigma\}$. If $M \to_R N$ in $\lambda\mathbf{gs}$, then $M^\diamond \to^+_{S1,S2,Log} N^\diamond$ in LJ, except for some cases of $R = \sigma$, of the trivial form (6), for which one has $M^\diamond = N^\diamond$.*

Proof: By induction on $M \to_R N$, using the fact that, for all $M, N, P \in \lambda\mathbf{gs}$: (i) $S1(M^\diamond, x, N^\diamond) \to^*_{S20} [M/x]N)^\diamond$; (ii) if $x \notin N, P$ then $S2(M^\diamond, x, N^\diamond, y, P^\diamond) = (M@(N, y, P))^\diamond$. ■

An inspection of the proof shows that $(\_)^\diamond$ maps a reduction sequence ρ in $\lambda\mathbf{gs}$ from M_1 to M_2 to a reduction sequence $\rho^\diamond$ in LJ from $M_1^\diamond$ to $M_2^\diamond$ in a, so to say, structure-preserving way. Let $R \in \{\beta, \pi, \sigma\}$. To each R-step in ρ, there is a corresponding R'-step in $\rho^\diamond$, where R' is given by the left table below:

R	R'
σ	$S1 \cup S20$
π'_1	$S21 \cup S22$
π'_3	$S22$
β	Log

R	R'
$S1$	σ
$S21$	π'_3
$S22$	π'
Log	β

(7)

In addition, these R'-steps in $\rho^\diamond$ may be interleaved with $S20$-reduction steps.

Proposition 12. *Let $R \in \{S1, S21, S22, Log\}$. If $L_1 \to_R L_2$ (resp. $L_1 \to_{S20} L_2$) in LJ, then $L_1^\blacktriangleright \to^+_{\beta\pi\sigma} L_2^\blacktriangleright$ in $\lambda\mathbf{gs}$ (resp. $L_1^\blacktriangleright = L_2^\blacktriangleright$). Moreover, a reduction sequence ρ in LJ from L_1 to L_2 to a reduction sequence $\rho^\blacktriangleright$ in $\lambda\mathbf{gs}$ from $L_1^\blacktriangleright$ to $L_2^\blacktriangleright$ in a, so to say, structure-preserving way. To each R-step in ρ, there is a corresponding R'-step in $\rho^\blacktriangleright$, where R' is given according to the right table in (7). In addition, these R'-steps in $\rho^\blacktriangleright$ may be interleaved with trivial σ-steps of the form (6).*

Proof: By induction on $L_1 \to_R L_2$ or $L_1 \to_{S20} L_2$. The proof uses the fact that, for all $L_1, L_2, L_3 \in LJ$: (i) either $\langle L_1^\blacktriangleright/x\rangle L_2^\blacktriangleright \to_\sigma \mathsf{Cut}(L_1, (x)L_2)^\blacktriangleright$ or $\langle L_1^\blacktriangleright/x\rangle L_2^\blacktriangleright = \mathsf{Cut}(L_1, (x)L_2)^\blacktriangleright$; (ii) $S1(L_1, x, L_2)^\blacktriangleright = [L_1^\blacktriangleright/x]L_2^\blacktriangleright$; (iii) if $x \notin L_2, L_3$ then $S2(L_1, x, L_2, y, L_3)^\blacktriangleright = L_1^\blacktriangleright@(L_2^\blacktriangleright, y, L_3^\blacktriangleright)$. An inspection of this inductive proof shows the statement regarding reduction sequences. ■

***LJ* versus $\lambda\mathbf{gs}$:** Let us extract some lessons from the comparison between LJ and $\lambda\mathbf{gs}$ (where the latter is equipped with π' instead of π). The two systems are close. Cut-free terms and $\beta\pi\sigma$-nfs are in bijective correspondence. Up to trivial reduction steps of the form $S20$ or (6), reduction sequences are similar, and obey a correspondence between reduction steps of the forms $S1, S2, Log$ and σ, π', β, respectively. However, $\lambda\mathbf{gs}$ is a preferable syntax for three reason: (1) it avoids the "imperfect substitution" problem; (2) it has a lighter notation; (3) reduction rules do not have side conditions. Side conditions in the reduction rules of LJ guarantee that a cut undergoes a $S2$ reduction only when it is not a $S1$-redex. This sequencing is built in the syntax of $\lambda\mathbf{gs}$: a π-redex is never a σ-redex.

Mapping $\lambda\mathbf{gs}$ into $\lambda\mathbf{s}$: There is a bridge via $\lambda\mathbf{gs}$ between LJ and $\lambda\mathbf{s}$. We now close the bridge by studying mapping $(\_)^* : \lambda\mathbf{gs} \to \lambda\mathbf{s}$. This mapping

extends the one between $\lambda\mathbf{g}$ and $\lambda\mathtt{s}$, introduced in Section 3, with the clause $(\langle N/x\rangle M)^* = \langle N^*/x\rangle M^*$. As before with $\lambda\mathbf{g}$, mapping $(\_)^*$ produces a strict simulation of reduction in $\lambda\mathbf{gs}$ by reduction in $\lambda\mathtt{s}$ (for the moment, π is taken in its "lazy" form both in $\lambda\mathbf{gs}$ and $\lambda\mathtt{s}$). So, the following is immediate.

Theorem 5 (SN). *If $M \in \lambda g\mathtt{s}$ is typable, then M is $\beta\pi\sigma$-SN.*

Indeed, the simulation is perfect (one step in the source mapped to exactly one step in the target) for β and σ, and it is so for π if π is taken in the "eager" form. We introduce in $\lambda\mathbf{gs}$ an equivalent[3] definition of rules π'_1 and π'_3:

$$\begin{array}{ll}(\pi'_1) & (\langle M/x\rangle\mathcal{C}[V])(N, y.P) \to \langle M/x\rangle\mathcal{C}[V(N, y, P)] \\ (\pi'_3) & M(N, x.\mathcal{C}[V])(N, y.P) \to M(N, x.\mathcal{C}[V(N, y, P)]) \ ,\end{array}$$

where V is a value and $\mathcal{C}$ is a context belonging to the class

$$\mathcal{C} ::= [] \mid M(N, x.\mathcal{C}) \mid \langle N/x\rangle\mathcal{C}$$

Proposition 13. *Let $R \in \{\beta, \sigma, \pi'\}$. If $M \to_R N$ in $\lambda\mathbf{gs}$, $M^* \to_R N^*$ in $\lambda\mathtt{s}$.*

Proof: Only π' remains to be checked. It is useful to define, for $M \in \lambda\mathtt{s}$, $\mathcal{S} = \langle M/x\rangle\mathcal{C}^*$ by recursion on $\mathcal{C}$ as follows: $\langle M/x\rangle[]^* = \langle M/x\rangle[]$; $\langle M/x\rangle(P(N, y.\mathcal{C}))^* = \langle M/x\rangle(\langle P^*N^*/y\rangle\mathcal{C}^*)$; $\langle M/x\rangle(\langle N/y\rangle\mathcal{C})^* = \langle M/x\rangle(\langle N^*/y\rangle\mathcal{C}^*)$. By induction on $\mathcal{C}$ one proves that, for all $M, N, P \in \lambda\mathbf{gs}$, $(\langle M/x\rangle\mathcal{C}[P])^* = (\langle M^*/x\rangle\mathcal{C}^*)[P^*]$ and $M(N, x.\mathcal{C}[P])^* = (\langle M^*N^*/x\rangle\mathcal{C}^*)[P^*]$. Next we do an induction on $M \to_{\pi'} N$. ■

We can finally give a proof of Theorem 4. It follows from Propositions 12 and 13, and the fact that $(\_)^\blacktriangledown = (\_)^* \circ (\_)^\blacktriangleright$ and that $(\_)^*$ maps reduction steps of the form (6) to reduction steps of the form (5).

We finish by obtaining strong cut-elimination[4] for LJ as a corollary to strong normalisation for $\lambda\mathtt{s}$. Indeed, all we need is Theorem 4, together with the fact that $\to_{S20}$ is terminating and that $(\_)^\blacktriangledown$ preserves typability.

Theorem 6 (Strong cut-elimination). *Let $R = S1 \cup S2 \cup Log$. For all $L \in LJ$, if L is typable, then L is R-SN.*

Improving the computational interpretation: Theorem 4 is an improvement of Proposition 10 as to the interpretation of cut-elimination. Depending on the role we attribute to $\lambda\mathbf{gs}$, we can see Propositions 12 and 13 as improvements over Theorem 4 w.r.t. the same goal. On the one hand, if we regard $\lambda\mathbf{gs}$ as an adaptation of $\lambda\mathtt{s}$ particularly suited for the comparison with LJ, then Proposition 12 improves Theorem 4, because it includes a bijection between cut-free terms and $\beta\sigma\pi$-normal forms. On the other hand, if we regard $\lambda\mathbf{gs}$ as and

[3] The equivalence follows from two facts: (1) for all $V, N, P \in \lambda\mathbf{gs}$, $\mathcal{C}[V]@(N, y, P) = \mathcal{C}[V(N, y.P)]$; (2) every $M \in \lambda\mathbf{gs}$ can be written as $\mathcal{C}[V]$, with V value.

[4] A (weak) cut-elimination result is obtained as follows. Let $L \in LJ$. From $M = L^\triangleright$, one gets a $\beta\pi\sigma$-nf N. Proposition 11 guarantees that $L = L^{\triangleright\diamond} \to^* N^\diamond$ in LJ. Since N is a $\beta\pi\sigma$-nf, $N^\diamond$ is cut-free.

adaptation of LJ which avoids the problem of "imperfect substitution", then Proposition 13 improves Theorem 4, because it states a perfect correspondence that avoids the little perturbations of having S20-steps in the source reduction sequence that are not simulated, or extra σ-steps interleaved in the target reduction sequence.

However, in Proposition 13 the mismatch between $\beta\sigma\pi$-normal forms in $\lambda\mathtt{gs}$ and in $\lambda\mathtt{s}$ remains, and in Proposition 12 the little perturbations related to S20-steps and extra σ-steps survive (Theorem 4 shared both defects). For an *isomorphism* between sequent calculus and natural deduction, see [3].

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
2. V. Danos, J-B. Joinet, and H. Schellinx. A new deconstructive logic: linear logic. *The Journal of Symbolic Logic*, 62(2):755–807, 1997.
3. J. Espírito Santo. Completing Herbelin's programme. In *Proceedings of TLCA'07*, Lecture Notes in Computer Science. Springer-Verlag, 2007.
4. J. Gallier. Constructive logics. Part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:248–339, 1993.
5. H. Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.
6. F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42:59–87, 2003.
7. Felix Joachimski. On Zucker's isomorphism for LJ and its extension to pure type systems (submitted).
8. E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
9. Y. Ohta and M. Hasegawa. A terminating and confluent linear lambda calculus. In F. Pfenning, editor, *Proc. of 17th Int. Conference RTA 2006*, volume 4098 of *Lecture Notes in Computer Science*, pages 166–180. Springer-Verlag, 2006.
10. Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292.
11. G. Revesz. *Lambda-Calculus, Combinators, and Functional Programming*, volume 4 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1988.
12. K. Rose. Explicit substitutions: Tutorial & survey. Technical Report LS-96-3, BRICS, 1996.
13. F. van Raamsdonk, P. Severi, M. Sorensen, and H. Xi. Perpectual reductions in λ-calculus. *Information and Computation*, 149(2):173–225, 1999.
14. R. Vestergaard and J. Wells. Cut rules and explicit substitutions. In *Second International Workshop on Explicit Substitutions*, 1999.
15. J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.