Abstract

Herbelin presented (at CSL'94) a simple sequent calculus for minimal implicational logic, extensible to full first-order intuitionistic logic, with a complete system of cut-reduction rules which is both confluent and strongly normalising. Some of the cut rules may be regarded as rules to construct explicit substitutions. He observed that the addition of a cut permutation rule, for propagation of such substitutions, breaks the proof of strong normalisation; the implicit conjecture is that the rule may be added without breaking strong normalisation. We prove this conjecture, thus showing how to model beta-reduction in his calculus (extended with rules to allow cut permutations).

1 Introduction

Herbelin gave in [12] a calculus for minimal implicational logic, using a notation for proof terms that, in contrast to the usual lambda-calculus notation for natural deduction, brings head variables to the surface. It is thus a sequent calculus, with the nice feature that its cut-free terms are in a natural 1-1 correspondence with the normal terms of the simply typed λ-calculus. Other intuitionistic connectives can be added without difficulty.

The cut rules of the calculus are in part analogous to explicit substitution constructors [15, 17] and certain auxiliary operators (e.g. list concatenation); the only exception is a rule that in some circumstances constructs an auxiliary term and in others constructs a term analogous to a β-redex. Herbelin showed strong normalisation and confluence of a complete system of rules for eliminating cuts, observed that the addition of a further "cut permutation" rule, needed to allow explicit substitutions to propagate properly, as required for the simulation of β-reduction of the untyped λ-calculus, would break (in the typed case) the strong normalisation proof, thus raising the question of whether it also broke the strong normalisation result. In the present paper we answer this question; strong normalisation holds for the calculus with the addition of this rule (and of other cut permutation rules, to retain confluence). In fact, we identify a fragment of the calculus that corresponds to ordinary untyped λ-calculus, and show that the addition of the extra constructors and reduction rules preserves strong normalisation of the fragment.

Herbelin's calculus can thus be seen in two ways:

1. In the cut-free case it is a natural basis [6] for automated proof search in logic programming, since it is a sequent calculus but free from the permutation problems of Gentzen's calculus;

2. In the general case, when extended with appropriate cut permutation rules, it can simulate β-reduction without losing strong normalisation and thus, with its strong proof-theoretic foundations, may be a natural basis for implementation of functional languages.

Our proof uses standard techniques, e.g. from [2]. That paper uses recursive path ordering techniques to show the strong normalisation of a similar calculus. It has been argued that such techniques are inappropriate, since the systems dealt with there and here are not first-order. In response, we make two points: (1) that, to show strong

normalisation, the method of translation to a first-order system in [2] is adequate—in order not to obscure our argument, we omit the details of this translation, of which the conscientious reader is welcome to fill in the missing details; and (more seriously) (2) that our appeal to the critical pair lemma of first-order rewriting (to show that our purification system is locally confluent) is a shorthand for an argument, as implicitly made elsewhere (e.g. in [2]), that handles first-order terms modulo α-conversion; a formal treatment can be derived as a special case of the critical pair lemma in [16] for certain higher-order systems.

In other words, the details of the issues with variable names are omitted for simplicity; instead the usual, but informal, practice of the Barendregt Variable Convention is employed (i.e. we use a suitable chosen representative for an α-equivalence class and require a certain hygiene in our proofs). In [20] it has been shown in the context of the λ-calculus that this informal practice can be completely formalised, but at the expense of making explicit many details left implicit by the variable convention. Alternatively, one might use the technique presented in [11], where a general theory is provided which (in essence) allows one to strengthen induction hypotheses for reasoning with terms involving binding constructors. This strengthening ensures that the validity of predicates one deals with in proofs is invariant under interchanging, or swapping, of bound variable names.

Some other details can be found in the appendices of [8], an earlier version of the present paper.

2 Technical Background

We use a notation developed in [7], since it distinguishes the different kinds of cut term and in proofs emphasises the role of the *stoup* formula. For comparison, we offer also the syntax for terms used in Herbelin's paper [12]. The syntax of the calculus is as follows: *formulae* (also called *types*) A are as in implicational logic, there are *(term-)variables* $x, y, \ldots$, there are two kinds Ms, M of *proof-term* and two kinds of *sequent*, one with and one without a *stoup* formula, which is written in the first case below the sequent arrow. Ms is used in the stoup sequents; although the notation may suggest a list, not all terms Ms are lists. Notions of "free" and "bound" and variable conventions are as usual, with variable binding not just in λ-terms but also in cut_2 and cut_4 terms. *Contexts* Γ are finite functions from variables to formulae; $\Gamma, x : A$ indicates the extension, by the assignment of A to x, of a context Γ in which there is no assignment to x. $[M]$ abbreviates $(M :: [])$.

Syntax of cut-free terms

$$\begin{array}{rcl} Ms & ::= & [] \mid (M :: Ms) \\ M & ::= & (x; Ms) \mid \lambda x.M \end{array}$$

Syntax of cut-free terms [Herbelin's notation]

$$\begin{array}{rcl} l & ::= & [] \mid [t :: l] \\ t & ::= & (xl) \mid (\lambda x.t) \end{array}$$

Logical/Typing rules

$$\frac{\Gamma \Rightarrow M : A \quad \Gamma \underset{B}{\dashrightarrow} Ms : C}{\Gamma \underset{A \supset B}{\dashrightarrow} (M :: Ms) : C}\ S\supset \qquad \frac{}{\Gamma \underset{A}{\dashrightarrow} [] : A}\ Ax$$

$$\frac{\Gamma, x : A \underset{A}{\dashrightarrow} Ms : B}{\Gamma, x : A \Rightarrow (x; Ms) : B}\ Sel \qquad \frac{\Gamma, x : A \Rightarrow M : B}{\Gamma \Rightarrow \lambda x.M : A \supset B}\ R\supset$$

We explain the meaning of the two kinds of judgment as follows, restricting for simplicity to the cut-free case. $\Gamma \Rightarrow M : B$ can (of course) be interpreted as "M has an interpretation as a normal natural deduction proof of B from the assumptions declared in Γ"; similarly $\Gamma \underset{A}{\dashrightarrow} Ms : B$ should be interpreted as "Ms is interpretable as a list of normal natural deductions, all using assumptions declared in Γ, occurring as minor premisses on a chain of $\supset -elim$ steps starting from the assumption A and ending with the conclusion B".

Syntax of cut terms (type information is omitted in the type-free case)

$$\begin{array}{rcll} Ms & ::= & cut_1^A(Ms, Ms) & \mid\ cut_2^A(M, x.Ms) \\ M & ::= & cut_3^A(M, Ms) & \mid\ cut_4^A(M, x.M) \end{array}$$

Syntax of cut terms [Herbelin's notation]

$$\begin{array}{rcll} l & ::= & (l@l) & \mid\ l[x := t] \\ t & ::= & (tl) & \mid\ (t[x := t]) \end{array}$$

Logical/Typing rules for Cuts

$$\frac{\Gamma \underset{B}{\dashrightarrow} Ms : A \quad \Gamma \underset{A}{\dashrightarrow} Ms' : C}{\Gamma \underset{B}{\dashrightarrow} cut_1^A(Ms, Ms') : C}\ Cut_1 \qquad \frac{\Gamma \Rightarrow M : A \quad \Gamma, x : A \underset{B}{\dashrightarrow} Ms : C}{\Gamma \underset{B}{\dashrightarrow} cut_2^A(M, x.Ms) : C}\ Cut_2$$

$$\frac{\Gamma \Rightarrow M : A \quad \Gamma \underset{A}{\dashrightarrow} Ms : B}{\Gamma \Rightarrow cut_3^A(M, Ms) : B}\ Cut_3 \qquad \frac{\Gamma \Rightarrow M : A \quad \Gamma, x : A \Rightarrow M' : B}{\Gamma \Rightarrow cut_4^A(M, x.M') : B}\ Cut_4$$

Restricted to the special case where its arguments are just term lists, cut_1 is an explicit "append" operator; cut_4 is an explicit substitution operator which might be written as in $M'[x := M]$; cut_2 is similarly an explicit substitution operator which might be written as in $Ms[x := M]$; and cut_3 can be regarded as an explicit "generalised application" operator, an explicit version of what we will later write as in $\{M\}Ms$. They are, in other words, explicit versions of four functions we will use later when defining implicit substitution on the terms of the calculus: for details, see section 5.

Rules for cut-reduction

Let AO denote the system of "auxiliary operation" rules $1, 3, 5(a), 5(c)$ on terms, and ES the system of "explicit substitution" rules $2, 4, 5(b), 5(d)$ on terms:

1. (a) $cut_1^A([], Ms) \rightsquigarrow Ms$
   (b) $cut_1^A((M :: Ms), Ms') \rightsquigarrow (M :: cut_1^A(Ms, Ms'))$

2. (a) $cut_2^A(M, x.[]) \rightsquigarrow []$
   (b) $cut_2^A(M, x.(M' :: Ms)) \rightsquigarrow (cut_4^A(M, x.M') :: cut_2^A(M, x.Ms))$
3. (a) $cut_3^A((x; Ms), Ms') \rightsquigarrow (x; cut_1^A(Ms, Ms'))$
   (b) $cut_3^A(\lambda y.M, []) \rightsquigarrow \lambda y.M$
4. (a) $cut_4^A(M, x.(y; Ms)) \rightsquigarrow (y; cut_2^A(M, x.Ms)) \quad (y \neq x)$
   (b) $cut_4^A(M, x.(x; Ms)) \rightsquigarrow cut_3^A(M, cut_2^A(M, x.Ms))$
   (c) $cut_4^A(M, x.\lambda y.M') \rightsquigarrow \lambda y.cut_4^A(M, x.M')$
5. (a) $cut_1^A(cut_1^B(Ms, Ms'), Ms'') \rightsquigarrow cut_1^B(Ms, cut_1^A(Ms', Ms''))$
   (b) $cut_2^A(M, x.cut_1^B(Ms, Ms')) \rightsquigarrow cut_1^B(cut_2^A(M, x.Ms), cut_2^A(M, x.Ms'))$
   (c) $cut_3^A(cut_3^B(M, Ms), Ms') \rightsquigarrow cut_3^B(M, cut_1^A(Ms, Ms'))$
   (d) $cut_4^A(M, x.cut_3^B(M', Ms)) \rightsquigarrow cut_3^B(cut_4^A(M, x.M'), cut_2^A(M, x.Ms))$

The last of all these rules, 5(d), is the one mentioned in the introduction as the proper propagation of an explicit substitution, e.g. inside a "beta-redex". Rules 5(a)...(c) are added to ensure confluence once that rule is added. We allow rule applications inside terms, i.e. we in fact consider the contextual closures of all rules, as usual.

We need to consider one further rule, deliberately omitted from group 3:

B $\quad cut_3^{A \supset B}(\lambda x.M, (M' :: Ms)) \rightsquigarrow cut_3^B(cut_4^A(M', x.M), Ms)$

which generates a cut_4-instance; to simulate ordinary beta-reduction, this may need to propagate into its body M; the latter may be a cut_3-term, hence the utility of rule 5(d), and so on.

Herbelin [12] showed that a system of rules, essentially $(1 \ldots 4 + B)$ with different terminology, is complete, confluent and (for typed terms) strongly normalising. Note that the rules in 5 are NOT required for completeness: without them, a strategy that (more or less) ignores cuts whose arguments are cuts is imposed.

A simpler (but less direct) proof of his SN theorem using the multiset path ordering theorem is given in [7]; the termination comes from the ordering of all the *cut* operators as greater than the non-cut operators, with $cut^A > cut^B$ for $A > B$ and with $cut_4 = cut_2 > cut_3 = cut_1$ for *cut* operators with the same type annotation.

The addition of any of the rules in 5 breaks both of these proofs, because of the switching of the types.

Note that there are no rules allowing permutation of cut_2 or cut_4 operators with cut_2 or cut_4 operators.

Routinely, "Subject Reduction" holds for all these reduction rules; thus, all the results (e.g. confluence) that we state or prove for untyped terms hold also for the typed terms.

3 Pure Terms

There are two obvious (and in fact equivalent) candidates for the class of terms that we will use to interpret lambda terms: those free of $(AO + ES)$-redexes and those free of all instances (other than B-redexes) of *cut*.

Definition 1 ($(AO + ES)$-normality; purity)

1. A term is $(AO + ES)$-*normal* iff it is free of all $(AO + ES)$-redexes;
2. A term is *pure* iff it is free of all instances (other than B-redexes) of cut.

By induction, $(AO + ES)$-normal terms Ms are of the form [] or $M :: Ms'$. Pure terms are $(AO + ES)$-normal, since all $(AO + ES)$-redexes are instances of cut; but the converse also holds:

Proposition 2 $(AO + ES)$-normal terms Ms, M are pure.

Proof The Ms case depends just on the M case. We proceed by induction and case analysis; we show that if an $(AO + ES)$-normal term M is an instance of cut, then it is a redex; our inductive hypothesis is that all smaller $(AO + ES)$-normal terms are pure. Consider the cases for M an instance of cut:

1. $cut_3((x; Ms), Ms')$ is a 3(a)-redex;
2. $cut_3(\lambda x.M', [])$ is a 3(b)-redex;
3. $cut_3(\lambda x.M', (M'' :: Ms))$ is a B-redex;
4. $cut_3(cut_3(M', Ms), Ms')$ is a 5(c)-redex;
5. $cut_3(cut_4(M', x.M''), Ms)$ is not allowed, since (by inductive hypothesis) the term $cut_4(M', x.M'')$ is pure, contrary to its being a (non B-redex) cut-term;
6. $cut_4(M', x.(z; Ms))$ is a 4(a) or 4(b)-redex;
7. $cut_4(M', x.\lambda y.M'')$ is a 4(c)-redex;
8. $cut_4(M', x.cut_3(M'', Ms))$ is a 5(d)-redex;
9. $cut_4(M', x.cut_4(M'', y.M'''))$ is not allowed, for the same reason as in (5) above.

It follows that the $(AO + ES)$-normal (i.e. pure) terms M are of the form $(y; Ms)$, $\lambda x.M'$ or $cut_3(\lambda y.M', (M'' :: Ms))$.

Definition 3 (Implicit substitution; concatenation; generalised application) For pure terms Ms, Ms', M, M', and for the variable x, we define by simultaneous induction on the structure of Ms (resp. Ms, resp. M, resp. M')

1. The *concatenation*[1] $Ms@Ms'$ *of* Ms *with* Ms';
2. The *implicit substitution* $[M/x]Ms$ *of* M *for* x *in* Ms;
3. The *generalised application* $\{M\}Ms$ *of* M *to* Ms;
4. The *implicit substitution* $[M/x]M'$ *of* M *for* x *in* M'.

[1] This is just the usual concatenation of lists, included here for completeness.

as follows:

$$
\begin{array}{rcl}
[]@Ms' & \triangleq & Ms' \\
(M'' :: Ms'')@Ms' & \triangleq & (M'' :: (Ms''@Ms'))
\end{array}
$$

$$
\begin{array}{rcl}
[M/x][] & \triangleq & [] \\
[M/x](M'' :: Ms'') & \triangleq & ([M/x]M'' :: [M/x]Ms'')
\end{array}
$$

$$
\begin{array}{rcl}
\{(y; Ms'')\}Ms & \triangleq & (y; (Ms''@Ms)) \\
\{\lambda y.M''\}[] & \triangleq & \lambda y.M'' \\
\{\lambda y.M''\}(M''' :: Ms'') & \triangleq & cut_3(\lambda y.M'', (M''' :: Ms'')) \\
\{cut_3(\lambda y.M'', (M''' :: Ms''))\}Ms & \triangleq & cut_3(\lambda y.M'', (M''' :: (Ms''@Ms)))
\end{array}
$$

$$
\begin{array}{rcl}
[M/x](y; Ms'') & \triangleq & (y; [M/x]Ms'') \quad (y \neq x) \\
[M/x](x; Ms'') & \triangleq & \{M\}[M/x]Ms'' \\
[M/x](\lambda y.M'') & \triangleq & \lambda y.[M/x]M'' \\
[M/x]cut_3(\lambda y.M'', (M''' :: Ms'')) & \triangleq & cut_3(\lambda y.[M/x]M'', ([M/x]M''' :: [M/x]Ms'')).
\end{array}
$$

Note that the mutual induction is straightforward; first, concatenation is well-defined (as usual); second, generalised application has a definition depending only on concatenation; finally, the two forms of implicit substitution depend on simpler instances of themselves and of each other and on instances of generalised application.

4 Strong Normalisation and Confluence of $AO + ES$

Proposition 4 The system $AO + ES$ is strongly normalising (SN).

Proof A lexicographic path order suffices, completely ignoring all the type information, with the $cut_2 = cut_4$ operators equal, and greater than $cut_1 = cut_3$, and with all *cut* operators greater than the non-*cut* operators. We rely throughout on the exposition of the lexicographic path order given in [1], with its extra condition **LPO-1**, rather than that in [2]. For an alternative proof, avoiding the implicit need either to translate into a first-order system or to use the theory of termination in higher-order rewrite systems, we may just use a polynomial interpretation, see Appendix A of [8]. However, our 'proof' using the LPO method seems to be more enlightening.

Proposition 5 The system $AO + ES$ is confluent.

Proof By Proposition 4, it suffices to check the local confluence; for details see Appendix B of [8], bearing in mind the need to appeal to Newman's Lemma (which holds for arbitrary reduction relations) and (e.g.) to a Critical Pair Lemma (e.g. that in [16]).

Definition 6 For a term Ms or M, its *purification* is its $(AO + ES)$-normal form, written $\overline{Ms}$ (resp. $\overline{M}$).

Lemma 7 If $M \rightsquigarrow^*_{AO+ES} M'$, then $\overline{M} = \overline{M'}$. (Similarly for Ms.)

Proof Trivial.

Proposition 8 Let M, M', Ms, Ms' be pure terms. Then

1. $cut_1(Ms, Ms') \quad \rightsquigarrow^*_{AO+ES} \quad Ms@Ms'$;
2. $cut_2(M, x.Ms) \quad \rightsquigarrow^*_{AO+ES} \quad [M/x]Ms$;
3. $cut_3(M, Ms) \quad \rightsquigarrow^*_{AO+ES} \quad \{M\}Ms$;
4. $cut_4(M, x.M') \quad \rightsquigarrow^*_{AO+ES} \quad [M/x]M'$.

Proof The first part is trivial; the third part is proved by induction; the remaining two parts are proved by a simultaneous induction on the heights of the LHS terms. For details see Appendix C of [8].

The above may be regarded as a weak normalisation result (for $AO + ES$), since we may use it to purify any innermost (non-B)-redex, repeating this operation until all such redexes are eliminated.

5 Lambda Calculus

We give two equivalent versions of the lambda calculus; first, the standard one and, second, one very similar to the Herbelin notation but found also in [14].

First, lambda terms N are defined by the grammar

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

Second, lambda terms J and lists Js of lambda terms are defined by the grammar

$$\begin{array}{rcl} J & ::= & (x\ Js) \mid (\lambda x.J) \mid ((\lambda x.J)J\,Js) \\ Js & ::= & [] \mid (J :: Js) \end{array}$$

in which, if we omit the last production for J, we get just the normal terms. Note that, for example, a term of the form $((\lambda x.J)J'Js)$ is NOT the term list $((\lambda x.J) :: (J' :: Js))$; it is a term. The structure of this inductive definition is intended to make certain parts of the normalisation proof in [14] easy.

The equivalence between the two notations is routinely shown using the translation $(.)^\diamond : N \to J$ and the auxiliary function $(.)^{\diamond\diamond} : N \times Js \to J$ defined as follows:

$$N^\diamond \quad \triangleq \quad (N, [])^{\diamond\diamond}$$

$$\begin{array}{rcl} (x, Js)^{\diamond\diamond} & \triangleq & (x\ Js) \\ ((\lambda x.N), [])^{\diamond\diamond} & \triangleq & (\lambda x.N^\diamond) \\ ((\lambda x.N), J :: Js)^{\diamond\diamond} & \triangleq & (\lambda x.N^\diamond)J\,Js \\ ((N\,N'), Js)^{\diamond\diamond} & \triangleq & (N, N'^\diamond :: Js)^{\diamond\diamond} \end{array}$$

No definition of substitution is given in [14]; essentially, the translation to standard notation is used, then the standard definition is used, then one translates back. We remedy this minor oversight as follows:

We define implicit substitution $[J/x]Js$ of J for x in Js (and similarly for substitution in J') as follows, by induction on the structure of Js (resp. J'); we need an auxiliary definition of a term $\{J\}Js$, by induction on the structure of J (and a subsidiary induction in one case on Js), to apply the term J to the list Js of arguments in the usual way (this is not the construction of a list, but of a term):

$$\begin{array}{rcl}
[J/x][] & \triangleq & [] \\
[J/x](J' :: Js) & \triangleq & ([J/x]J' :: [J/x]Js) \\
 & & \\
\{(y\ Js')\}Js & \triangleq & (y\ (Js'@Js)) \\
\{(\lambda y.J)\}[] & \triangleq & (\lambda y.J) \\
\{(\lambda y.J)\}(J' :: Js) & \triangleq & ((\lambda y.J)J'Js) \\
\{((\lambda y.J)J'Js)\}Js' & \triangleq & ((\lambda y.J)J'(Js@Js')) \\
 & & \\
[J/x](y\ Js) & \triangleq & (y\ [J/x]Js) \quad (y \neq x) \\
[J/x](x\ Js) & \triangleq & \{J\}[J/x]Js \\
[J/x](\lambda y.J') & \triangleq & (\lambda y.[J/x]J') \\
[J/x]((\lambda y.J')J''Js) & \triangleq & ((\lambda y.[J/x]J')([J/x]J''[J/x]Js))
\end{array}$$

where @ is for the standard function that concatenates two lists.

Termination of this definition is as in the previous section. The equivalence between this definition and the usual definition of substitution (with the usual notation) is a tedious exercise.

β-reduction is thus the reduction of a term by careful replacement of a subterm (the *β-redex*) of the form $((\lambda x.J)J'Js)$ by the *reduct* $\{[J'/x]J\}Js$ in a single step.

Proposition 9 For λ-terms J and term lists Js', Js'', the following holds:

$$\{\{J\}Js'\}Js'' = \{J\}(Js'@Js'')$$

Proof Routine induction on the size of J and case analysis, using associativity of @.

6 Interpretation of Lambda Calculus

We give two equivalent interpretations of the lambda calculus, in each case using only the pure terms. The first $(.)^\star$ uses the traditional presentation of the lambda calculus; the second $(.)^\bullet$ uses that from [14] outlined above.

Definition 10 [Interpretation 1] We define the interpretation $(.)^\star$ of λ-terms N as the pure terms of the form M, as follows:

$$\begin{array}{rcl} (x)^\star & \triangleq & (x;[]) \\ (\lambda x.N)^\star & \triangleq & (\lambda x.N^\star) \\ (NN')^\star & \triangleq & \overline{cut_3(N^\star,[N'^\star])} \end{array}$$

By Proposition 8, we may rephrase the last clause as

$$(NN')^\star \quad \triangleq \quad \{N^\star\}[N'^\star].$$

Definition 11 [Interpretation 2] We define the interpretations $(.)^\bullet$ of λ-terms J as the pure terms of the form M and $(.)^{\bullet\bullet}$ of term-lists Js as the pure terms of the form Ms, as follows:

$$\begin{array}{rcl} (x\ Js)^\bullet & \triangleq & (x;Js^{\bullet\bullet}) \\ (\lambda x.J)^\bullet & \triangleq & (\lambda x.J^\bullet) \\ ((\lambda x.J)J'Js)^\bullet & \triangleq & cut_3(\lambda x.J^\bullet,(J'^\bullet :: Js^{\bullet\bullet})) \end{array}$$

$$\begin{array}{rcl} []^{\bullet\bullet} & \triangleq & [] \\ (J :: Js)^{\bullet\bullet} & \triangleq & (J^\bullet :: Js^{\bullet\bullet}) \end{array}$$

Proposition 12 For λ-terms J, J' and term lists Js, Js', the following hold:

1. $(Js@Js')^{\bullet\bullet} = Js^{\bullet\bullet}@Js'^{\bullet\bullet}$;
2. $([J/x]Js)^{\bullet\bullet} = [J^\bullet/x]Js^{\bullet\bullet}$;
3. $(\{J\}Js)^\bullet = \{J^\bullet\}Js^{\bullet\bullet}$;
4. $([J/x]J')^\bullet = [J^\bullet/x]J'^\bullet$.

Proof Routine.

Lemma 13

$$(N,Js)^{\diamond\diamond} = \{N^\diamond\}Js$$

Proof By induction on the size of N and cases; application terms are handled by Proposition 9.

Lemma 14

$$((N,Js)^{\diamond\diamond})^\bullet = \{N^{\diamond\bullet}\}Js^{\bullet\bullet}$$

Proof By Proposition 8 and Lemma 13.

Lemma 15

$$((N,N'^\diamond :: Js)^{\diamond\diamond})^\bullet = \{N^{\diamond\bullet}\}(N'^{\diamond\bullet} :: Js^{\bullet\bullet})$$

Proof By instantiating Lemma 14 and using the last clause of the definition of $(.)^{\bullet\bullet}$.

Corollary 16

$$((N,[N'^{\diamond}])^{\diamond\diamond})^{\bullet} = \{N^{\diamond\bullet}\}[N'^{\diamond\bullet}]$$

Proposition 17 The two interpretations of the lambda calculus are the same, i.e. for every term N, $N^{\diamond\bullet} = N^{\star}$.

Proof By induction on the size of N and case analysis:

1. $x^{\diamond\bullet} = ((x,[])^{\diamond\diamond})^{\bullet} = (x\ [])^{\bullet} = (x;[]^{\bullet\bullet}) = (x,[]) = x^{\star}$
2. $(\lambda x.N')^{\diamond\bullet} = (((\lambda x.N'),[])^{\diamond\diamond})^{\bullet} = (\lambda x.N'^{\diamond})^{\bullet} = (\lambda x.N'^{\diamond\bullet}) = (\lambda x.N'^{\star}) = (\lambda x.N')^{\star}$
3. $(N'N'')^{\diamond\bullet} = (((N'N''),[])^{\diamond\diamond})^{\bullet} = ((N',[N''^{\diamond}])^{\diamond\diamond})^{\bullet} = \{N'^{\diamond\bullet}\}[N''^{\diamond\bullet}]$
   $= \{N'^{\star}\}[N''^{\star}] = (N'N'')^{\star}$

in which all equations are definitional or inductive, except for one appeal to Corollary 16.

7 Simulation of β-reduction

Proposition 18 For pure terms M, M' and Ms,

$$cut_3(cut_4(M', x.M), Ms) \leadsto^*_{AO+ES} \{[M'/x]M\}Ms.$$

Proof By Proposition 8.

Theorem 19 If $J_1 \leadsto_\beta J_2$ in the λ-calculus, then $J_1^{\bullet}$ reduces to $J_2^{\bullet}$ by a B-reduction followed by 0 or more $(AO+ES)$ reductions.

Proof Consider first the case where J_1 is the β-redex, and so of the form $((\lambda x.J)J'Js)$, for terms J, J' and term list Js. Thus, $J_2 = \{[J'/x]J\}Js$. By Proposition 18, the reduct

$$cut_3(cut_4(J'^{\bullet}, x.J^{\bullet}), Js^{\bullet\bullet})$$

of the B-redex $J_1^{\bullet} = cut_3(\lambda x.J^{\bullet}, (J'^{\bullet} :: Js^{\bullet\bullet}))$ is $(AO+ES)$-reducible to $\{[J'^{\bullet}/x]J^{\bullet}\}Js^{\bullet\bullet}$. By Proposition 12, this is just $J_2^{\bullet}$. The general case, where the reduction is not at the root position of J_1, follows by induction on the structure of J_1.

In other words, the system $(AO+ES+B)$ of rules acting on the untyped terms *simulates* β-reduction of the untyped λ-calculus; similarly for typed terms and the typed λ-calculus.

8 β-reduction

We may now define a rule β on pure terms, typed or untyped, omitting the types in the latter case, as the contextual closure of the following rule:

β $\qquad cut_3^{A\supset B}(\lambda x.M, M' :: Ms) \leadsto_\beta \overline{cut_3^B(cut_4^A(M', x.M), Ms)}$

Note that, in the untyped case, the RHS of this is, by Proposition 18, just $\{[M'/x]M\}Ms$. The correspondence between pure untyped terms and the terms of the untyped λ-calculus routinely extends to β-reduction.

Thus, a β-reduction is a single B-reduction followed by purification, i.e. zero or more $(AO + ES)$-reductions to normal form.

Proposition 20 The system, on pure typed terms, consisting just of the rule β is SN.

Proof Use, for example, the proof, in different notation, in [14].

If we had a direct proof of Proposition 20, then we would have shown the strong normalisation of the simply-typed λ-calculus.

Definition 21 For any term M, we define $\| M \|$ to be the maximal length of all β-reduction sequences from $\overline{M}$ if the latter is β-SN; otherwise we define $\| M \| = \infty$. (Similarly for Ms.)

Corollary 22 For every typed term M, we have $\| M \| < \infty$. (Similarly for Ms.)

Proof By Proposition 20 and König's Lemma.

Lemma 23 For pure terms M, Ms, M', M'', Ms', with $M \leadsto_\beta^* M'$ and $Ms \leadsto_\beta^* Ms'$, we have

1. $\{M\}Ms \leadsto_\beta^* \{M'\}Ms$;
2. $\{M\}Ms \leadsto_\beta^* \{M\}Ms'$;
3. $[M/x]Ms \leadsto_\beta^* [M'/x]Ms$;
4. $[M/x]Ms \leadsto_\beta^* [M/x]Ms'$;
5. $[M/x]M'' \leadsto_\beta^* [M'/x]M''$;
6. $[M''/x]M \leadsto_\beta^* [M''/x]M'$.

Proof These follow from consideration of the untyped λ-calculus: for example, substitution into a β-redex leaves it as a β-redex.

Lemma 24 For pure terms M, M', Ms with $M \leadsto_\beta M'$ we have $\{M\}Ms \leadsto_\beta \{M'\}Ms$.

Proof By induction on the definition of $\{.\}$. (generalised application). The essential idea is that any β-redex in M is still a β-redex in $\{M\}Ms$, even though M may well not be a subterm of $\{M\}Ms$.

9 Adding B-reduction

We will show that the addition of the B-rule upsets neither confluence nor, provided we stick at least to (e.g.) typed terms, termination. A key ingredient in both of these proofs is the Projection Lemma, i.e. that a root B-reduction translates (under purification) to exactly one β-reduction.

Lemma 25 $\overline{cut_3(\lambda x.M, (M' :: Ms))} \leadsto_\beta \overline{cut_3(cut_4(M', x.M), Ms)}$.

Proof The $LHS = cut_3(\lambda x.\overline{M}, (\overline{M'} :: \overline{Ms}))$, the latter (as a B-redex) being pure; this β-reduces to the term $\overline{cut_3(cut_4(\overline{M'}, x.\overline{M}), \overline{Ms})}$. By Lemma 7, this equals the RHS.

Lemma 26 (Projection Lemma) If $M \rightsquigarrow_B M'$ at the root position, then $\overline{M} \rightsquigarrow_\beta \overline{M'}$.

Proof Apart from the names of variables, this is just a restatement of Lemma 25.

Corollary 27 For terms M, M', Ms, we have (if the $RHS < \infty$)

$$\| cut_3(\lambda x.M, (M' :: Ms)) \| > \| cut_3(cut_4(M', x.M), Ms) \|.$$

We also need to know that an arbitrary B-reduction translates, after purification, to zero or more β-reductions.

Proposition 28 The following hold:

1. If $Ms \rightsquigarrow_B Ms'$, then $\overline{Ms} \rightsquigarrow_\beta^* \overline{Ms'}$;
2. If $M \rightsquigarrow_B M'$, then $\overline{M} \rightsquigarrow_\beta^* \overline{M'}$.

Proof By simultaneous inductions on the size of Ms or M, and case analysis:

1. (a) $Ms = []$: trivial;
   (b) $Ms = (M :: Ms'')$: routine use of inductive hypothesis;
   (c) $Ms = cut_1(Ms'', Ms''')$: similar to the first two parts of case 2(c) below.
   (d) $Ms = cut_2(M'', x.Ms''')$: similar to the case 2(d) below.
2. (a) $M = \lambda x.M''$: routine;
   (b) $M = (x; Ms'')$: routine;
   (c) $M = cut_3(M'', Ms'')$: there are three cases:
      i. the B-reduction is of M'' to M''': by inductive hypothesis, $\overline{M''} \rightsquigarrow_\beta^* \overline{M'''}$, whence, by Lemma 23 (1), $\{\overline{M''}\}\overline{Ms''} \rightsquigarrow_\beta^* \{\overline{M'''}\}\overline{Ms''}$. Now,

$$\overline{cut_3(M'', Ms'')} = \overline{cut_3(\overline{M''}, \overline{Ms''})} = \{\overline{M''}\}\overline{Ms''}$$

      by Lemma 7 and Proposition 8 (3) respectively; and similarly

$$\overline{cut_3(M''', Ms'')} = \overline{cut_3(\overline{M'''}, \overline{Ms''})} = \{\overline{M'''}\}\overline{Ms''}$$

      whence $\overline{M} \rightsquigarrow_\beta^* \overline{M'}$.
      ii. the B-reduction is of Ms'' to Ms''': similar, using Lemma 23 (2).
      iii. the B-reduction is at the root of M: we use the Projection Lemma.
   (d) $M = cut_4(M'', x.M''')$: there are two cases, dealt with as in (c), using Lemma 23 (5) and (6).

It is now easy to show that the system $(AO + ES + B)$ is confluent, using the confluence of β-reduction in the λ-calculus.

Theorem 29 The system $(AO + ES + B)$ is confluent.

Proof Suppose that $M \leadsto^*_{AO+ES+B} M_1$ and $M \leadsto^*_{AO+ES+B} M_2$. Then, by Lemma 7 and Proposition 28, both $\overline{M} \leadsto^*_\beta \overline{M_1}$ and $\overline{M} \leadsto^*_\beta \overline{M_2}$, whence, by confluence of β-reduction in the λ-calculus, for some (pure) term M° we have $\overline{M_1} \leadsto^*_\beta M^\circ$ and $\overline{M_2} \leadsto^*_\beta M^\circ$. But then, both $M_1 \leadsto^*_{AO+ES+B} M^\circ$ and also $M_2 \leadsto^*_{AO+ES+B} M^\circ$. (Similarly for Ms.)

10 Strong Normalisation

Definition 30 (Bounded terms) A term M (or Ms) is *bounded* iff for every subterm M' or Ms' thereof, $\|M'\| < \infty$ (resp. $\|Ms'\| < \infty$).

Proposition 31 (Boundedness)

1. Every typed term is bounded;
2. Every $(AO + ES + B)$-SN term is bounded;
3. For pure terms, β-SN is equivalent to "bounded".

Proof We consider the three parts in order:

1. Trivial, since every subterm of a typed term is typed and the purification of a typed term is typed, and (by Proposition 20) every pure typed term is β-SN.
2. Consider a subterm M of an $(AO+ES+B)$-SN term; it also is $(AO+ES+B)$-SN and so is its purification $\overline{M}$. But then any infinite sequence of β-reductions from $\overline{M}$ can be seen, by Corollary 18, as a sequence of $(AO+ES+B)$-reductions, including infinitely many B-reductions. (Similarly for subterms Ms.)
3. Pure β-SN terms are bounded, since, for every subterm M of such a term, M is pure and β-SN; so $\overline{M}$ $(= M)$ is β-SN (and similarly for subterms Ms). The converse is even more trivial.

Our aim now is to prove Theorem 39, i.e. the converse of part (2) of Proposition 31.

Lemma 32 For all terms M, M', Ms, Ms':

1. If $M \leadsto_{AO+ES} M'$ then $\|M\| = \|M'\|$;
2. If $Ms \leadsto_{AO+ES} Ms'$ then $\|Ms\| = \|Ms'\|$;
3. If $M \leadsto_B M'$ then $\|M\| \geq \|M'\|$;
4. If $Ms \leadsto_B Ms'$ then $\|Ms\| \geq \|Ms'\|$.

Proof The first part is trivial, since $\overline{M} = \overline{M'}$; the second part is similar. The other two parts use Proposition 28.

Definition 33 (Cosy occurrence; cosy embedding) An occurrence of a proper subterm in a term is *cosy* iff no cut_2 or cut_4 constructors intervene on the path to the root, apart from a possible occurrence at the subterm itself. A term is *cosily embedded in* a term iff it has a cosy occurrence in the latter term.

For example, in a term of the form $cut_1(cut_2(M, x.Ms), cut_2(M, x.Ms'))$, the two cut_2 subterms are the only cosily embedded proper subterms; and in a term of the form $cut_2(M, x.Ms)$, no proper subterms are cosily embedded.

Lemma 34 For all terms M, Ms, Ms':

1. (a) $\|(M :: Ms)\| \geq \|M\|$;
   (b) $\|(M :: Ms)\| \geq \|Ms\|$;
2. $\|(x; Ms)\| = \|Ms\|$;
3. $\|\lambda x.M\| = \|M\|$;
4. (a) $\|cut_1(Ms, Ms')\| \geq \|Ms\|$;
   (b) $\|cut_1(Ms, Ms')\| \geq \|Ms'\|$;
5. $\|cut_3(M, Ms)\| \geq \|M\|$;
6. $\|cut_3(M, Ms)\| \geq \|Ms\|$.

Proof 1. Because $\overline{(M :: Ms)} = (\overline{M} :: \overline{Ms})$;

2. Because $\overline{(x; Ms)} = (x; \overline{Ms})$;

3. Because $\overline{\lambda x.M} = \lambda x.\overline{M}$;

4. Because, by Proposition 8 (1), $\overline{cut_1(Ms, Ms')} = \overline{Ms}@\overline{Ms'}$;

5. Since $\overline{cut_3(M, Ms)} = \overline{cut_3(\overline{M}, \overline{Ms})}$ and $\|M\| = \|\overline{M}\|$, we may, without loss of generality, assume that the terms M and Ms are pure; we now just appeal to Lemma 24.

6. Without loss of generality, for the same reason as in (5), M and Ms may be assumed to be pure. If $Ms = []$, then the result is trivial. Otherwise, we argue by induction on the size of M and case analysis. Consider the possible forms of M:

   (a) $M = (x; Ms'')$, whence by rule 3(a) it suffices to observe that $\|(x; Ms''@Ms)\| = \|Ms''\| + \|Ms\|$;

   (b) $M = \lambda x.M'$; so $cut_3(M, Ms)$ is pure and has Ms as a sub-term, whence $\|cut_3(M, Ms)\| \geq \|Ms\|$;

   (c) $M = cut_3(M', Ms'')$; so, by rule 5(c), $\overline{cut_3(M, Ms)} = \overline{cut_3(M', Ms''@Ms)}$ and, by inductive hypothesis, $\|cut_3(M', Ms''@Ms)\| \geq \|Ms''@Ms\| \geq \|Ms\|$.

Corollary 35 If a term M is cosily embedded in M', then $\|M\| \leq \|M'\|$, and similarly for other combinations such as M cosily embedded in Ms, etc.

Proof By the lemma, using induction on the number of constructors between the subterm and the term.

On the other hand, whenever $\|M\| > 0$, we have

$$\|M\| \not\leq \|cut_2(M, x.[])\| = 0$$

and

$$\|M\| \not\leq \|cut_4(M, x.\lambda y.(y; []))\| = 0.$$

Corollary 36 We have the following:

1. For each of the $(AO + ES)$-reduction rules $L \rightsquigarrow R$, and for each non-variable subterm S of R, and instantiation θ of the variables in the rule, we have $\|L^\theta\| \geq \|S^\theta\|$;
2. For the B-reduction rule $L \rightsquigarrow R$, and for each non-variable subterm S of R, and instantiation θ of the variables in the rule, we have $\|L^\theta\| \geq \|S^\theta\|$, the inequality being strict if either side is finite.

Proof 1. By Lemma 32 (1 or 2), we have $\|L^\theta\| = \|R^\theta\|$. It now suffices to note that, for each rule $L \rightsquigarrow R$, each non-variable proper subterm of R is cosily embedded.

2. By Lemma 26, assuming $\|L^\theta\| < \infty$, we have $\|L^\theta\| > \|R^\theta\|$; as before, the only non-variable proper sub-term S of R is cosily embedded.

This corollary is the crux of the present paper: it gives us information about the $(AO+ES+B)$ rules that will be exploited when we show that the rules are decreasing w.r.t. a suitably chosen ordering. Note that the above corollary tells us just about reductions at the root position; for reductions at non-root positions, it says nothing.

Corollary 37 Bounded terms are closed under $(AO + ES + B)$-reduction.

Proof Let M be bounded and let R be a rule in (AO+ES+B) such that $M \rightsquigarrow_R M'$. Consider a subterm M'' of M'; we must show that $\|M''\| < \infty$. Comparing the position of M'' in M' to that of the reduct, we find three cases:

1. The reduct occurs as a subterm of M''; so we can pull M'' back to a subterm M^* of M, with $M^* \rightsquigarrow M''$. Since M is bounded, $\|M^*\| < \infty$. By Lemma 32, $\|M''\| < \infty$.
2. The reduct has M'' as a proper subterm, and M'' is obtained by instantiation of a variable in the rule R; thus already M'' is a subterm of M, which is bounded, so $\|M''\| < \infty$.
3. The reduct has M'' as a proper subterm, and M'' is obtained by instantiation of a non-variable subterm of the RHS of the rule R; by Corollary 36, $\|M''\| \leq \|M^*\|$ for some term M^* which is in fact a subterm of M, so $\|M''\| < \infty$.

We now consider the bounded cut terms superfixed not, as before, with types but with, for each such term M, the natural number $\| M \|$ (and similarly for terms Ms). We again order the cut operators by $cut^n > cut^m$ for $n > m$, use the suffices $(4 = 2 > 3 = 1)$ as before for ordering cut operators with the same superfix, and order all cut operators as greater than all non-cut operators. There are now infinitely many operators; but for a given bounded term, with only finitely many cut sub-terms, we can compute an upper bound for all their superfixes; by Corollary 36, this bound suffices for all terms reachable from the term by any of the reduction rules, so we can w.l.o.g. assume that our signature is finite, as required for use of the fact that the lexicographic path ordering generated below is a simplification ordering and thus is well-founded.

From this ordering on this (finite) signature we generate the lexicographic path ordering "$>_{LPO}$" on all terms. As in [2], we can avoid the problem of the LPO techniques not being applicable to higher-order systems by translation into a terminating (but non-confluent) intermediate system where the bound variables are omitted. However in order not to obscure our argument, we will not give this translation, but rather apply the LPO techniques directly to our higher-order system.

Proposition 38 If M' is a bounded term and $M' \leadsto_{AO+ES+B} M''$, then $M' >_{LPO} M''$. (Similarly for Ms.)

Proof It suffices to consider only reductions at root position, since $>_{LPO}$ is closed under contextual closure. In fact, the previous proof (of Proposition 4) for $(AO+ES)$ now works almost unchanged. We consider the rule B in order to illustrate the method: let

$$M' = cut_3^m(\lambda x.M, (M''' :: Ms)) \leadsto_B cut_3^r(cut_4^s(M''', x.M), Ms) = M''$$

be an instance of rule B. By Corollary 27, we have $m > r$; similarly $m > s$ by this and by Corollary 36 (2). Then, since $m > r$, we just need to compare M' with the two main subterms of M''. That $M' >_{LPO} cut_4^s(M''', x.M)$ follows because $m > s$ and M''', M are variables properly occurring in M'; that $M' >_{LPO} Ms$ is trivial, the latter being a variable properly occurring in M'. It follows that $M' >_{LPO} M''$.

Theorem 39 Every bounded term is $(AO + ES + B)$-SN.

Proof By the well-foundedness of $>_{LPO}$ and Proposition 38.

Corollary 40 Every typed term is $(AO + ES + B)$-SN.

Proof By Proposition 31 (1) and the above theorem.

(This answers Herbelin's question.)

Corollary 41 The calculus of terms Ms, M with the $(AO+ES+B)$-reduction rules preserves strong normalisation w.r.t. the calculus of pure terms under β-reduction.

11 Comments

Our cut-reductions 5(c), 5(d) for Herbelin's explicit substitution calculus already appeared, in different notation, as the rules $\overline{\lambda}22$ and $\overline{\lambda}44$ in Espírito Santo's [9]; this paper *inter alia*

1. establishes a 1-1 correspondence, concerning both terms and reductions, between the lambda calculus and his calculus λ_H of terms (similar to our calculus of pure terms);

2. shows that λ_H can be extended to a calculus λ_H^+ of terms which, in our notation, have no instances of cut_1 and cut_2. These two operators are treated as defined functions; our category of terms Ms can thus be replaced by that of term lists. This corresponds to a certain strategy of reducing cut_1 and cut_2 terms to terms normal w.r.t. our rules 1 and 2; our rules 5(a) and 5(b) are then superfluous. Moreover, cut_3 and cut_4 terms are, following a reduction step, dealt with immediately by auxiliary functions, defined by equations, rather than by use of explicit operations.

However, the issue of whether B-reductions can be combined in an *arbitrary* fashion with $(AO + ES)$-reductions is not addressed; this appears to us to be the main issue raised by Herbelin's paper, with its emphasis on explicit concatenations and explicit substitutions. We gratefully acknowledge José Espírito Santo's helpful comments illuminating the content of his paper. Further details of his work are in his thesis [10].

Vestergaard and Wells [21] have considered explicit substitution calculi based on Gentzen's L-systems, with de Bruijn indices rather than variable names and with "weak correspondences" with some known explicit substitution calculi. Their calculi are not powerful enough to simulate β-reduction; in fact, their results are that some existing explicit substitution calculi (modified slightly) can simulate their new calculi rather than *vice-versa*.

It would be interesting to have a direct strong normalisation proof for the above calculus (with rules $(AO + ES + B)$) and to compare it with those for the simply-typed λ-calculus of [14, 18]. Such a proof might, for example, use the reducibility method of Tait and Girard; indeed, such a proof is available for a more complex calculus, the $\overline{\lambda}\mu\overline{\mu}$-calculus of Herbelin [13]. Herbelin (private commmunication, March 2001) conjectured that our SN result for $(AO + ES + B)$ also follows from his SN result in the paper just cited, mentioning however that a similar conjecture for the strong normalisability of Parigot's $\lambda\mu$-calculus turned out to be mistaken. We have no opinion on this conjecture; in any case, it is good to have a more elementary proof.

Herbelin's calculus also appears in the work of Cervesato and Pfenning [4], in the guise of a "spine calculus"; no theory of explicit substitutions therein appears to have been worked out, although some implementation of such substitutions is apparently in the Twelf implementation. We thank Iliano Cervesato for bringing this report to our attention.

There are of course issues in the explicit substitution world that are not addressed by the above, such as the questions of optimality, of sharing and of confluence on open terms. We make no claims about superiority of the system $(AO + ES + B)$ over other explicit substitution calculi; we remark merely that it has a simple proof-theoretic pedigree.

Appendix D of [8] shows that our calculus simulates the $\lambda\mathbf{x}$-calculus [3], in the weak sense that every reduction of the latter translates to one or more reductions in our system, thus (with some minor extra effort) proving SN for the typed fragment of $\lambda\mathbf{x}$. As seen above, in the case of the ordinary λ-calculus, there is a stronger sense in which one might simulate such a system, in order to prove preservation of strong normalisation for the simulated system. We have however not been able to identify a class of "weakly pure" terms of our calculus (and appropriate reductions thereon) to match the $\lambda\mathbf{x}$ terms and reductions in the same way that the "pure" terms (and β-reduction) match the ordinary terms (and β-reduction) of the λ-calculus. The stumbling block is (again) the need to push an explicit substitution into an application term; consider (e.g.) the translated term $cut_4(N^*, x.\overline{cut_3(N'^*, [N''^*])})$ where the * superscript indicates the translation and the overbar denotes a notion of "weak purification" extending the notion of purification as applied to the translates of substitution-free λ-terms. The obvious candidate reduction is by $5(d)$ to $cut_3(cut_4(N^*, x.N'^*), cut_4(N^*, x.N''^*))$, but the weak purification in the LHS prevents this, typically merging the term N'^* with the list $[N''^*]$ (whereas in the reduct they are separate).

On the other hand, [13] reports the simulation of $\lambda\mathbf{x}$ in the $\overline{\lambda}\mu\overline{\mu}$-calculus; further work is therefore required to clarify this issue.

Finally, not so much as an afterthought but as a reference back to earlier work [6] relating Herbelin's cut-free calculus to proof search, we mention again our interest in the discrepancy between calculi organised for logic programming, i.e. for cut-free proof search, and those organised for functional programming, i.e. for proof normalisation. The work in the present paper is intended to narrow the gap between the two kinds of calculi; but issues like confluence on open terms have yet to be addressed before the gap may be regarded as satisfactorily closed.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] R. Bloo and H. Geuvers. Explicit Substitution: On the Edge of Strong Normalisation. *Theoretical Computer Science*, 211(1–2):375–395, 1999.

[3] R. Bloo and K. H. Rose. Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection. In *Proceedings of Computing Science in the Netherlands'95*, pages 62–72, 1995.

[4] I. Cervesato and F. Pfenning. A Linear Spine Calculus. Technical report, CMU-CS-97-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1997. Preliminary version of [5].

[5] I. Cervesato and F. Pfenning. A Linear Spine Calculus. *Journal of Logic and Computation (this volume)*, To appear.

[6] R. Dyckhoff and L. Pinto. Proof Search in Constructive Logic. In S. B. Cooper and J. K. Truss, editors, *Proceedings of the Logic Colloquium 1997*, volume 258 of *London Mathematical Society Lecture Note Series*, pages 53–65. Cambridge University Press, 1997.

[7] R. Dyckhoff and L. Pinto. Cut-Elimination and a Permutation-Free Sequent Calculus for Intuitionistic Logic. *Studia Logica*, 60(1):107–118, 1998.

[8] R. Dyckhoff and C. Urban. Strong Normalisation of Herbelin's Explicit Substitution Calculus with Substitution Propagation. In P. Lescanne, editor, *Proceedings of the Fourth Workshop on Explicit Substitutions Theory and Applications*, Logic Group Preprint Series No. 210, pages 26–45. Institute of Philosophy, Utrecht University, 2001.

[9] J. C. Espírito Santo. Revisiting the Correspondence between Cut Elimination and Normalisation. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Proceedings of the 27th Int. Coll. Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 600–611. Springer-Verlag, 2000.

[10] J. C. Espírito Santo. *Conservative Extensions of the λ-calculus for the Computational Interpretation of Sequent Calculus*. PhD thesis, Division of Informatics, University of Edinburgh, June 2002.

[11] M. J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[12] H. Herbelin. A λ-calculus Structure Isomorphic to Sequent Calculus Structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of the 1994 conference on Computer Science Logic*, volume 933 of *LNCS*, pages 67–75. Springer Verlag, 1995.

[13] H. Herbelin. Explicit Substitutions and Reducibility. *Journal of Logic and Computation*, 11(3):429–449, 2001.

[14] F. Joachimski and R. Matthes. Short Proofs of Normalisation. Archive for Mathematical Logic (to appear). Preprint, 1999.

[15] P. Lescanne. From $\lambda\sigma$ to λv: a Journey through Calculi of Explicit Substitutions. In H.-J. Boehm, editor, *Proceedings of the 21st Annual Symposium on Principles of Programming Languages*, pages 60–69. ACM Press, 1994.

[16] R. Mayr and T. Nipkow. Higher-Order Rewrite Systems and their Confluence. *Theoretical Computer Science*, 192:3–29, 1998.

[17] K. H. Rose. Explicit Substitution: Tutorial & Survey. Technical report, BRICS, Department of Computer Science, University of Aarhus, 1996.

[18] F. v. Raamsdonk and P. Severi. On Normalisation. Technical report, TR CS-R9545, Centrum voor Wiskunde en Informatica, Amsterdam, 1996. Later incorporated into [19].

[19] F. v. Raamsdonk, P. Severi, M. H. Sörensen, and H. Xi. Perpetual Reductions in Lambda Calculus. *Information and Computation*, 149:173–225, 1999.

[20] R. Vestergaard and J. Brotherston. A Formalised First-Order Confluence Proof for the λ-Calculus using One-Sorted Variable Names. In A. Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, pages 306–321. Springer-Verlag, 2001.

[21] R. Vestergaard and J. Wells. Cut Rules and Explicit Substitutions. *Mathematical Structures in Computer Science*, 11(1):131–168, 2000.