Revisiting Cut-Elimination: One Difficult Proof Is Really a Proof

Christian Urban[1] and Bozhi Zhu[2]

[1] Technical University of Munich
[2] North China Electric Power University

Abstract. Powerful proof techniques, such as logical relation arguments, have been developed for establishing the strong normalisation property of term-rewriting systems. The first author used such a logical relation argument to establish strong normalising for a cut-elimination procedure in classical logic. He presented a rather complicated, but informal, proof establishing this property. The difficulties in this proof arise from a quite subtle substitution operation, which implements proof transformation that permute cuts over other inference rules. We have formalised this proof in the theorem prover Isabelle/HOL using the Nominal Datatype Package, closely following the informal proof given by the first author in his PhD-thesis. In the process, we identified and resolved a gap in one central lemma and a number of smaller problems in others. We also needed to make one informal definition rigorous. We thus show that the original proof is indeed a proof and that present automated proving technology is adequate for formalising such difficult proofs.

1 Introduction

Proofs about syntax are often not very deep; rather the difficulties arise from the huge amount of details. Human reasoners seem to be ill-equipped to cope with such amounts of details. This observation is based on the experience obtained with a formalisation [18] of a paper on LF by Harper and Pfenning [6]. Their paper contained many informal proofs spread over more than 30 pages. The formalisation revealed a gap in one of the proofs and a small number of minor lacunae in others. Also in the present paper we describe a formalisation of an informal 20-page proof given by the first author [14] (see also [17]). This proof claims to establish a strong normalisation result of cut-elimination in classical logic. However, this formalisation, too, uncovers a number of errors in the informal proof, including one that required to restate two central lemmas.

In the literature there are numerous informal proofs for the termination of various cut-elimination procedures. One of the main applications of these procedures is to ensure consistency of sequent-calculi, that means that there is no proof for the sequent $\vdash \bot$. Gentzen [5] was the first who proved in this way the consistency of a sequent-calculus for intuitionistic and classical logic. Most of such cut-elimination procedures, including Gentzen's original, are weakly normalising, i.e., they employ a particular cut-elimination procedure strategy. While for establishing consistency a weakly normalising procedure is usually sufficient, if one wants to do computations with sequent proofs then strong normalisation is a more useful property. One reason for this is that

A. Voronkov (Ed.): RTA 2008, LNCS 5117, pp. 409–424, 2008.

cut-elimination in classical logic is *not* confluent and therefore one might reach different cut-free proofs by reducing cuts in a different order or applying different reduction rules.

For the purpose of calculating the collection of all cut-free proofs reachable from a classical proof, the first author introduced in [14,17] a strongly normalising procedure for cut-elimination. Note that simply taking an unrestricted version of Gentzen's cut-elimination procedure, that is removing the strategy, leads to infinite reduction sequences. Therefore the strongly-normalising cut-elimination procedure in [14,17] uses Gentzen's original rules for logical cuts, but modifies the rules for commuting cuts. (An instance of the cut-rule is said to be a *logical* cut when both cut-formulae are introduced by axioms or logical inference rules; otherwise the cut is said to be a *commuting* cut.) An interesting feature of this procedure is that it allows commuting cuts to pass over other cuts. It achieves strong normalisation by restricting the rules for commuting cuts so that they must "transport" in *one* step a commuting cut to *all* places where the corresponding cut-formula is introduced (Gentzen defined for this process local reduction rules, which only rewrite neighboring inference rules in a proof). As a result one ends up with a quite general reduction system for cut-elimination: for example it can simulate β-reduction in the λ-calculus [14].

Unfortunately, the generality of the reduction system means also that strong normalisation is much more difficult to prove. Our proof establishing this property is based on symmetric reducibility candidates [2], a powerful proof technique from the term-rewriting literature. To present the proof in a convenient form, sequent proofs are annotated with terms and the cut-elimination procedure is defined as a term-rewriting system. In particular, the proof transformation for commuting cuts is expressed as a special sort of proof substitution.

The disadvantage of using terms is that in order to deal with them in a convenient manner they nearly always need to be quotiented modulo α-equivalence—for example in order to have capture-avoiding substitution being definable as a total function. However, this quotienting makes (formal) reasoning much harder: inductions and recursions over the structure of α-equated terms are not immediately defined concepts; that means one has to spend some effort to derive them (in contrast with unquotient, or raw, terms where these concepts are for "free"). Moreover function definitions need to respect α-equivalence. This precludes, for example, the definition of the function that returns the immediate subterms of an α-equated term [15]. When working with such terms, one also often employs an informal variable convention [3] without giving a proper justification for its validity. By using this convention, one does not consider truly arbitrary bound variables, as *required* by the induction principles, but rather bound variables about which various freshness assumptions are made. Such reasoning is in general however unsound (see [16] for an example).

In informal "pencil-and-paper" proofs such problems are usually ignored. While this is harmless in easy proofs of simple properties, in difficult ones ignoring such problems carries the danger of overlooking errors (see [8, Page 16] for one overlooked by Kleene). Since the proof given by the first author for the strong-normalisation property is quite difficult and since a number of researchers have built their results directly on the strong-normalisation property (for example the *lemuridæ* system [4] and the typed version of

the $\mathcal{X}$-calculus [20]) or adapted the same proof-technique to other rewrite systems [21], it seems prudent to reconsider whether the original informal proof is actually a proof.

The Nominal Datatype Package [19] provides an infrastructure for reasoning conveniently about datatypes with a built-in notion of α-equivalence: it allows to specify such datatypes, provides appropriate recursion combinators and derives strong induction principles that have the usual variable convention already built-in. The latter comes with safeguards that make the variable convention a safe reasoning principle.

The main contribution of this paper is a complete formalisation of a difficult strong normalisation proof.[1] This formalisation uncovers a number of errors in the informal proof and makes one informal definition from the infromal proof more rigorous. The techniques used for the latter are applicable also in other calculi where non-trivial operation need to be defined over terms with involving binders. In the formalisation we also encounter some difficulties with a standard formulation for notion of strong normalisation. The rest of the paper is organised as follows: Sec. 2 reviews the informal proof, that is the definitions for terms, typing-rules and cut-elimination reductions given in [14,17]. The details about the formalisation are given in Sec. 3; Sec. 4 concludes and gives suggestions for further work.

2 Sequent Proofs and Cut-Elimination

The main idea behind the cut-elimination procedure presented in [14,17] is to transport one subderivation of a commuting cut to the place(s) where the cut-formula is introduced. To specify this operation, we used terms to annotate sequent proofs, whose inference rules are inspired by Kleene's sequent calculus G3a [7] and the sequent calculus G3c of [13]. These terms encode the structure of a proof and are defined as:

$$\begin{array}{rcll} M, N ::= & \mathsf{Ax}(x, a) & \text{Axiom} & \\ \mid & \mathsf{Cut}(\langle a\rangle M, (x)N) & \text{Cut} & \\ \mid & \mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, c) & \text{And-R} & \\ \mid & \mathsf{And}^i_L((x)M, y) & \text{And-L}_i & (i = 1, 2) \\ \mid & \mathsf{Or}^i_R(\langle a\rangle M, b) & \text{Or-R}_i & (i = 1, 2) \\ \mid & \mathsf{Or}_L((x)M, (y)N, z) & \text{Or-L} & \\ \mid & \mathsf{Imp}_R((x)\langle a\rangle M, b) & \text{Imp-R} & \\ \mid & \mathsf{Imp}_L(\langle a\rangle M, (x)N, y) & \text{Imp-L} & \end{array} \tag{1}$$

where x, y, z are taken from a set of *names* and a, b, c from a set of *co-names*. We use round brackets to signify that a name becomes bound and angle brackets that a co-name becomes bound.

Our sequents, or *typing judgements*, are of the form $\Gamma \triangleright M \triangleright \Delta$, where Γ is a left-context, M a term and Δ a right-context. The inference rules for those typing judgements are given in Fig. 1. One distinguishing feature of this term-calculus is that the structural rules, weakening and contraction, are completely implicit in the form of the inference rules. Thus we regard contexts as sets of (label,formula)-pairs, as in type theory, and *not* as multisets, as in LK or LJ. A label is either a name (for left-contexts) or a co-name (for right-contexts).

[1] Available at `http://isabelle.in.tum.de/nominal`.

$$\frac{}{x : B, \Gamma \triangleright \mathsf{Ax}(x, a) \triangleright \Delta, a : B}$$

$$\frac{x : B_i, \Gamma \triangleright M \triangleright \Delta}{y : B_1 \wedge B_2, \Gamma \triangleright \mathsf{And}^i_L((x)M, y) \triangleright \Delta}\ \wedge_{L_i} \qquad \frac{\Gamma \triangleright M \triangleright \Delta, a : B \quad \Gamma \triangleright N \triangleright \Delta, b : C}{\Gamma \triangleright \mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, c) \triangleright \Delta, c : B \wedge C}\ \wedge_R$$

$$\frac{x : B, \Gamma \triangleright M \triangleright \Delta \quad y : C, \Gamma \triangleright N \triangleright \Delta}{z : B \vee C, \Gamma \triangleright \mathsf{Or}_L((x)M, (y)N, z) \triangleright \Delta}\ \vee_L \qquad \frac{\Gamma \triangleright M \triangleright \Delta, a : B_i}{\Gamma \triangleright \mathsf{Or}^i_R(\langle a\rangle M, b) \triangleright \Delta, b : B_1 \vee B_2}\ \vee_{R_i}$$

$$\frac{\Gamma \triangleright M \triangleright \Delta, a : B \quad x : C, \Gamma \triangleright N \triangleright \Delta}{y : B \supset C, \Gamma \triangleright \mathsf{Imp}_L(\langle a\rangle M, (x)N, y) \triangleright \Delta}\ \supset_L \qquad \frac{x : B, \Gamma \triangleright M \triangleright \Delta, a : C}{\Gamma \triangleright \mathsf{Imp}_R((x)\langle a\rangle M, b) \triangleright \Delta, b : B \supset C}\ \supset_R$$

$$\frac{\Gamma_1 \triangleright M \triangleright \Delta_1, a : B \quad x : B, \Gamma_2 \triangleright N \triangleright \Delta_2}{\Gamma_1, \Gamma_2 \triangleright \mathsf{Cut}(\langle a\rangle M, (x)N) \triangleright \Delta_1, \Delta_2}\ Cut$$

Fig. 1. The inference, or typing, rules of our sequent calculus.

To see how our terms encode sequent proofs, suppose a sequent $\ldots A \vdash B \ldots$ can be proved. Then in our judgments, A and B have labels (say $x : A$ and $a : B$), and M would be an encoding of the proof of $\ldots A \vdash B \ldots$, with these labels, so denoted $\ldots x : A \triangleright M \triangleright a : B \ldots$ Then, where the sequent proof is extended further downwards, $x : A$ and $a : B$ might disappear from the contexts. At the point where they disappear, the corresponding proof-term includes the binding $(x)\_$ or $\langle a\rangle\_$, reflecting the fact that the choice of label (x or a) is not relevant to the proof as a whole.

To form contexts we have the following conventions: a context can only include a single association for each name (similarly for co-names); a comma in a conclusion stands for the set union and a comma in a premise stands for the *disjoint* set union. Consider for example the $\supset_R$-rule. This rule introduces the (co-name,formula)-pair $b : B \supset C$ in the conclusion, and consequently, b is a free co-name in the term $\mathsf{Imp}_R((x)\langle a\rangle M, b)$. However, b can already be free in the subterm M, in which case $b : B \supset C$ belongs to Δ. Thus the conclusion of the $\supset_R$-rule is of the form

$$\Gamma \triangleright \mathsf{Imp}_R((x)\langle a\rangle M, b) \triangleright \Delta \oplus b : B \supset C$$

where $\oplus$ denotes set union. Note that $x : B$ and $a : C$ in the premise are *not* part of the conclusion because they are intended to become bound. Hence the premise must be of the form

$$x : B \otimes \Gamma \triangleright M \triangleright \Delta \otimes a : C$$

where $\otimes$ denotes disjoint set union. Our cut-rule requires that two contexts are joined on each side of the conclusion. Thus we take this rule to be of the following form:

$$\frac{\Gamma_1 \vdash \Delta_1 \otimes a : B \quad x : B \otimes \Gamma_2 \vdash \Delta_2}{\Gamma_1 \oplus \Gamma_2 \vdash \Delta_1 \oplus \Delta_2}\ Cut$$

Next we focus on the cut-elimination rules. For this consider the following logical cut:

$$\dfrac{\dfrac{\Gamma_1 \vdash \Delta_1, B \quad \Gamma_1 \vdash \Delta_1, C}{\Gamma_1 \vdash \Delta_1, B \wedge C}\,\wedge_R \quad \dfrac{B, \Gamma_2 \vdash \Delta_2}{B \wedge C, \Gamma_2 \vdash \Delta_2}\,\wedge_{L_1}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,Cut$$

where we omitted for better readability the labels and term-annotations. We expect this cut to reduce to

$$\dfrac{\Gamma_1 \vdash \Delta_1, B \quad B, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,Cut\,.$$

However because of our implicit treatment of the structural rules, some care is needed: we have to ensure that the cut-formula $B \wedge C$ does not occur in Δ_1 or Γ_2. If it does, then we have not a logical cut, but a commuting cut. In order to distinguish between both kinds of cuts, we introduce the notion when a term introduces freshly a name or a co-name (this corresponds to the usual definition of a main formula in an inference rule).

Definition 1. *A term, M, introduces the name z or co-name c, if and only if M is of the form*

for z:		*for c:*	
	$\mathsf{Ax}(z, c)$		$\mathsf{Ax}(z, c)$
	$\mathsf{And}^i_L(\langle x\rangle S, z)$		$\mathsf{And}_R(\langle a\rangle S, \langle b\rangle T, c)$
	$\mathsf{Or}_L(\langle x\rangle S, \langle y\rangle T, z)$		$\mathsf{Or}^i_R(\langle a\rangle S, c)$
	$\mathsf{Imp}_L(\langle a\rangle S, \langle x\rangle T, z)$		$\mathsf{Imp}_R(\langle x\rangle\langle a\rangle S, c)$

A term freshly introduces a name, if and only if none of its proper subterms introduces this name. In other words, the name must not be free in a proper subterm. Similarly for co-names.

Armed with this definition we can state the cut-reduction rules for dealing with logical cuts (they correspond to Gentzen's rules for logical cuts):

Definition 2 (Reductions for Logical Cuts, $i = 1, 2$)

$\mathsf{Cut}(\langle b\rangle \mathsf{And}_R(\langle a_1\rangle M_1, \langle a_2\rangle M_2, b), \langle y\rangle \mathsf{And}^i_L(\langle x\rangle N, y)) \longrightarrow \mathsf{Cut}(\langle a_i\rangle M_i, \langle x\rangle N)$
if $\mathsf{And}_R(\langle a_1\rangle M_1, \langle a_2\rangle M_2, b)$ *and* $\mathsf{And}^i_L(\langle x\rangle N, y)$ *freshly introduce b and y, resp.*

$\mathsf{Cut}(\langle b\rangle \mathsf{Or}^i_R(\langle a\rangle M, b), \langle y\rangle \mathsf{Or}_L(\langle x_1\rangle N_1, \langle x_2\rangle N_2, y)) \longrightarrow \mathsf{Cut}(\langle a\rangle M, \langle x_i\rangle N_i)$
if $\mathsf{Or}^i_R(\langle a\rangle M, b)$ *and* $\mathsf{Or}_L(\langle x_1\rangle N_1, \langle x_2\rangle N_2, y)$ *freshly introduce b and y, resp.*

$\mathsf{Cut}(\langle b\rangle \mathsf{Imp}_R(\langle x\rangle\langle a\rangle M, b), \langle z\rangle \mathsf{Imp}_L(\langle c\rangle N, \langle y\rangle P, z))$
$\longrightarrow \mathsf{Cut}(\langle a\rangle \mathsf{Cut}(\langle c\rangle N, \langle x\rangle M), \langle y\rangle P)$ *or*
$\longrightarrow \mathsf{Cut}(\langle c\rangle N, \langle x\rangle \mathsf{Cut}(\langle a\rangle M, \langle y\rangle P))$
if $\mathsf{Imp}_R(\langle x\rangle\langle a\rangle M, b)$ *and* $\mathsf{Imp}_L(\langle c\rangle N, \langle y\rangle P, z)$ *freshly introduce b and z, resp.*

$\mathsf{Cut}(\langle a\rangle M, \langle x\rangle \mathsf{Ax}(x, b)) \longrightarrow M[a \mapsto b]$ *if M freshly introduces a*

$\mathsf{Cut}(\langle a\rangle \mathsf{Ax}(y, a), \langle x\rangle M) \longrightarrow M[x \mapsto y]$ *if M freshly introduces x*

In this definition we use $M[a \mapsto b]$ to stand for capture-avoiding renaming of a to b in M (similarly $M[x \mapsto y]$ for names).

The definition of the reduction rules for dealing with commuting cuts is more subtle. Consider the following proof (where again we left out the labels and annotations):

$$\pi_1 \left\{ \dfrac{\dfrac{\dfrac{}{A \vdash B \supset C, A^\bullet}\quad \dfrac{A, B \vdash C, A^\bullet}{A \vdash B \supset C, A}\ \supset_R}{A \vee A \vdash B \supset C, A}\ \vee_L \quad \dfrac{\dfrac{A^\star \vdash D, A \quad A^\star \vdash D, A}{A \vdash D, A \wedge A}\ \wedge_R \quad \dfrac{A^\star, E \vdash A \quad A^\star, E \vdash A}{A, E \vdash A \wedge A}\ \wedge_R}{A, D \supset E \vdash A \wedge A}\ \supset_L}{A \vee A, D \supset E \vdash B \supset C, A \wedge A}\ Cut \right\} \pi_2$$

The cut-formula A is neither a main formula in the inference rule $\vee_L$, nor in $\supset_L$ (on the term-level that means that the terms for π_1 and π_2 do not freshly introduce the name and co-name corresponding for A). Therefore the cut is a commuting cut. In π_1 the cut-formula is a main formula in the axioms marked with a bullet; in π_2, respectively, in the axioms marked with a star. Eliminating the cut in the proof above means to either transport the derivation π_2 to the places marked with a bullet and "cut it against" the corresponding axioms, or to transport π_1 and "cut it against" the axioms marked with a star. In both cases the derivation being transported is duplicated. We realise these operations with two symmetric forms of substitution, which we shall write as $P\{x := \langle a \rangle Q\}$ and $S\{b := (y)T\}$.

Whenever such a substitution is "next" to a term in which the cut-formula is introduced, then the substitution becomes an instance of the Cut-term constructor. In the following two examples we shall write $\{\sigma\}$ and $\{\tau\}$ for the substitutions $\{c := (x)P\}$ and $\{x := \langle b \rangle Q\}$, respectively.

$$\begin{array}{lcl} \mathsf{And}_R(\langle a \rangle M, \langle b \rangle N, c)\{\sigma\} & = & \mathsf{Cut}(\langle c \rangle \mathsf{And}_R(\langle a \rangle\, M\{\sigma\}, \langle b \rangle\, N\{\sigma\}, c), (x)P) \\ \mathsf{Imp}_L(\langle a \rangle M, (y)N, x)\{\tau\} & = & \mathsf{Cut}(\langle b \rangle Q, (x)\mathsf{Imp}_L(\langle a \rangle\, M\{\tau\}, (y)\, N\{\tau\}, x)) \end{array}$$

In the first term the formula labelled with c is the main formula and in the second the one labelled with x. So in both cases the substitutions "expand" to cuts, and in addition, the substitutions are pushed inside the subterms. This is because there might be several occurrences of c and x: both labels need not have been freshly introduced. We are left with specifying the cases where the name or co-name that is being substituted for is not a label of the main formula. In these cases the substitutions are pushed inside the subterms or vanish in case of the axioms. Fig. 2 gives all clauses for the cases where a cut expands and the clauses for when a substitution is pushed inside the terms. We do not need to worry about inserting contraction rules when a term is duplicated, since our contexts are sets of labelled formulae, and thus contractions are made implicitly.

There is one point worth mentioning about the clauses marked with $\star$ in Fig. 2. Although these clauses are not needed for strong normalisation, they are needed to have the property

$$M\{x := \langle a \rangle P\}\{b := (y)Q\} \;=\; M\{b := (y)Q\}\{x := \langle a \rangle P\}$$

for b not free in $\langle a \rangle P$ and x not free in $(y)Q$. This property is crucial in our strong normalisation proof. However, this property does *not* hold for a slightly simpler definition of the substitution operation where the lines marked with $\star$ are deleted and the first two clauses are replaced by $\mathsf{Ax}(x, c)\{c := (y)P\} \stackrel{\text{def}}{=} P[y \mapsto x]$ and $\mathsf{Ax}(y, a)\{y :=$

$$
\begin{array}{rcl}
\mathsf{Ax}(x,c)\{c{:=}(y)P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle \mathsf{Ax}(x,c), (y)P) \\
\mathsf{Ax}(y,a)\{y{:=}\langle c\rangle P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle P, (y)\mathsf{Ax}(y,a)) \\
\mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, c)\{c{:=}(y)P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle \mathsf{And}_R(\langle a\rangle M\{c{:=}(y)P\}, \langle b\rangle N\{c{:=}(y)P\}, c), (y)P) \\
\mathsf{And}^i_L((x)M, y)\{y{:=}\langle c\rangle P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle P, (y)\mathsf{And}^i_L((x)M\{y{:=}\langle c\rangle P\}, y)) \\
\mathsf{Or}^i_R(\langle a\rangle M, c)\{c{:=}(y)P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle \mathsf{Or}^i_R(\langle a\rangle M\{c{:=}(y)P\}, c), (y)P) \\
\mathsf{Or}_L((x)M, (y)N, z)\{z{:=}\langle c\rangle P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle P, (z)\mathsf{Or}_L((x)M\{z{:=}\langle c\rangle P\}, (y)N\{z{:=}\langle c\rangle P\}, z)) \\
\mathsf{Imp}_R((x)\langle a\rangle M, b)\{b{:=}(y)P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle b\rangle \mathsf{Imp}_R((x)\langle a\rangle M\{b{:=}(y)P\}, b), (y)P) \\
\mathsf{Imp}_L(\langle a\rangle M, (x)N, y)\{y{:=}\langle c\rangle P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle c\rangle P, (y)\mathsf{Imp}_L(\langle a\rangle M\{y{:=}\langle c\rangle P\}, (x)N\{y{:=}\langle c\rangle P\}, y)) \\
\mathsf{Cut}(\langle a\rangle \mathsf{Ax}(x,a), (y)M)\{x{:=}\langle b\rangle P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle b\rangle P, (y)M\{x{:=}\langle b\rangle P\}) \quad \star \\
\mathsf{Cut}(\langle a\rangle M, (x)\mathsf{Ax}(x,b))\{b{:=}(y)P\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle a\rangle M\{b{:=}(y)P\}, (y)P) \quad \star
\end{array}
$$

Otherwise we push the substitution inside the subterms

$$
\begin{array}{rcl}
\mathsf{Ax}(x,a)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Ax}(x,a) \\
\mathsf{Cut}(\langle a\rangle M, (x)N)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Cut}(\langle a\rangle\ M\{\sigma\}, (x)\ N\{\sigma\}) \\
\mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, c)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{And}_R(\langle a\rangle\ M\{\sigma\}, \langle b\rangle\ N\{\sigma\}, c) \\
\mathsf{And}^i_L((x)M, y)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{And}^i_L((x)\ M\{\sigma\}, y) \\
\mathsf{Or}^i_R(\langle a\rangle M, b)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Or}^i_R(\langle a\rangle\ M\{\sigma\}, b) \\
\mathsf{Or}_L((x)M, (y)N, z)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Or}_L((x)\ M\{\sigma\}, (y)\ N\{\sigma\}, z) \\
\mathsf{Imp}_R((x)\langle a\rangle M, b)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Imp}_R((x)\langle a\rangle\ M\{\sigma\}, b) \\
\mathsf{Imp}_L(\langle a\rangle M, (x)N, y)\{\sigma\} & \stackrel{\text{def}}{=} & \mathsf{Imp}_L(\langle a\rangle\ M\{\sigma\}, (x)\ N\{\sigma\}, y)
\end{array}
$$

Fig. 2. Definition of the substitution operation. This operation is used in the cut-reduction dealing with commuting cuts. For more details see [14,17].

$\langle c\rangle P\} \stackrel{\text{def}}{=} P[c \mapsto a]$. This simpler definition corresponds to the more familiar method how cuts are eliminated. A consequence of the lines marked with $\star$, as we shall see, is that calculations and properties involving substitution are quite subtle.

However, we are now in a position to complete the definition of our cut-elimination procedure by stating how commuting cuts reduce, namely:

Definition 3 (Reductions for Commuting Cuts)

$$
\begin{array}{rcll}
\mathsf{Cut}(\langle a\rangle M, (x)N) & \longrightarrow & M\{a{:=}(x)N\} & \textit{if } M \textit{ does not freshly introduce } a, \ \textbf{or} \\
 & \longrightarrow & N\{x{:=}\langle a\rangle M\} & \textit{if } N \textit{ does not freshly introduce } x
\end{array}
$$

and close the reduction relation under term-formation. The important properties of this cut-elimination procedure are subject-reduction and strong normalisation.

Theorem 1 (Subject Reduction and Strong Normalisation [14,17])

- *If* $\Gamma \triangleright M \triangleright \Delta$ *and* $M \longrightarrow M'$ *then* $\Gamma \triangleright M' \triangleright \Delta$.
- *If* $\Gamma \triangleright M \triangleright \Delta$ *then* M *is strongly normalising w.r.t.* $\longrightarrow$.

3 The Formalisation of the Strong Normalisation Proof

In this section we describe the formalisation of the strong normalisation proof. While the informal description of this proof is already quite detailed—the details are spread over more than 20 pages in [14], we found that several subtle points were overlooked, one central lemma is faulty and has to be restated, and a definition has to be made rigorous.

The definition of the terms given in (1) and formulae (which we omitted in this paper) pose no problem for the Nominal Datatype Package, as it was designed to deal with such definitions. From the definition of terms, the package derives automatically a weak and a strong structural induction principle (the strong one has the variable convention already built in [19]), and provides a recursion combinator for defining functions over the structure of the terms [15]. With this combinator, it is easy to define the capture-avoiding renaming functions $M[a \mapsto b]$ and $M[x \mapsto y]$, although these definitions require that several proof-obligations are discharged by the user (the proof-obligations ensure that the renaming-functions preserve α-equivalence).

The typing-system rules given in Fig. 1 can also be formalised with ease, except that we have chosen to represent typing-context as (label,formula)-lists rather than sets. This requires that we add appropriate validity and freshness-constraints to the inference rules. A context is defined to be *valid* provided no name or co-name occurs twice. This can be stated with the rules:

$$\frac{}{valid([])} \qquad \frac{a \# \Delta \quad valid(\Delta)}{valid((a : B) :: \Delta)} \qquad \frac{x \# \Gamma \quad valid(\Gamma)}{valid((x : B) :: \Gamma)}$$

where $a \# \Delta$ (similarly $x \# \Gamma$) stands for a being fresh for Δ (i.e. not occurring in Δ). Using this definition and freshness, the axiom and $\wedge_R$-rules, for example, look in the formalisation as follows:

$$\frac{valid(\Gamma) \quad valid(\Delta) \quad (x : B) \in \Gamma \quad (a : B) \in \Delta}{\Gamma \triangleright \mathsf{Ax}(a, b) \triangleright \Delta}$$

$$\frac{\begin{array}{c}\Gamma \triangleright M \triangleright (a : B) :: \Delta \quad \Gamma \triangleright N \triangleright (b : C) :: \Delta \\ a \# \Delta \quad b \# \Delta \quad \Delta' \approx (c : B \wedge C) :: \Delta \quad valid(\Delta')\end{array}}{\Gamma \triangleright \mathsf{And}_R(\langle a \rangle M, \langle b \rangle N, c) \triangleright \Delta'} \wedge_R$$

where $\approx$ stands for two lists being equal if regarded as sets.

Most of the effort during the formalisation we had to invest in defining the substitution operation and proving associated lemmas. One reason for this is that the informal definition given in Fig. 2 makes from a formal point of view only little sense. For example, merging the two clauses given for $\{c := (x)P\}$ and the term-constructor $\mathsf{And}_R(\langle a \rangle M, \langle b \rangle N, d)$ into an *if*-statement (as is required in a formal definition by recursion over the structure of terms) leads to:

$$\begin{array}{l}\mathsf{And}_R(\langle a \rangle M, \langle b \rangle N, d)\{c := (x)P\} \stackrel{\text{def}}{=} \\ \quad \mathit{if}\ c = d \\ \quad\quad \mathit{then}\ \mathsf{Cut}(\langle d \rangle \mathsf{And}_R(\langle a \rangle (M\{c := (x)P\}), \langle b \rangle (N\{c := (x)P\}), d), (x)P) \\ \quad\quad \mathit{else}\ \mathsf{And}_R(\langle a \rangle (M\{c := (x)P\}), \langle b \rangle (N\{c := (x)P\}), d)\end{array}$$

where the "true"-branch corresponds to the case where the substitution expands to a cut, and the "false"-branch where the substitution is just pushed inside the subterms M and N. The obvious problem is that we attempt to push a substitution under binders (in this example the binders are $\langle a\rangle\_$ and $\langle b\rangle\_$). This is only possible provided a and b do not occur freely in the term P. Hence we have to restrict the clause with suitable preconditions, namely:

$$\begin{array}{l}\text{provided } a \# P \text{ and } b \# P \text{ then}\\ \quad \mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, d)\{c:=(x)P\} \stackrel{\text{def}}{=}\\ \qquad \mathit{if}\ c = d\\ \qquad\quad \mathit{then}\ \mathsf{Cut}(\langle d\rangle \mathsf{And}_R(\langle a\rangle(M\{c:=(x)P\}), \langle b\rangle(N\{c:=(x)P\}), d), (x)P)\\ \qquad\quad \mathit{else}\ \mathsf{And}_R(\langle a\rangle(M\{c:=(x)P\}), \langle b\rangle(N\{c:=(x)P\}), d)\end{array}$$

Since we define substitution over α-equivalence classes, we still obtain a total function with this restriction in place. The hope is that we can always rename $\mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, d)$ appropriately so that the preconditions are met. However, this is futile for the proof substitution operation, because in the "true"-branch also the (free) co-name d is bound with the scope of P—and we cannot rename (potentially) free co-names in a term without violating α-equivalence. The way out is to choose in the "true"-branch explicitly a fresh co-name d' and define the clause formally as

$$\begin{array}{l}\text{provided } a \# P \text{ and } b \# P \text{ then}\\ \quad \mathsf{And}_R(\langle a\rangle M, \langle b\rangle N, d)\{c:=(x)P\} \stackrel{\text{def}}{=}\\ \qquad \mathit{if}\ c = d\\ \qquad\quad \mathit{then}\ \mathit{fresh}\ (\lambda d'.\mathsf{Cut}(\langle d'\rangle \mathsf{And}_R(\langle a\rangle(M\{c:=(x)P\}), \langle b\rangle(N\{c:=(x)P\}), d'), (x)P))\\ \qquad\qquad \mathit{else}\ \mathsf{And}_R(\langle a\rangle(M\{c:=(x)P\}), \langle b\rangle(N\{c:=(x)P\}), d)\end{array}$$

using the fresh function defined in [10]. Space constraints prevent us to give more details about this function here, except that this function characterises when a construction that picks a fresh (co-)name is independent of which fresh (co-)name is chosen. While this clause (and similar ones for the other term-constructors) give us the properties we expect from the substitution operation (which defines cut-reductions), the corresponding definition leads to quite complicated proofs. One reason is that in the "true"-branches we need to find a fresh (co-)name so that the fresh function provides us with the desired result. At the moment this has to be done by hand, as the Nominal Datatype Package provides only little help for dealing conveniently with the fresh functions.

Having properly defined the substitution operation, we can prove facts about how substitutions interact; for example the following form of the substitution lemma is needed in several places in the proof:

Lemma 1 (Some Substitution Lemmas)

- *If* $x \# P$ *then* $M\{x:=\langle c\rangle \mathsf{Ax}(y,c)\}\{y:=\langle c\rangle P\} = M\{y:=\langle c\rangle P\}\{x:=\langle c\rangle P\}$
- *If* $x \# (y)Q$ *then* $M\{x:=\langle a\rangle P\}\{b:=(y)Q\} = M\{b:=(y)Q\}\{x:=\langle a\rangle(P\{b:=(y)Q\})\}$

where the first one is needed in the proof of the second (note that in the second $x \# (y)Q$ stands for $x = y$ or $x \# Q$). We prove such lemmas using the strong structural induction principle for terms, as this minimises the need for renaming bound names and co-names. Still these proofs require some considerable effort due to the sheer number of cases that

need to be analysed. In order to appreciate the difficulties involving the substitution operation for terms, note that the symmetric property for the first part of the lemma, namely

$$M\{y:=\langle c\rangle P\}\{x:=\langle c\rangle \mathsf{Ax}(y,c)\} = M\{y:=\langle c\rangle P\}\{x:=\langle c\rangle P\}$$

does not hold. The formalisation of properties about substitution requires approximately 20% of the formalisation code.

In comparison with the definition of the substitution operation, the definition of the cut-reduction relation is relatively simple. It relies on the auxiliary notions for when a term freshly introduces a name or co-name; for example

$$\frac{}{\mathit{fin}(\mathsf{Ax}(z,a),z)} \qquad \frac{z \# (x)M \quad z \# (y)N}{\mathit{fin}(\mathsf{Or}_L((x)M,(y)N,z),z)}$$

and so on for the notion of freshly introducing a name (similarly *fic* for freshly introducing a co-name). The formal definition of the cut-reductions look then as follows (the first two are examples for logical cuts; the last for a commuting cut):

$$\frac{\mathit{fic}(M,a)}{\mathsf{Cut}(\langle a\rangle M,(x)\mathsf{Ax}(x,b)) \longrightarrow M[a\mapsto b]}$$

$$\frac{\mathit{fic}(\mathsf{And}_R(\langle a_1\rangle M_1,\langle a_2\rangle M_2,b),b) \qquad \mathit{fin}(\mathsf{And}^1_L((x)N,y),y)}{\mathsf{Cut}(\langle b\rangle \mathsf{And}_R(\langle a_1\rangle M_1,\langle a_2\rangle M_2,b),(y)\mathsf{And}^1_L((x)N,y)) \longrightarrow \mathsf{Cut}(\langle a_1\rangle M_1,(x)N)}$$

$$\frac{\neg\mathit{fic}(M,a)}{\mathsf{Cut}(\langle a\rangle M,(x)N) \longrightarrow M\{a:=(x)N\}}$$

In addition to those rules we specified in the formalisation a slew of congruence rules, such as:

$$\frac{M \longrightarrow M'}{\mathsf{And}_R(\langle a\rangle M,\langle b\rangle N,c) \longrightarrow \mathsf{And}_R(\langle a\rangle M',\langle b\rangle N,c)}$$

$$\frac{N \longrightarrow N'}{\mathsf{And}_R(\langle a\rangle M,\langle b\rangle N,c) \longrightarrow \mathsf{And}_R(\langle a\rangle M,\langle b\rangle N',c)}$$

These rules were not explicitly mentioned in the informal proof. For the cut-reductions and *fin* (similarly *fic*) we need to establish the lemma:

Lemma 2. *If* $M \longrightarrow M'$ *and* $\mathit{fin}(M,x)$ *then* $\mathit{fin}(M',x)$.

This is relatively easy to prove by a strong induction over $M \longrightarrow M'$. We next establish important properties characterising the interactions between cut-reductions and substitution:

Lemma 3

- $M\{x:=\langle c\rangle \mathsf{Ax}(y,c)\} \longrightarrow^* M[x\mapsto y]$
- $M\{c:=(x)\mathsf{Ax}(x,d)\} \longrightarrow^* M[c\mapsto d]$
- *If* $M \longrightarrow M'$ *then* $M\{\sigma\} \longrightarrow^* M'\{\sigma\}$.

The first two properties are by strong structural inductions over M; the third is by strong induction over the reduction $M \longrightarrow M'$. All proofs require many case distinctions and rely on additional proofs relating substitutions and the inductively defined predicates fin and fic. The third property in this lemma is interesting insofar as it is quite un-intuitive considering a similar property for capture avoiding substitution in the lambda-calculus.[2] In the informal proof [14,17], this lemma was stated and proved as:

Lemma 4 (Faulty). *If $M \longrightarrow M'$ then either $M\{\sigma\} = M'\{\sigma\}$ or $M\{\sigma\} \longrightarrow M'\{\sigma\}$.*

The case where $M\{\sigma\} = M'\{\sigma\}$ was correctly analysed. It involves reductions of the form

$$\mathsf{Cut}(\langle a\rangle M, (x)\mathsf{Ax}(x, b)) \longrightarrow M[a \mapsto b]$$

with M being of the form $\mathsf{Ax}(y, a)$ and $\{\sigma\}$ being $\{y := \langle c\rangle P\}$. In this case $M\{y := \langle c\rangle P\}$ is defined as $\mathsf{Cut}(\langle c\rangle P, (x)\mathsf{Ax}(x, b))$, and $M'\{y := \langle c\rangle P\}$ as $\mathsf{Ax}(y, b)\{y := \langle c\rangle P\}$, which in turn is defined as $\mathsf{Cut}(\langle c\rangle P, (y)\mathsf{Ax}(y, b))$. Both terms are equal by α-equivalence.

However, the case where $M\{\sigma\}$ needs more than one reduction to reach $M'\{\sigma\}$ was overlooked! Such a case occurs with logical cuts, for example

$$\mathsf{Cut}(\langle b\rangle \mathsf{And}_R(\langle a_1\rangle M_1, \langle a_2\rangle M_2, b), (z)\mathsf{And}^1_L((x)N, z)) \longrightarrow \mathsf{Cut}(\langle a_1\rangle M_1, (x)N)$$

with the proviso that M_1 is of the form $\mathsf{Ax}(y, a_1)$. In this case the left-hand side $M\{\sigma\}$ is

$$\mathsf{Cut}(\langle b\rangle \mathsf{And}_R(\langle a_1\rangle \mathsf{Cut}(\langle c\rangle P, (y)\mathsf{Ax}(y, a_1)), \langle a_2\rangle M_2\{\sigma\}, b), (z)\mathsf{And}^1_L((x)N\{\sigma\}, z))$$

which in a single step reduces to

$$\mathsf{Cut}(\langle a_1\rangle \mathsf{Cut}(\langle c\rangle P, (y)\mathsf{Ax}(y, a_1)), (x)N\{\sigma\})$$

and in possibly more than one step reduces to

$$\mathsf{Cut}(\langle a_1\rangle P[c \mapsto a_1], (x)N)$$

which in turn is equal to $M'\{\sigma\}$. Since in the formalisation we have to go through every case one by one, such cases cannot be overlooked there. Fortunately, the proof of the more general lemma goes through. Fortunately, also, the more general property does not destroy the overall proof: the next lemma (Lem. 7 below) that uses this lemma can be modified to deal with the many step-reduction sequence.

Next the informal proof considered the notion for a term being strongly normalising. This notion was stated as all reductions sequences starting from a term must be finite. As the formal definition of a term M being strongly normalising we used the inductive definition:

$$\frac{\forall M'.M \longrightarrow M' \text{ implies } M' \in SN}{M \in SN} \tag{2}$$

This is a standard definition used in many formalisations. Two interesting phenomena arose however with this definition. One was that in the informal proof we stated in a passing (one-sentence) remark that the strong normalisation is preserved under renamings, namely

[2] In the λ-calculus the property is if $M \longrightarrow_\beta M'$ then $M\{\sigma\} \longrightarrow_\beta M'\{\sigma\}$.

Lemma 5. *If $M \in SN$ then $M[a\mapsto b] \in SN$ and also $M[x\mapsto y] \in SN$.*

This lemma is "obvious" because renaming cannot create any new redexes, or cuts (unlike the proof substitution which might create new cuts). Surprisingly, however, this fact caused us a lot of frustration in the formalisation and resulted in slightly more than 10%(!) of the formalisation code. The problem is that we know by induction hypothesis that $(\forall M'.M \longrightarrow M'$ implies $M' \in SN)$. We can further assume that for an M', $M[a \mapsto b]$ reduces to M', and we have to show that $M' \in SN$. To do so, we have to analyse how $M[a\mapsto b]$ reduces w.r.t. M. As a result we have to show a fact:

Lemma 6. *If $M[a \mapsto b] \longrightarrow M'$, then there exists an M_0 such that $M' = M_0[a \mapsto b]$ and $M \longrightarrow M_0$.*

Its proof needs to analyse all the term constructors and all the applicable reductions. This is extremely laborious. The problem is independent of our calculus and would also arise in the λ-calculus. The fact that the lemma is "obvious", but its proof is hard, seems to indicate that the definition shown in (2) is not the right definition for establishing Lemma 5. We have however not seen any better formal definition for strong normalisation in the term-rewriting literature.

Finally the informal proof establishes that all typable terms are strongly normalising. Surprisingly the symmetric candidates $[\![(B)]\!]$ and $[\![\langle B\rangle]\!]$ defined for this part of the proof do not create any difficulties (the corresponding definitions are therefore omitted here, see [14,17]). We show that the candidates are closed under reductions

Lemma 7 (Reduction Preserves Candidates)

- *If $\langle a\rangle M \in [\![\langle B\rangle]\!]$ and $M \longrightarrow^* M'$, then $\langle a\rangle M' \in [\![\langle B\rangle]\!]$.*
- *If $(x)M \in [\![(B)]\!]$ and $M \longrightarrow^* M'$, then $(x)M' \in [\![(B)]\!]$.*

In comparison with the informal proof, however, the assumptions in this lemma had to be strengthened to deal with many-step reductions (i.e. $\longrightarrow^*$) because of the flaw in the third part of Lemma 3. However, this generalisation does not affect the structural induction over B that is employed to establish Lemma 7. Now we can show how the candidates imply the property of strong normalisation, namely

Lemma 8

- *If $\langle a\rangle M \in [\![\langle B\rangle]\!]$, then $M \in SN$;*
- *If $(x)M \in [\![(B)]\!]$, then $M \in SN$.*

The last difficult lemma spells out the conditions when a cut is strongly normalising, namely:

Lemma 9. *If $M, N \in SN$ and $\langle a\rangle M \in [\![\langle B\rangle]\!]$, $(x)N \in [\![(B)]\!]$ then*

$$\mathsf{Cut}(\langle a\rangle M, (x)N) \in SN\ .$$

The informal proof of this lemma is inspired by a technique of Prawitz [11]. It proceeds by induction over a lexicographically ordered induction value of the form (δ, μ, ν) where δ is the size of the cut-formula B; μ and ν are the longest reductions sequences

starting from M and N. Because of the assumptions that $M \in SN$and $N \in SN$, the informal proof claims without proof that these maximal lengths must be finite and the induction therefore is sensible.

Here arises, however, the second problem with the definition of strong normalisation shown in (2): while this claim is indeed true for the reduction system at hand, it is not true in general. The reason is that a strongly normalising term does not need to have an upper bound for the longest reduction sequence: consider the term M that reduces in one step to the normal form M_1, but also in *two* steps to the normal form M_2 and so on for any n. For this term we have that every reduction sequence starting from M is finite, but there is no upper bound for the length of the longest reduction sequence starting from M. This problem does not arise in our reduction system, because $\longrightarrow$ is only finitely branching. Together with the König's lemma one can then infer that a longest reduction sequence indeed exists for every strongly normalising term. The problem with this argument, however, is that establishing that $\longrightarrow$ is only finitely branching is far from trivial and also a formalisation of König's lemma is not readily available in Isabelle.

We were able to completely avoid the work involved with this argument by performing a well-founded induction, not on the triple using the longest reduction sequence, but directly on the predicate SN (which is well-founded). This change in the induction value does not require any changes to the other arguments in the proof. Though we had to supply details for cases which were not present in the informal proof and which were not like the other cases that were shown.

The final proof builds up a closing substitution for a well-typed term $\Gamma \triangleright M \triangleright \Delta$ and shows that M is strongly normalising under this closing substitution. While all these proofs involving candidates are quite laborious, they do not contain any surprises (except the point about the length of the longest reduction sequence of a strongly normalising term). Therefore we omit all details about them. The formalisation is part of the Nominal Datatype Package and can be downloaded from

`http://isabelle.in.tum.de/nominal`

4 Conclusion

We have described a formalisation of an informal proof establishing the strong-normalisation property of the cut-elimination procedure for classical logic given in [14,17]. Besides confirming that the informal proof is really a proof (all errors can be fixed), the purpose of this paper is to convey the point that such formalisations are feasible and the formal proving techniques are within reach of being useful in "everyday" reasoning (The distribution of the Nominal Datatype Package contains a number of other formalisations from a wide range of topics). The formalisation of the strong-normalisation proof was still quite demanding. However, given that the time formalising the informal proof (which was however already quite detailed) is roughly equal to the time finding and writing down this informal proof, then this additional effort seems more than acceptable to us. The additional time spent with formalising the proof ensures that no case is overlooked and that the definitions are rigorous. Also, having a formalisation of

the proof allows one to "play" with the definitions. This is in contrast with an informal proof where it is rather impractical to change any definition, since checking that the change does not affect the proof is "equal" to re-doing the proof. In contrast, we hope to be able to improve in the future upon the problems we encountered with the definition of strong normalisation. Once we find a more convenient alternative definition for strong normalisation, we can just re-run the formalisation and quickly focus on the places where the proof might break with the new definition.

Our formalisation is at the moment the biggest single-file formalisation in the whole Isabelle distribution. Its size is slightly more than 770 KByte. It took us approximately 5 person-weeks to complete the formalisation (including finding fixes for all the problems). The big size and speed with which the formalisation was completed is due to the fact that in cut-elimination proofs many cases are repetitive and only differ in details. So we were often able to complete one case and then cut-and-paste this case in place of the other cases. The copied code then often only needed tweaking to deal with slightly different assumptions and proof-obligations. The formalisation needs approximately 14 minutes to check on a standard laptop. Our work is now used for benchmarking Isabelle and also has proved to be a very useful testcase for any new features that are implemented in Isabelle.

The Nominal Datatype Package has been invaluable for proving properties about terms involving *simple*, lambda-calculus-like binders (our terms annotated to sequent-proofs are only slightly more complicated than the λ-terms annotated to natural deduction proofs). We note that de-Bruijn indices can be used in *principle* for such formalisations involving α-equated terms; but also note practical difficulties when several kinds of binders need to be treated and some binders even occur iterated in term-constructors (like in our Imp_R). In our opinion, a proof on the scale that we have done here employing de-Bruijn indices is not feasible, because of the complications arising from our substitution operation. Twelf, a system that provides an infrastructure for reasoning about higher-order abstract syntax—another existing technique for dealing with binders, seems not yet streamlined enough to deal conveniently with logical relation arguments on the scale that are used in the informal proofs above (See [12] for an approach about how to perform logical relation arguments in Twelf). The formalisation of a weakly normalising cut-elimination procedure done by Pfenning [9] using higher-order abstract syntax in Twelf does not seem to scale to our strong normalisation proof, as it is impossible to define our notion of symmetric reducibility candidates in Twelf. Also our proof is substantially more complex than the proof underlying the formalisation by Pfenning (he considers only weak normalisation). Aydemir *et al.* have reported recently [1] that a locally nameless representation for terms with binders has been very useful in formalising informal proofs from programming language theory. We have not yet been able to thoroughly compare their results with ours and do not know how their results scale to our quite difficult proof. For example, what helped us to avoid mistakes in our formalisation was that names and co-names have different type. As a result the type-system will immediately complain whenever we mixed up these names. It remains to be seen whether a locally nameless representation of terms can be defined so that formalisations have a similar convenience.

The most annoying aspect in our formalisation is the lack of automated support for dealing with the fresh function. Finding an appropriate fresh name or co-name that meets the conditions can be easily automated. However the verification of the conditions associated with the fresh function seems hard to automate. This and comparing our work with the one by Aydemir *et al.* we leave as future work.

Acknowledgements. The first author thanks Jeremy Dawson and Michael Norrish who gave helpful hints to formalise Lemma 9. Markus Wenzel and Stefan Berghofer streamlined Isabelle to cope with the size of the formalisation.

References

1. Aydemir, B., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering Formal Metatheory. In: Proc. of the 35rd Symposium on Principles of Programming Languages (POPL), pp. 3–15. ACM, New York (2008)
2. Barbanera, F., Berardi, S.: A Symmetric Lambda Calculus for "Classical" Program Extraction. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 495–515. Springer, Heidelberg (1994)
3. Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics, vol. 103. North-Holland, Amsterdam (1981)
4. Brauner, P., Houtmann, C., Kirchner, C.: Principles of Superdeduction. In: Proc. of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 41–50 (2007)
5. Gentzen, G.: Untersuchungen über das logische Schließen I and II. Mathematische Zeitschrift 39, 176–210, 405–431 (1935)
6. Harper, R., Pfenning, F.: On Equivalence and Canonical Forms in the LF Type Theory. ACM Transactions on Computational Logic 6(1), 61–101 (2005)
7. Kleene, S.C.: Introduction to Metamathematics. North-Holland, Amsterdam (1952)
8. Kleene, S.C.: Disjunction and Existence Under Implication in Elementary Intuitionistic Formalisms. Journal of Symbolic Logic 27(1), 11–18 (1962)
9. Pfenning, F.: Structural Cut Elimination. Information and Computation 157(1–2), 84–141 (2000)
10. Pitts, A.: Alpha-Structural Recursion and Induction. Journal of the ACM 53, 459–506 (2006)
11. Prawitz, D.: Ideas and Results of Proof Theory. In: Proceedings of the 2nd Scandinavian Logic Symposium. Studies in Logic and the Foundations of Mathematics, vol. 63, pp. 235–307. North-Holland, Amsterdam (1971)
12. Schürmann, C., Sarnat, J.: Towards a Judgemental Reconstruction of Logical Relation Proofs. In: Proc. of the 23rd IEEE Symposium on Logic in Computer Science (LICS) (to appear, 2008)
13. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, vol. 43. Cambridge University Press, Cambridge (1996)
14. Urban, C.: Classical Logic and Computation. PhD thesis, Cambridge University (October 2000)
15. Urban, C., Berghofer, S.: A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 498–512. Springer, Heidelberg (2006)
16. Urban, C., Berghofer, S., Norrish, M.: Barendregt's Variable Convention in Rule Inductions. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 35–50. Springer, Heidelberg (2007)

17. Urban, C., Bierman, G.: Strong Normalisation of Cut-Elimination in Classical Logic. Fundamenta Informaticae 45(1–2), 123–155 (2001)
18. Urban, C., Cheney, J., Berghofer, S.: Mechanizing the Metatheory of LF. In: Proc. of the 23rd IEEE Symposium on Logic in Computer Science (LICS). Technical report (to appear, 2008), http://isabelle.in.tum.de/nominal/LF
19. Urban, C., Tasson, C.: Nominal Techniques in Isabelle/HOL. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 38–53. Springer, Heidelberg (2005)
20. van Bakel, S., Lengrand, S., Lescanne, P.: The Language X: Circuits, Computations and Classical Logic. In: Coppo, M., Lodi, E., Pinna, G.M. (eds.) ICTCS 2005. LNCS, vol. 3701, pp. 81–96. Springer, Heidelberg (2005)
21. Yoshida, N., Berger, M., Honda, K.: Strong Normalisation in the π-Calculus. In: Proc. of the 16th IEEE Symposium on Logic in Computer Science (LICS), pp. 311–322 (2001)