Conservative extensions of the λ-calculus for the computational interpretation of sequent calculus

José Carlos Soares do Espírito Santo

Doctor of Philosophy

Laboratory for Foundations of Computer Science
Division of Informatics
University of Edinburgh

2002

Abstract

This thesis offers a study of the Curry-Howard correspondence for a certain fragment (the *canonical* fragment) of sequent calculus based on an investigation of the relationship between cut elimination in that fragment and normalisation. The output of this study may be summarised in a new assignment Θ, to proofs in the canonical fragment, of terms from certain conservative extensions of the λ-calculus. This assignment, in a sense, is an optimal improvement over the traditional assignment φ, in that it is an isomorphism both in the sense of sound bijection of proofs and isomorphism of normalisation procedures.

First, a systematic definition of calculi of cut-elimination for the canonical fragment is carried out. We study various *right* protocols, *i.e.* cut-elimination procedures which give priority to right permutation. We pay particular attention to the issue of what parts of the procedure are to be implicit, that is, performed by meta-operators in the style of natural deduction. Next, a comprehensive study of the relationship between normalisation and these calculi of cut-elimination is done, producing several new insight of independent interest, particularly concerning a generalisation of Prawitz's mapping of normal natural deduction proofs into sequent calculus.

This study suggests the definition of conservative extensions of natural deduction (and λ-calculus) based on the idea of a built-in distinction between *applicative term* and application, and also between *head* and *tail* application. These extensions offer perfect counterparts to the calculi in the canonical fragment, as established by the mentioned mapping Θ. Conceptual rearrangements in proof-theory deriving from these extensions of natural deduction are discussed.

Finally, we argue that, computationally, both the canonical fragment and natural deduction (in the extended sense introduced here) correspond to extensions of the λ-calculus with applicative terms; and that what distinguishes them is the way applicative terms are structured. In the canonical fragment, the head application of an applicative term is "focused". This, in turn, explains the following observation: some reduction rules of calculi in the canonical fragment may be interpreted as transition rules for abstract call-by-name machines.

Acknowledgements

I am very grateful to my supervisor Samson Abramsky for his guidance and encouragement and also for being patient with my slow progress.

Thanks are also due to Alex Simpson, Ian Mackie and Harold Schellinx, who kindly helped me in several ways.

I was supported by Fundação para a Ciência e Tecnologia, Portugal, grant PRAXIS XXI/BD/11273/97.

Paul Taylor's macros for diagrams and proof trees were used in typesetting this thesis.

My colleagues Cristian, Matias, Dilsun, Bruce, Marco and Sibylle made my stay at LFCS more enjoyable. I will always relate Edinburgh with Joseph and Long Shun.

Some colleagues at Minho must be mentioned: Fernando Miranda for being helpful, Luís Pinto and Jorge Sousa Pinto for their interest in my work.

Some friends made indirect, but important contributions: Hélder, Milucha and Susana.

I thank my parents for their high expectations.

I am indebted to my mother-in-law Maria Alexandra for her immense generosity.

Finally, I am deeply grateful to my wife Luísa for her background support, without which this work would not have been done.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*José Carlos Soares do Espírito Santo*)

Dedicated to André and Pedro

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Curry-Howard correspondence [Curry and Feys, 1958, Howard, 1980], in its simplest form, establishes a connection[1] between natural deduction for the intuitionistic implicational logic and simply typed λ-calculus. When types are seen as formulas, λ-terms may be seen as proofs of their types in natural deduction. When formulas are seen as types, proofs may be seen as programs in a rudimentary programming language, actually the core of functional languages. Moreover, up to this correspondence, normalisation in natural deduction is the same as β-reduction in λ-calculus. The correspondence may, then, be extended to much stronger type theories [Barendregt, 1992], which integrate, according to the perspective, both a proof system for a constructive logic, and a functional language with a sophisticated type system.[2]

This thesis is about extensions of the Curry-Howard correspondence, but in the sense of investigating whether it holds for other kinds of proof systems. Actually, the correspondence was first observed for combinatory logic, which is the type-theoretic counterpart of Hilbert systems [Curry and Feys, 1958]. Here we study the extension of the correspondence to sequent calculus. Therefore, we are interested in the project of finding a programming calculus whose terms may be put in 1-1 correspondence with the proofs of a sequent calculus, in such a way

[1]Whether this connection is an isomorphism or not is highly sensitive to technical formalities like the style of typing (*à la* Curry or *à la* Church) or the management of labels in natural deduction [Hindley, 1997]

[2]The correspondence may also be extended to classical logic [Griffin, 1990, Parigot, 1992].

that each step of cut elimination reads as an execution step in the corresponding program and *vice-versa*. In this thesis we restrict ourselves to intuitionistic implicational logic.

Recently, it has been clearly demonstrated the interest of extending the Curry-Howard correspondence to sequent calculus. We take the following quotation from [Curien and Herbelin, 2000]:

> (...) The correspondence between programs and proofs is traditionally explained through natural deduction (...). We believe that this tradition is in good part misleading. (...) Sequent calculus is far more well-behaved than natural deduction: it enjoys the subformula property, and destruction rules - cuts - are well characterized in contrast with the elimination rules of natural deduction which superimpose both a construction and a destruction operation: the application is a constructor in a term xM, but is destructive in a term $(\lambda x.M)N$.

Let us emphasize that the real challenge in the Curry-Howard correspondence is that the term calculus must be meaningful in programming terms. This is what we mean by a *computational interpretation*. As observed in [Abramsky, 1993],

> What is particularly satisfying about this correspondence in the case of Intuitionistic Logic is that the formalism on the computational side is immediately recognisable as an attractive programming paradigm.

Otherwise, we are left with the void exercise of converting a proof system into the type system for some anonymous term calculus.

Now, the search for the desired programming calculus does not start from square zero. Actually, it starts from the λ-calculus. This is quite natural because there are close links between sequent calculus and natural deduction. Specifically, there is a well-know mapping φ introduced in [Prawitz, 1965] and deeply studied in [Zucker, 1974, Pottinger, 1977] that assigns natural deduction proofs (hence λ-terms) to sequent calculus proofs. Mapping φ interprets axioms as assumptions, right inferences as introductions, every left inference as a certain combination of an application and a substitution, and cuts as substitutions.

Assignment φ is the starting point of this thesis as well. However, this assignment is far from giving an extension of the Curry-Howard correspondence to sequent calculus. The λ-calculus has to be refined and extended in several ways in order to describe cut elimination. In the following, we review previous work attempting to turn φ into a Curry-Howard correspondence. Later on, we explain the contribution of this thesis to the same goal.

1.1 Curry-Howard correspondence and sequent calculus

Just a few years ago, the situation as to the possibility of extending to sequent calculus the Curry-Howard correspondence seemed discouraging.

> From an algorithmic point of view, the sequent calculus has no Curry-Howard isomorphism, because of the multitude of ways of writing the same proof.[Girard et al., 1989]

This remark refers to the possibility of permutation of rules, observed both in classical and intuitionistic sequent calculus [Troelstra and Schwitchtenberg, 2000, Kleene, 1952]. Another manifestation of this is the fact that the traditional mapping φ from intuitionistic sequent calculus to natural deduction is not injective [Zucker, 1974]. As a consequence, the traditional assignment of λ-terms to sequent calculus proofs does not produce a Curry-Howard correspondence [3].

Furthermore, even in a paper where a term calculus with typing rules in the style of sequent calculus is proposed, one may read:

> The reader has probably noticed that our operational semantics is quite different from the cut elimination rules; many of these rules do not seem to have computational significance, at least not in the spirit of current programming practice.[Kesner et al., 1995]

Whether real or apparent, these difficulties did not stop the search for a Curry-Howard correspondence for sequent calculus in the last ten years or so.

Quite naturally, pioneer works attempted a direct interpretation of the fact that in sequent calculus one has left introduction rules. The basic idea was that

[3] For a different opinion, see [Barendregt and Ghilezan, 2000].

right rules *produce* data, whereas left rules *consume* it [Abramsky, 1993]. Hence, left rules seemed to correspond to *pattern matching*, an insight that goes back to [Lafont, 1989]. For instance, using the notation of [Wadler, 1993], the left rule for conjunction looks like

$$\frac{\Gamma, x : A, y; B \vdash t : C}{\Gamma, z : A \wedge B \vdash case\ z = (x, y)\ of\ t : C} \tag{1.1}$$

The constructor $case\ z = (x, y)\ of\ t$ wants to decompose a value z of type $A \wedge B$ into the two components x and y. This matches with the right rule for conjunction, which produces values of the form (u, v), where u has type A and v has type B. The rule that does the actual match is cut

$$\frac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash let\ x\ be\ u\ in\ t : B}\ . \tag{1.2}$$

The intended meaning of these [4] constructors is

$$let\ z\ be\ (u, v)\ in\ (case\ z = (x, y)\ of\ t) \rightarrow t[u/x][v/y]\ . \tag{1.3}$$

Another possibility offered by sequent calculus and fully exploited since the early days is the fact that inference rules may act in any formula of the sequent, unlike natural deduction (in sequent style), in which only the RHS formula is transformed. This suggests a system of term assignment in which not only the RHS formula, but instead *any* formula of the sequent is assigned a term, recording, so to say, its history. Let us call this kind of system *asynchronous*. Indeed, there is no term recording the global history of the derivation. For instance, instead of (1.1), one has

[4] We started with conjunction instead of implication, which is the connective we are interested in, because the *case* constructor for the left introduction of implication has a very unusual form

$$\frac{\Gamma \vdash u : A \quad \Gamma, y : B \vdash v : C}{\Gamma, z : A \supset B \vdash case\ z = \lambda u.y\ of\ v : C}\ .$$

The notation $\lambda u.y$ is due to [Lafont, 1989]. The idea is to match $\lambda x.t$ with $\lambda u.y$, similarly as we match (u, v) with (x, y). The reduction rule is

$$let\ z\ be\ \lambda x.t\ in\ (case\ z = \lambda u.y\ of\ v) \rightarrow v[t[u/x]/y]\ .$$

$$\frac{\Gamma, p : A, q; B \vdash t : C}{\Gamma, (p, q) : A \wedge B \vdash t : C} \tag{1.4}$$

Here, Γ contains declarations of patterns $p : A$. Observe how the term t remains unchanged. The first system fully developed along these lines seems to have been a term assignment system for classical linear logic in [Abramsky, 1993].

Asynchronous term assignment systems have the potential of modelling *nested* patterns, like $(x, (y, z))$. This line of research was pursued in [Kesner et al., 1995] and [Cerrito and Kesner, 1999]. Another characteristic is that these systems are highly insensitive to permutation of rules. To see this, suppose Γ in (1.4) is of the form $\Gamma_0, p' : A, y' : B'$ and suppose we want to construct both $A \wedge B$ and $A' \wedge B'$. Independently of the order by which the two instances of the left rule occur, the final sequent will be

$$\Gamma_0, (p', q') : A' \wedge B', (p, q) : A \wedge B \vdash t : C$$

There is no global record telling which conjunction was built first.[5] Let us give another example. Suppose in (1.4) t is of the form $\lambda x.t_0$ and suppose we build a cut with cutformula C, whose left subderivation is a derivation ending with (1.4). The term annotating this cut is of the form

$$let\ z\ be\ \lambda x.t_0\ in\ u\ . \tag{1.5}$$

In this annotation we have direct access to the last time a right rule was applied in the left subderivation. In a synchronous system, the access to the last right rule is gained by explicitly permuting the cut to the left.

Asynchronous term assignment system are a very interesting approach to the problems caused by permutability of rules in sequent calculus. Nevertheless, it is an approach we do not follow here. This is so because, in this thesis, we avoid the permutability problem in a different way, by studying a permutation-free fragment of the sequent calculus, as explained below.

[5] This is why the mentioned term calculus for classical linear logic in [Abramsky, 1993] is proposed as an alternative to proof-nets [Girard, 1987].

Once we decided not to follow the path of, say, [Kesner et al., 1995], the analysis of sequent calculus so far leaves us with a system of the kind of [Wadler, 1993], where left rules are interpreted as pattern matching constructors. We are back to (1.1) and (1.2). Now, some obscure points remain in this interpretation. What does it mean the left permutation of cuts, a necessary feature in this setting as explained above when discussing (1.5)? Another example is the mismatch between the intended meaning of pattern matching constructors (1.3) and what happens in the key step of cut elimination

$$let\ z\ be\ (u,v)\ in\ (case\ z=(x,y)\ of\ t) \to let\ y\ be\ v\ in\ (let\ x\ be\ u\ in\ t)\ \ , \tag{1.6}$$

where the LHS cut, with cut formula $A \wedge B$, say, is replaced by two cuts with cut formulas A and B, respectively. How do the two *let*'s in the RHS of (1.6) relate to $t[u/x][v/y]$ in the RHS of (1.3)? Take, for instance, $let\ x\ be\ u\ in\ t$. In terms of cut elimination, what we want is to permute to the right the cut represented by this *let*. In other words, we want to permute u inside t and, somehow, this is to be related to $t[u/x]$. While people were thinking about this, a new metaphor, a new conceptual tool appeared - that of *explicit substitution* [Abadi et al., 1991] - that provided the right language in which to describe the portion of the cut elimination process we are analysing. The right permutation of $let\ x\ be\ u\ in\ t$ eventually performs $t[u/x]$, but in a stepwise fashion. While the cut in the LHS of (1.6) matches the pair (u,v) with the pattern (x,y), the cuts in the RHS of (1.6) are explicit substitutions and (1.6) should be rewritten as

$$let\ z\ be\ (u,v)\ in\ (case\ z=(x,y)\ of\ t) \to t\langle x:=u\rangle\langle y:=v\rangle\ \ . \tag{1.7}$$

This is the first piece of evidence that cuts bear different interpretations according to the stage of cut elimination they are going through.[6]

Several refinements of the traditional assignment φ of λ-terms to sequent calculus proofs are suggested. Instead of interpreting cut as ("meta"-)substitution, one should interpret it as *explicit* substitution:

[6] The impact of this observation is clear in the evolution of the system of [Kesner et al., 1995] to that of [Cerrito and Kesner, 1999].

$$\frac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash t\langle x := u\rangle : B} \ .$$

But this is so only for newly-born cuts, which are willing to be permuted to the right, like those in the RHS of (1.7). Key cuts, like the cut in the LHS of (1.7), should, for the moment, be annotated with a matching constructor, as in (1.2).

As firstly observed by [Gallier, 1993], explicit substitutions allow, then, to express in the term calculus (some of the) stepwise cut elimination rules (see also [Vestergaard and Wells, 1999]). This idea was fully realized for the first time in [Herbelin, 1995]. Furthermore, as suggested in [Barendregt and Ghilezan, 2000], explicit substitutions improve the situation as to the permutability problem, if one refines the traditional term assignment of left rules. For instance, the left rule for implication becomes

$$\frac{\Gamma \vdash u : A \quad \Gamma, y : B \vdash t : C}{\Gamma, x : A \supset B \vdash t\langle y := xu\rangle : C} \ .$$

Explicit substitution shows up in the place where meta-substitution appeared according to the old assignment. Let us see an example, taken from *op. cit.* (we will be negligent about contexts). By permuting the two rules of

$$\frac{..., z : A \vdash z : A \qquad \dfrac{..., x : C, w : B \vdash w : B}{..., w : B \vdash (\lambda x.w) : C \supset B}\,Right}{..., z : A, y : A \supset B \vdash (\lambda x.w)\langle w := yz\rangle : C \supset B}\,Left \tag{1.8}$$

one obtains

$$\frac{\dfrac{..., z : A \vdash z : A \qquad ..., x : C, w : B \vdash w : B}{..., x : C, z : A, y : A \supset B \vdash w\langle w := yz\rangle : B}\,Left}{..., z : A, y : A \supset B \vdash \lambda x.(w\langle w := yz\rangle) : C \supset B}\,Right$$

In the traditional term assignment, these two proofs would get the term $\lambda x.yz$. With the new assignment, the distinction between these two proofs is reflected in the distinction between $(\lambda x.w)\langle w := yz\rangle$ and $\lambda x.(w\langle w := yz\rangle)$.

Unfortunately, explicit substitutions are not a panacea. Let us consider the following situation (we focus on implication from now on):

$$\dfrac{\dfrac{\begin{matrix}d_1\\ \vdots\\ ...\vdash u : A\end{matrix} \qquad \begin{matrix}d_2\\ \vdots\\ ..., x : B \vdash t : C\end{matrix}}{..., z : A \supset B \vdash t\langle x := zu\rangle : C}\,Left \qquad \begin{matrix}d_3\\ \vdots\\ ..., y : C \vdash t' : D\end{matrix}}{..., z : A \supset B \vdash t'\langle y := t\langle x := zu\rangle\rangle : D}\,Cut \tag{1.9}$$

Assume that the last inference of d_3 is a left rule that introduced the displayed C without (implicit) contraction. This cut is, thus, right-permuted, *i.e.* cannot be permuted to the right any further. On the other hand, it is left-permutable. The result of permuting it over the displayed left inference is

$$\dfrac{\begin{matrix}d_1\\ \vdots\\ ...\vdash u : A\end{matrix} \qquad \dfrac{\begin{matrix}d_2\\ \vdots\\ ..., x : B \vdash t : C\end{matrix} \qquad \begin{matrix}d_3\\ \vdots\\ ..., y : C \vdash t' : D\end{matrix}}{..., x : B \vdash t'\langle y := t\rangle : D}\,Cut}{..., z : A \supset B \vdash t'\langle y := t\rangle\langle x := zu\rangle : D}\,Left \tag{1.10}$$

With the present assignment of terms, this permutation reads

$$t'\langle y := t\langle x := zu\rangle\rangle \to t'\langle y := t\rangle\langle x := zu\rangle \enspace .$$

This is an unnatural and unusual rule. It seems that the explicit substitution metaphor, appropriate as it is for describing the right permutation of cuts, is not appropriate anymore for the left permutation. So we are, again, in need of a new idea.

A radical new idea may be found in [Herbelin, 1995]. The problem of permutability is completely avoided by interpreting, not the whole sequent calculus, but a permutation-free fragment of it. The point is that nothing is lost in this fragment, as it proves the same sequents as full LJ. At the same time, cut-free proofs in this fragment are in 1-1 correspondence with normal natural deduction proofs. Actually, the fragment has many "structural" advantages, as emphasized in [Dyckhoff and Pinto, 1998]: for instance, head variables are brought to the surface of an applicative term $xN_1...N_k$, let alone the fact that it keeps being a sequent calculus, therefore keeping the subformula property.

In fact, this fragment, which we call the *canonical* fragment (or the fragment of *canonical* proofs), was rediscovered several times [Danos et al., 1997, Dyckhoff and Pinto, 1999, Mints, 1996].[7] In a cut-free setting, both [Mints, 1996] and [Dyckhoff and Pinto, 1999] called the proofs in the fragment "normal" and showed that they are the proofs irreducible w.r.t. a set of permutation rules. Since we will not restrict ourselves to a cut-free setting, we cannot adopt the "normal" terminology. [Danos et al., 1997] shows that the fragment is closed for the "tq-protocol" - or rather the t-protocol, as the fragment we have in mind requires, in the terminology of *op. cit.*, all formulas to be t-coloured. We will come back to the relation between the t-protocol and Herbelin's cut elimination procedure.

The canonical fragment will be explained in detail in Chapter 2. However, we show briefly here how the interpretation of reduction step (1.9) $\rightarrow$ (1.10) improves.

The restriction defining the canonical fragment is such that in (1.9) and (1.10) $B = B_1 \supset ... \supset B_k \supset C$ and $C = C_1 \supset ... \supset C_m \supset D$, for some k, m. Moreover, derivation d_2 (resp. d_3) consists of the stack of k (resp. m) left inferences, starting from axiom $C \vdash C$ (resp. $D \vdash D$), that builds the $B = B_1 \supset ... \supset B_k \supset C$ (resp. $C = C_1 \supset ... \supset C_m \supset D$) displayed in its end-sequent, and is annotated with a *list* of terms $l = [v_1, ..., v_k]$ (resp. $l' = [v'_1, ..., v'_m]$), where v_i (resp. v'_i) annotates the left subderivation of the i-th left inference in the stack (from bottom to top). The annotation for each left inference in those stacks is consing (notation ::) and for the axioms $C \vdash C$ and $D \vdash D$ is the empty list (notation []). Since the formulas

[7] [Troelstra and Schwitchtenberg, 2000] attributes the identification of the fragment to Curry, on the basis of a passage of [Howard, 1980]. See Chapter 2. As to [Danos et al., 1997], we are thinking of the intuitionistic, t-coloured restriction of LK^η.

$$\begin{array}{rcr} C & & D \\ B_k \supset C & & C_m \supset D \\ B_{k-1} \supset B_k \supset C & \text{and} & C_{m-1} \supset C_m \supset D \\ ... & & ... \\ B = B_1 \supset ... \supset B_k \supset C & & C = C_1 \supset ... \supset C_m \supset D \end{array}$$

successively introduced in those stacks are linear and main (in the usual sense of sequent calculus), they do not get a variable. The new annotations for (1.9) are

$$\dfrac{\dfrac{\begin{matrix} d_1 \\ \vdots \\ ... \vdash u : A \end{matrix} \qquad \begin{matrix} d_2 \\ \vdots \\ ..., B \vdash l : C \end{matrix}}{..., z : A \supset B \vdash z(u :: l) : C}\ Left \qquad \begin{matrix} d_3 \\ \vdots \\ ..., C \vdash l' : D \end{matrix}}{..., z : A \supset B \vdash (z(u :: l))l' : D}\ Cut \tag{1.11}$$

The displayed left inference is assigned $z(u :: l)$. This is not simply consing because $A \supset B$ is not necessarily linear. An informal reading of $z(u :: l)$ is that z is "applied" to u, with l providing further arguments. Let $t = z(u :: l)$. Then, the displayed cut is annotated with tl'. Again, think of this as an application, where l' provides m arguments.

As to (1.10), the new assignment is

$$\dfrac{\begin{matrix} d_1 \\ \vdots \\ ... \vdash u : A \end{matrix} \qquad \dfrac{\begin{matrix} d_2 \\ \vdots \\ ..., B \vdash l : C \end{matrix} \qquad \begin{matrix} d_3 \\ \vdots \\ ..., C \vdash l' : D \end{matrix}}{..., x : B \vdash ll' : D}\ Cut}{..., z : A \supset B \vdash z(u :: (ll')) : D}\ Left \tag{1.12}$$

The displayed cut is of a new kind, between the two list l and l'. This is notated ll' and should be understood as an explicit append, or "concatenation". The left permutation (1.11) $\to$ (1.12) now reads

$$(z(u :: l))l' \to z(u :: (ll')) \ . \tag{1.13}$$

Cut elimination will keep permuting d_3 over d_2 and, in the term calculus, this corresponds to a stepwise performance of the append of l with l'. The final result of this permutation is a stack of $k+m$ left inferences, starting from axiom $D \vdash D$ and generating $B = B_1 \supset ... \supset B_k \supset C_1 \supset ... \supset C_m \supset D$.

The remarkable result of Herbelin is that the Curry-Howard counterpart to his sequent calculus is a version of the λ-calculus with explicit substitutions, but in which applicative terms no longer have the form $(...(tu_1)...u_k)$ but instead have the form $t[u_1, ..., u_k]$. These two ways of representing applicative terms are regarded by Herbelin as explaining the difference between a sequent calculus structure and a natural deduction structure. As to cut elimination, Herbelin's system is the first with several kinds of cuts bearing different computational interpretations. In his own words

> Each elementary step of cut-elimination exactly matches with a β-reduction step, a substitution propagation step or a concatenation computation step.

Substitution propagation corresponds to right permutation and "concatenation" of lists is related to the left permutation. As to the key-step of cut elimination, observe that a key-cut is a right-permuted cut whose left subderivation ends with a right rule. It has the form

$$\dfrac{\dfrac{\begin{matrix} d \\ \vdots \\ ..., x : A \vdash t : B \end{matrix}}{... \vdash \lambda x.t : B}\,Right \qquad \dfrac{\begin{matrix} d' \\ \vdots \\ ... \vdash u : A \end{matrix} \qquad \begin{matrix} d'' \\ \vdots \\ ..., B \vdash l : C \end{matrix}}{..., A \supset B \vdash u :: l : C}\,Left}{... \vdash (\lambda x.t)(u :: l) : C}\,Cut$$

Hence, a key-cut is a kind of β-redex. The key-step of cut elimination produces

$$\dfrac{\dfrac{\begin{matrix} d' \\ \vdots \\ ... \vdash u : A \end{matrix} \qquad \begin{matrix} d \\ \vdots \\ ..., x : A \vdash t : B \end{matrix}}{... \vdash t\langle x := u\rangle : B}\,Cut \qquad \begin{matrix} d'' \\ \vdots \\ ..., B \vdash l : C \end{matrix}}{... \vdash t\langle x := u\rangle l : C}\,Cut$$

Instead of a "β-reduction step", this is a *beta*-reduction step, in the terminology of explicit substitution calculi [Abadi et al., 1991], as it is the step that generates an explicit substitution.

The weak point of this interpretation is the meaning given to (1.13). Is it a "concatenation computation step"? A better interpretation is obtained by a combination of two observation. First, the effect of (1.13) is to bring the head variable z to the surface, so to speak. Second, an interpretation in terms of "lists", "concatenation" and so on seems too literal. Already in [Herbelin, 1995] this was recognised, as the idea of "applicative context" is briefly mentioned in connection with the idea of lists. An applicative context (other names: call-by-name evaluation context, or continuation[8]) is an expression of the form $(...([-]u_1)...u_k)$, where $[-]$ represents a "hole". When a term t is "filled" in the hole, an applicative term $(...(tu_1)...u_k)$ results.

In [Curien and Herbelin, 2000], the interpretation of lists as evaluation contexts was fully developed. Cuts tl are interpreted as t filled in the hole of l. Consing $u :: l$ means filling the hole of l with $[-]u$. Although cuts of the form ll' were not considered in *op. cit.*, it is clear that they correspond to the composition of contexts, in an obvious sense. As sketched in *op. cit.*, it results that reduction rules close to (1.13), namely

$$(t[u])l \to t(u :: l) \ ,$$

model certain transition rules of environment machines like Krivine's machine [Krivine]. Here, lists receive yet another interpretation, as stacks in the abstract machines terminology.

The search for the computational interpretation of sequent calculus reached its highest point in [Curien and Herbelin, 2000]. In this paper it is argued that there is a Curry-Howard match between the symmetry of classical logic, as expressed in the sequent calculus LK, and implicit symmetries of programming languages like program/context and call-by-name/call-by-value.

[8]The idea of evaluation contexts may be traced back to [Felleisen et al., 1986].

1.2 Contribution of this thesis

The story we have just told may be seen as a long struggle for improving the traditional assignment φ of λ-terms to sequent calculus. Manifestly, λ-calculus is too poor a language to express what is going on in cut elimination.

In accordance with the main requirement for obtaining a Curry-Howard interpretation, several authors proposed to extend the λ-calculus with features which were meaningful from a programming point of view, like pattern matching, or were theoretical tools introduced for reasoning about programs, like explicit substitutions or evaluation contexts. We say that these extensions are of a *computational* nature.

In this thesis we propose a new assignment Θ, to the *canonical* fragment of sequent calculus, of terms from certain conservative extensions of the λ-calculus. The main property of Θ is to be an isomorphism, both in the sense of sound bijection of proofs, and in the sense of isomorphism of normalisation procedures.

The extension of λ-calculus proposed is based on the idea of a built-in distinction between *applicative terms* and applications, and also between *head* and *tail* applications.

This extension is *proof-theoretical* in nature, for three reasons. First, it does not leave the framework of natural deduction - actually it represents an extension of it. Second, it is motivated by an analysis of a mapping from natural deduction to sequent calculus introduced by Prawitz. Third, the issue of *explicitness* (in particular, the issue of explicit substitutions), already present in the computational interpretations mentioned above, will be taken seriously here from the point of view of normalisation procedures (both for sequent calculus and natural deduction), their relationship and interpretation.

As to the computational interpretation of sequent calculus, we have seen how the identification of the canonical fragment of sequent calculus allowed a *modular* approach, by which the permutability problem is abstracted away, and a smaller system - precisely that closer to natural deduction - is studied first. Here we do the inverse, so to speak. By extending natural deduction so as to obtain a system isomorphic (but far from equal) to the canonical fragment, we will be able to *separate*, in the interpretation of the latter, among the features added to λ-calculus, *the* feature that characterises computationally the canonical fragment, from the features that were added because λ-calculus is a weak system of natural deduction.

It turns out that the computational interpretation of both the canonical fragment and natural deduction (in the extended sense introduced here) is certain extensions of the λ-calculus with applicative terms. Moreover, what distinguishes computationally the canonical fragment from natural deduction is the way applicative terms are structured. In the natural deduction side, applicative terms are built out of head and tail eliminations, and the head application is deeply buried. In the canonical fragment, applicative terms are built out of an evaluation context and the head application, and the latter is "focused", *i.e.* immediately available.

As to proof theory, we think that, after the identification of the canonical fragment, there was no systematic study of cut-elimination in this fragment and its relation with normalisation. We regard as a contribution of this thesis the systematic definition of calculi of cut-elimination for the canonical fragment, as well as the comprehensive study of the relationship with natural deduction that follows. Moreover, we show how the idea of a built-in distinction between applicative term and application, simple as it is, causes a vast rearrangement of the relationship between sequent calculus, natural deduction and λ-calculus.

Overview of the thesis

In Chapter 2 we fix notations definitions and terminology as to sequent calculus and cut elimination. It also provides the proof-theoretical background for the following chapters.

In Chapter 3, calculi of cut elimination for the canonical fragment are systematically developed.

In Chapters 4 and 5 we produce the study of the relationship between cut elimination in the canonical fragment and normalisation.

In Chapter 6, extensions of natural deduction are defined which provide perfect counterparts to the calculi defined in the canonical fragment. The mapping Θ mediates between these two kinds of calculi.

In Chapter 7, applications to the computational interpretation of sequent calculus are discussed.

In Chapter 8, we summarise the contributions of this thesis and propose future work.

Notations and terminology

Types: We just treat intuitionistic implicational logic. Formulas (or types) are given by

$$A, B, C, D ::= p \,|\, A \supset B$$

where p ranges over propositional letters. As usual, we assume that implication is bracketed to the right. *E.g.* $A \supset B \supset C = A \supset (B \supset C)$.

Contexts: A *context* is a *consistent* set of *declarations* $x : A$. By consistent we mean that if $x : A$ and $x : B$ are in a context, then $A = B$. Contexts are ranged over by Γ. We write $x \in \Gamma$ meaning $x : A \in \Gamma$ for some A. $\Gamma, x : A$ denotes the *consistent union* $\Gamma \cup \{x : A\}$, which means that, if x is already declared in Γ, then it is declared with type A.

Rewriting: If R is a binary relation (sometimes called a notion of reduction) on a set of terms, then $\to_R$ denotes its compatible closure and $\to^+, \to^*$ the usual closures of $\to_R$. We will never deal with *conversion*. Therefore, $=$ will always mean equality. If $\to_R$ is confluent, $\downarrow_R$ denotes the associated normal-form mapping. Usually we write R_1, R_2 instead of $R_1 \cup R_2$.

λ-calculus: See Table 1.1. We only work with pure terms. As usual, we assume that application is bracketed to the left. *E.g.* $MN_1N_2 = (MN_1)N_2$.

Typing is *à la* Curry. See Table 1.2. Sequent(s) above the deduction line of a typing rule are the *premiss(es)* of the rule and the sequent below the deduction line is called its *conclusion*. The order of premisses in rules matters, and is as in Table 1.2, so that we can refer without ambiguity to the left premiss or the right

Table 1.1: The λ-calculus

$(Terms)$	$M, N \;::=\; x \mid \lambda x.M \mid MN$
(β)	$(\lambda x.M)N \;\rightarrow\; M[N/x]$

where

$$\begin{array}{rcl} x[N/x] & = & N \\ y[N/x] & = & y,\ y \neq x \\ (\lambda y.M)[N/x] & = & \lambda y.M[N/x] \\ (MM')[N/x] & = & M[N/x]M'[N/x] \end{array}$$

subderivation. The left premiss of $Elim$ is also called the *main* premiss. We may refer to $Elim$ (resp. $Intro$) as the elimination (resp. introduction) rule.

A *value* is a variable or a λ-abstraction [Plotkin, 1975]. If a λ-term is not a value, it is an *applicative term*. Given an application MN, we say the application is a *value* application if M is a value.

In [Joachimski and Matthes] the syntax of the λ-calculus is given as follows:

$$M, N ::= x \mid \lambda x.M \mid xN\vec{\mathbf{N}} \mid (\lambda x.M)N\vec{\mathbf{N}}$$

Here $\vec{\mathbf{N}}$ ranges over (possibly empty) "vectors" of λ-terms. This is an informal device for bringing head variables and redexes to the "surface" of applicative terms. In this thesis we will find formal ways of achieving the same effect.

For future reference, we give here the following definition.

Definition 1 (Compatible closure) *Given a binary relation R on λ-terms, the compatible closure $\rightarrow_R$ is the least binary relation $\rightarrow$ on λ-terms containing*

Table 1.2: Typing rules for λ

$$Var\,\frac{}{\Gamma, x : A \vdash x : A} \quad Intro\,\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B} x \notin \Gamma$$

$$Elim\,\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : B}{\Gamma \vdash MN : B}$$

R closed under:

$$Intro\,\frac{M \to M'}{\lambda x.M \to \lambda x.M'}$$

$$Elim1\,\frac{M \to M'}{MN \to M'N} \quad Elim2\,\frac{N \to N'}{MN \to MN'}$$

Renaming of bound variables: α-equivalent terms are seen as equal. Renaming of bound variables is assumed whenever appropriate. In particular, we may assume that, in an expression, the sets of free and of bound variables are disjoint. This is Barendregt's variable convention [Barendregt, 1984], which applies to all calculi in this paper.

Grammars: We present syntax as in the top part of Table 1.1. $Terms$ is the syntactic class and M, N are meta-variables ranging over $Terms$.

Lists: Empty list and cons are written [] and ::, respectively. $[u_1, u_2, ..., u_k]$ abbreviates $u_1 :: (u_2 :: ... :: (u_k :: [])...)$. In particular, $[u]$ is $u :: []$.

Naming of λ-calculi: Often we refer to the λ-calculus simply as λ, and similarly for other calculi. In naming λ-calculi, we follow some conventions. (1) $\overline{\lambda}$ means that the calculus is close to Herbelin's $\overline{\lambda}$-calculus [Herbelin, 1995]. (2) $\mathcal{G}$ is after Gentzen. (3) $\mathcal{P}$ is after Prawitz. (4) $\mathcal{N}$ means that the calculus is a natural deduction system. (5) h signals a reduction rule for simplifying $h$ead applications. (6) $\mathtt{x}$ signals explicit substitutions.

Relationship between calculi: We will find several times the following situation. (1) The terms of a calculus λ_1 are also terms of another calculus λ_2. (2) If $t \to u$ in λ_1 then $t \to^+ u$ in λ_2. (3) There is a mapping $p : \lambda_2 \to \lambda_1$ such that (i) $pt = t$, for all t in λ_1 and (ii) $t \to u$ in λ_2 implies $pt \to^* pu$ in λ_1. Such mapping will be called a *projection*. Then, we say that λ_2 is a *conservative extension* of λ_1, because it holds that

$$t \to^* u \text{ in } \lambda_1 \text{ iff } t \to^* u \text{ in } \lambda_2, \text{ for all } t, u \text{ in } \lambda_1.$$

"Only if" follows from (2). As to "if", suppose $t \to^* u$ in λ_2, with t, u in λ_1. Then, by (3-ii), $pt \to^* pu$ in λ_1. But $pt = t$ and $pu = u$, by (3-i).

Moreover, quite often p is such that $t \to^* pt$ in λ_2. In that case, we say that λ_2 is *internally* conservative over λ_1.

Simultaneous induction: Consider the following example:

$$\begin{array}{rccl} NormalTerms & N & ::= & x \,|\, (\lambda x.N) \,|\, app(A) \\ NormalApplications & A & ::= & (xN) \,|\, (AN) \end{array}$$

Let P be a property over the elements of $NormalTerms$ and Q a property over the elements of $NormalApplications$. Suppose we want to prove

$$\begin{array}{c} \text{for all } N,\ P(N) \\ \text{and} \\ \text{for all } A,\ Q(A)\ . \end{array} \tag{1.14}$$

It suffices to prove

$$\frac{}{P(x)} \quad \frac{P(N)}{P(\lambda x.N)} \quad \frac{Q(A)}{P(app(A))}$$

$$\frac{P(N)}{Q(xN)} \quad \frac{Q(A) \quad P(N)}{Q(AN)}$$

A proof of these five implications is what we call a proof of (1.14) by *simultaneous induction* on N and A. We will refer to induction hypotheses $P(N)$ and $Q(A)$ as IH1 and IH2 respectively.

Sequent calculus: See Chapter 2.

Abbreviations: We will use the following: RHS (= right hand side), LHS (= left hand side), iff (= if and only if), IH (= induction hypothesis), whnf (= weak head normal form).

Chapter 2

Background

In this chapter we introduce some preliminary material. First, we define the family of cut-elimination procedures we adopt in this thesis and give one example in detail - the so-called t-protocol [Danos et al., 1997]. Second, we explain the canonical fragment of sequent calculus. We adopt the approach of [Herbelin, 1995] as to the way the syntactic machinery is set up.

2.1 A cut-elimination procedure

Consider the sequent calculus defined in Table 2.1. Sequents have the form

$$\Gamma \vdash L : A \ , \tag{2.1}$$

where Γ is a consistent set of declarations $x : B$ and L is a term in a certain language defined by

$$L ::= \mathsf{Ax}(x) \,|\, \mathsf{Cut}(L, (y)L) \,|\, \mathsf{L}(x, L, (y)L) \,|\, \mathsf{R}((x)L)$$

Rules in Table 2.1 have a natural reading as typing rules for this language. However they have another reading, as natural as this, as rules for generating logical derivations. This is clear if one understands a sequent (2.1) occurring in a derivation as the usual sequent

Table 2.1: Sequent calculus inference rules

$$Ax \frac{}{\Gamma, x : A \vdash \mathsf{Ax}(x) : A}$$

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

$$Right \frac{\Gamma, x : A \vdash L : B}{\Gamma \vdash \mathsf{R}((x)L) : A \supset B} x \notin \Gamma$$

$$Cut \frac{\Gamma \vdash L_1 : A \quad \Gamma, y : A \vdash L_2 : C}{\Gamma \vdash \mathsf{Cut}(L_1, (y)L_2) : C} y \notin \Gamma$$

$$\Gamma \vdash A \ ,$$

plus some information about the derivation above the sequent, contained in the term L, and which is made locally available. This is the same phenomenon as natural deduction "in sequent style", in which, for each formula occurrence in a derivation, the information about undischarged assumptions (that is, the undischarged leaves of the derivation above the formula occurrence) is made locally available in the form of a context.

In the case of (2.1), L keeps record of the sequence of rules by which the derivation above the sequent was built, together with extra information contained in the bound and free variables of L. This information refers, respectively, to which formulas in the contexts were active in, or were introduced by, the application of a rule.

The record L is an incomplete one in the sense that we cannot reconstruct, from $\Gamma \vdash A$ plus L the whole derivation above the sequent. Actually, there may be many derivations of $\Gamma \vdash L : A$. This situation is due to the fact that L is untyped, and is typical of Curry-style typing [Hindley, 1997]. Nevertheless, we stick to this logical understanding of sequents (2.1). The advantage is that we may, for instance, define cut elimination directly in typing derivations. We do not have two entities - the typing system and the underlying logical calculus - but a single one, and we will never talk about term erasure. This approach will apply to any calculus in this thesis.

We now introduce some terminology. In a sequent, we often refer to the part to the left (resp. right) of $\vdash$ as the *LHS* (resp. *RHS*) of the sequent. In a sequent calculus rule, the sequent(s) above the deduction line is (are) called the *premiss(es)* of the rule and the sequent below the line is called the *conclusion* of the rule. By an *inference* we mean an occurrence of a rule. An occurrence of Ax, $Left$, *Right* and Cut may be referred to as an axiom, a left inference, a right inference and a cut, respectively.

In each sequent calculus rule, some formulas play a distinguished role. Consider again Table 2.1. The occurrences of A and B in rules $Left$, *Right* and Cut are said to be *active*. In the case of Cut, active formulas are also called the *cut formulas*. Moreover, the *right* (resp. the *left*) cut formula of an occurrence of cut is the cut formula in the end-sequent of the right (resp. left) subderivation. For this terminology to make sense, order of premisses in rules matters and is fixed as in Table 2.1. Observe that the right (resp. left) cut formula occurs in the LHS (resp. RHS) of a sequent. The occurrences of A in Ax and of $A \supset B$ in $Left$ and *Right* are said to be *main*. Cut has no main formula, Ax has two, the *left-main* and the *right-main*. We distinguish between the main formulas of $Left$ and *Right*, on the one hand, and the main formulas of Ax. The former are said to be *logical*. If a formula in a rule is neither active not main, it is said to be *passive*.

Often, passive formulas are not as passive as they seem. Passive formulas in rule Ax are called *weakened*. Passive formulas in the conclusion of $Left$ and Cut are said to be *contracted*. The passive formulas that are simply passive are those of *Right* and the formula C in $Left$ and Cut. Even a main formula may be contracted. This is when, in the rule $Left$, x is already in Γ. Finally, a formula is *linear* if it is neither weakened nor contracted.

The variable displayed at the right premiss of the Cut rule is called the *cut* *variable*.

We now want to define the *complete right permutation* of a cut. This is the expected process, consisting in performing the following instructions as long as possible, to the initial cut and its descendants. The cut is permuted upwards through the right subderivation past *Right* inferences, other cuts and even $Left$ inferences when the main formula of the latter is not the right cut formula. This process causes duplication (resp. erasing) of the cut whenever the right cut formula is contracted (resp. weakened). If the right cut formula is main in Ax, the cut is replaced by its left subderivation. If the right cut formula is main in $Left$, the permutation of the cut stops (with two new copies of the cut continuing their own permutations, if the main formula of the $Left$ inference is contracted).

The *complete left permutation* of a cut is simpler. Permute the cut upwards through the left subderivation as long as the last inference of the latter is $Left$ or *Cut*. When the last inference of the left subderivation is Ax, replace the cut with its right subderivation; when it is *Right*, do nothing and stop the process.

A cut is *right permuted* if its right cut formula is main in $Left$ and linear; otherwise, the cut is *right permutable*. A cut is *left permuted* if its left cut formula is main in *Right*; otherwise, the cut is *left permutable*. A *key-cut* is a cut that is both right and left permuted.

By a *cut-elimination procedure* we mean a (possibly non-deterministic) set of rules describing which transformations are to be applied to *an arbitrary instance of cut*. In this sense, a cut-elimination procedure does not determine which instance of cut is to be reduced next.

The *t-protocol* [Danos et al., 1997] is the cut elimination procedure consisting of the repeated application of the following instruction, regarded as a single step of reduction, to an arbitrary instance of the cut rule. Given a cut, it is either right permutable or right permuted. In the first case, perform its *complete* right permutation. In the second case, the cut is either left permutable or left permuted. In the former case, perform its *complete* left permutation. In the latter case the cut is a key cut. In this case, both cut formulas are logical, hence apply the key step of cut elimination.

The t-protocol is an example of a right protocol. A *right* protocol is a cut-elimination procedure such that, when reducing a cut that is simultaneously right and left-permutable, gives priority to the right permutation.

Given a cut

$$\frac{\begin{array}{c} d_1 \\ \vdots \\ \Gamma \vdash L_1 : A \end{array} \qquad \begin{array}{c} d_2 \\ \vdots \\ \Gamma, x : A \vdash L_2 : B \end{array}}{\Gamma \vdash \mathsf{Cut}(L_1, (x)L_2) : B}$$

its complete right permutation generates right permuted cuts. These have the form

$$\frac{\begin{array}{c} d_1 \\ \vdots \\ \Gamma \vdash L_1 : A_1 \supset A_2 \end{array} \qquad \dfrac{\begin{array}{c} d_{21} \\ \vdots \\ \Gamma \vdash L_{21} : A_1 \end{array} \qquad \begin{array}{c} d_{22} \\ \vdots \\ \Gamma, y : A_2 \vdash L_{22} : B \end{array}}{\Gamma, x : A_1 \supset A_2 \vdash \mathsf{L}(x, L_{21}, (y)L_{22}) : B}\,Left}{\Gamma \vdash \mathsf{Cut}(L_1, (x)\mathsf{L}(x, L_{21}, (y)L_{22})) : B}$$

where x is linear. We may compact this in a single construction $\mathsf{Cut}(L_1, L_{21}, (y)L_{22})$ with typing rule

$$\frac{\begin{array}{c} d_1 \\ \vdots \\ \Gamma \vdash L_1 : A_1 \supset A_2 \end{array} \qquad \begin{array}{c} d_{21} \\ \vdots \\ \Gamma \vdash L_{21} : A_1 \end{array} \qquad \begin{array}{c} d_{22} \\ \vdots \\ \Gamma, y : A_2 \vdash L_{22} : B \end{array}}{\Gamma \vdash \mathsf{Cut}(L_1, L_{21}, (y)L_{22}) : B}$$

Observe that variable x disappears. Indeed, the variable is irrelevant when a formula is linear and becomes active immediately after being introduced. This inference rule is called generalised application in [Negri and von Plato, 2001]. The complete left permutation of each of these cuts generates, at most, one cut of the form

$$\frac{\dfrac{\begin{array}{c} d_{11} \\ \vdots \\ \Gamma, z : A_1 \vdash L_{11} : A_2 \end{array}}{\Gamma \vdash \mathsf{R}((z)L_{11}) : A_1 \supset A_2}\,Right \qquad \dfrac{\begin{array}{c} d_{21} \\ \vdots \\ \Gamma \vdash L_{21} : A_1 \end{array} \qquad \begin{array}{c} d_{22} \\ \vdots \\ \Gamma, y : A_2 \vdash L_{22} : B \end{array}}{\Gamma, x : A_1 \supset A_2 \vdash \mathsf{L}(x, L_{21}, (y)L_{22}) : B}\,Left}{\Gamma \vdash \mathsf{Cut}(\mathsf{R}((z)L_{11}), (x)\mathsf{L}(x, L_{21}, (y)L_{22})) : B}$$

or, in compact form,

$$\dfrac{\begin{array}{c} d_{11} \\ \vdots \\ \Gamma, z : A_1 \vdash L_{11} : A_2 \end{array} \qquad \begin{array}{c} d_{21} \\ \vdots \\ \Gamma \vdash L_{21} : A_1 \end{array} \qquad \begin{array}{c} d_{22} \\ \vdots \\ \Gamma, y : A_2 \vdash L_{22} : B \end{array}}{\Gamma \vdash \mathsf{Cut}((z)L_{11}, L_{21}, (y)L_{22}) : B}$$

where $\mathsf{Cut}((z)L_{11}, L_{21}, (y)L_{22})$ is the corresponding new construction. This is a key cut.

Consider the derivations

$$\begin{array}{c} d_1 \\ \vdots \\ \Gamma \vdash L_1 : A \end{array} \qquad \begin{array}{c} d_2 \\ \vdots \\ \Gamma, x : A \vdash L_2 : B \end{array}$$

We say that this pair of derivations (by this order) constitutes an *implicit* cut. Now it should be clear how to define the *complete right permutation of* d_1 *over* d_2 *at* x. It is as if we completely right permuted the "ghost" cut that this implicit cut is. The complete right permutation of a cut may then be defined as the complete right permutation of its left subderivation over its right permutation at the cut variable.

Similarly one may define the *complete left permutation of* d_2 *over* d_1 *with* x. The complete left permutation of a cut is then the complete left permutation of its right subderivation over its left subderivation with its cut variable. In the particular case of an implicit right permuted cut

$$\begin{array}{c} d_1 \\ \vdots \\ \Gamma \vdash L_1 : A_1 \supset A_2 \end{array} \qquad \begin{array}{c} d_{21} \\ \vdots \\ \Gamma \vdash L_{21} : A_1 \end{array} \qquad \begin{array}{c} d_{22} \\ \vdots \\ \Gamma, y : A_2 \vdash L_{22} : B \end{array}$$

we may also refer to the *complete left permutation of* d_{21} *and* d_{22} *over* d_1 (without any variable, because the right cut formula is main and linear).

Finally, the *complete permutation of* d_1 *over* d_2 *at* x is like the complete right permutation of d_1 over d_2 at x, except that instead of generating right permuted cuts, we immediately perform their complete left permutation.

2.2 Herbelin's system

In [Howard, 1980] (written in the late 1960's), Howard attributes to Curry the remark that, if one wants to generate "irreducible" λ-terms alone, then one should replace application by a new term-formation rule, building $xN_1...N_k$ from given $N_1, ..., N_k$, with typing rule (ignoring contexts)

$$\frac{\ldots \vdash N_1 : A_1 \quad \ldots \quad \ldots \vdash N_k : A_k}{\ldots, x : A_1 \supset \ldots \supset A_k \supset B \vdash xN_1\ldots N_k : B} \tag{2.2}$$

replacing usual elimination rule. Then Howard observes that this typing rule can be obtained by k applications of Gentzen's left rule (plus an axiom). Explicitly,

$$\dfrac{\begin{array}{c}\vdots\\ \ldots \vdash A_1\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \ldots\vdash A_2\end{array}\qquad \begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ \ldots\vdash A_k\end{array}\qquad \dfrac{}{\ldots, z_k : \mathbf{B}\vdash B}\,Ax}{\ldots, z_{k-1} : \mathbf{A_k}\supset\mathbf{B}\vdash B}\,Left\\ \vdots \\ \ldots, z_2 : \mathbf{A_3}\supset\ldots\supset\mathbf{A_k}\supset\mathbf{B}\vdash B\end{array}}{\ldots, z_1 : \mathbf{A_2}\supset\ldots\supset\mathbf{A_k}\supset\mathbf{B}\vdash B}\,Left}{\ldots, x : \mathbf{A_1}\supset\mathbf{A_2}\supset\ldots\supset\mathbf{A_k}\supset\mathbf{B}\vdash B}\,Left \tag{2.3}$$

In fact, we will explain in detail in Chapter 5 that what is happening here is a mapping of normal natural deduction proofs into cut-free sequent calculus derivations introduced in [Prawitz, 1965], except that, instead of (2.2), Prawitz uses k elimination rules and talks about the *main branch* of a normal proof, *i.e.* the sequence of bold formulas in

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{}{\ldots, x\vdash x : \mathbf{A_1}\supset\ldots\supset\mathbf{A_k}\supset\mathbf{B}}\,Var \qquad \begin{array}{c}\vdots\\ \ldots\vdash N_1 : A_1\end{array}}{\ldots, x\vdash xN_1 : \mathbf{A_2}\supset\ldots\supset\mathbf{A_k}\supset\mathbf{B}}\,Elim\\ \vdots\\ \ldots, x\vdash xN_1\ldots N_{k-1} : \mathbf{A_k}\supset\mathbf{B}\end{array} \qquad \begin{array}{c}\vdots\\ \ldots\vdash N_k : A_k\end{array}}{\ldots, x\vdash xN_1\ldots N_k : \mathbf{B}}\,Elim$$

Observe how the sequence of formulas in the main branch corresponds to the sequence of bold formulas in (2.3), except that, as it were, the main branch was *turned upside down*. This is a typical phenomenon that one should bear in mind.

By the observation of (2.3), one realizes that in the range of this mapping of natural deduction into sequent calculus there are derivations of a particular kind. Actually, all formulas declared with z_i in (2.3) ($i = 1, ..., k$) are linear. Only recently the importance of this fact was fully recognised [Herbelin, 1995, Danos et al., 1997, Mints, 1996, Dyckhoff and Pinto, 1999].

Definition 2 *A left inference is* canonical *if the active formula of its right premiss is main and linear. A sequent calculus derivation is canonical if all left inferences occurring in it are canonical.*

Indeed, every left inference in (2.3) is canonical. Conversely, let us see the effect of this restriction on derivations. Let d be a derivation ending with a left inference introducing $A_1 \supset B_1$ and consider the active formula B_1 of its right premiss. It is main and linear, and, moreover: (1) if it is not logical, it is main in an axiom, and the right subderivation of the left inference consists of this axiom alone. (2) if it is logical, it is main in a canonical left inference. Now look again at the active formula of the right premiss of this new left inference.

By means of this process, while going upwards through the rightmost branch of d, we visit the sequence $B_1, B_2, ...B_k$ of the main formulas of successive (possibly zero) left inferences, ending in the left-main formula B_k of an axiom. Let us put $B_0 = A_1 \supset B_1$ and call the sequence $B_0, B_1, B_2, ...B_k$ the *principal path* of d. This is exactly as the sequence of bold formulas in (2.3).

Now, there is a difference between all these left inferences that we visit, and the bottom-most left inference of d, because the main formula of the latter is not necessarily linear. Moreover, we know that the conclusion of a left inference of the former kind is the right premiss of another left inference. The same is not true of the bottom-most inference of d. By the same reason, we may distinguish two kinds of axioms. The first is the kind of axiom we find at the top of the rightmost branch of d. We know that its conclusion is the right premiss of a left inference. The second kind of axiom is an unrestricted one.

The splitting of axioms and of left inferences into two cases may be expressed in the term language for the sequent calculus used above by the existence of two left constructors $\mathsf{L}(x, L_1, (\_)L_2)$ and $\mathsf{L}(\_, L_1, (\_)L_2)$ and two axiom constructors $\mathsf{Ax}(x)$ and $\mathsf{Ax}(\_)$. Variables are omitted because when a linear formula that just became main is immediately going to be active in the next inference. However, how do we express that, in $\mathsf{L}(\_, L_1, (\_)L_2)$, L_2 has to be either $\mathsf{Ax}(\_)$ or another $\mathsf{L}(\_, L_3, (\_)L_4)$?

A solution due to [Herbelin, 1995] is to arrange the syntax into two classes (we concentrate on cut-free proofs for the moment)

$$\begin{array}{rcl} L & ::= & \mathsf{Ax}(x) \mid \mathsf{L}(x, L, (\_)K) \mid \mathsf{R}((x)L) \\ K & ::= & \mathsf{Ax}(\_) \mid \mathsf{L}(\_, L, (\_)K) \end{array} \tag{2.4}$$

where K annotates proofs introducing a linear formula on the LHS of sequents. The actual syntax of *op. cit.* is

$$\begin{array}{rcl} u, v, t & ::= & xl \mid \lambda x.t \\ l, l' & ::= & [] \mid t :: l \end{array} \tag{2.5}$$

with typing rules given in Table 2.2. There are axiom, left and right rules (named Ax, Lft and *Right*) and a *dereliction* rule Der. This terminology comes from a connection with linear logic explained in [Danos et al., 1995]. We refer to Lft as *Herbelin's left rule*. Rules operate on two kinds of sequents

$$\Gamma; - \vdash t : A \tag{2.6}$$

and

$$\Gamma; B \vdash l : A \tag{2.7}$$

both containing a distinguished position in the LHS, called the "stoup" - a device invented in [Girard, 1991]. The stoup either is empty, as in (2.6), or contains a formula, as in (2.7). Observe that: (1) the active formula in the right premiss of Herbelin's left rule is in the stoup. (2) the only rules whose conclusion is a sequent with a formula in the stoup are Lft and Ax. (3) The formula introduced by Lft

Table 2.2: Inference rules for Herbelin's sequent calculus

$$Ax\ \frac{}{\Gamma; A \vdash [] : A} \qquad Der\ \frac{\Gamma; A \vdash l : B}{\Gamma, x : A; - \vdash xl : B}$$

$$Lft\ \frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C} \qquad Right\ \frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B} x \notin \Gamma$$

is linear. Therefore, every formula in the stoup belongs to some the principal path.

Constructors $\lambda x.t$, $[]$ and $u :: l$ correspond to $\mathsf{R}((x)L)$, $\mathsf{Ax}(\_)$ and $\mathsf{L}(\_, L, (\_)K)$. In terms of derivations, *Der* allows a formula to leave the stoup, possibly causing a contraction, if $x \in \Gamma$. The main formula of *Der* (*i.e.* the displayed occurrence of A in the conclusion of *Der*) is the bottom-most formula of some principal path. With dereliction we recover as $x[]$ and $x(u :: l)$ the versions $\mathsf{Ax}(x)$ and $\mathsf{L}(x, L, (\_)K)$ of axiom and left rule.

Let us go back to (2.3) and see how Herbelin's system annotates a principal path. The z_i disappear, as each formula in the principal path is in the stoup. Suppose each sequent $... \vdash A_i$ is annotated with u_i. Then the displayed axiom gets $[]$, the left inference just below gets $u_k :: []$, and so on, until we get $[u_2, ..., u_k]$. The bottom-most left inference is annotated with a combination of dereliction and Herbelin's left inference. We get $x(u_1 :: [u_2, ..., u_k])$, that is $x[u_1, ..., u_k]$.

Besides dereliction, there is an evident difference between syntaxes (2.4) and (2.5) in that the latter suggests and intended interpretation. The right constructor is λ-abstraction and xl is like x applied to a *list* of arguments. This interpretation is supported by the fact, firstly observed in [Herbelin, 1995], that there is a bijection between normal natural deduction proofs and cut-free derivations of the canonical fragment. This bijection is nothing but the mapping between natural deduction and sequent calculus suggested above. The main branch $xN_1...N_k$ is mapped to the principal path $x[u_1, ..., u_k]$. More formally, in the style of [Dyckhoff and Pinto, 1998], this is a mapping Ψ from

$$\begin{array}{rcl} N & ::= & x \,|\, \lambda x.N \,|\, app(A) \\ A & ::= & xN \,| AN \end{array}$$

to (2.5) given by

$$\begin{array}{rcl} \Psi(x) & = & x[] \\ \Psi(\lambda x.N) & = & \lambda x.\Psi N \\ \Psi(app(A)) & = & \Psi'(A, []) \\ & & \\ \Psi'(xN, l) & = & x(\Psi N :: l) \\ \Psi'(AN, l) & = & \Psi'(A, \Psi N :: l) \end{array}$$

A generalisation of this mapping to non-normal proofs will be extensively studied in the following chapters.

In the calculi for the canonical fragment we are going to introduce in Chapter 3, we will never use dereliction, but we will adopt a syntax in the style of (2.5), suggesting a λ-calculus interpretation.

We now consider cut-elimination in Herbelin's system. The simple fact that sequents in Table 2.2 have a distinguished position to the left of $\vdash$ determines the existence of two kinds of cut, *head-cuts* and *mid-cuts*, according to whether the right cut formula is or is not in the stoup, respectively. Actually, Herbelin needed two species for each of these kinds of cuts,

mid-cuts	head-cuts
$t\{x := v\}$	tl
$l\{x := v\}$	ll'

with typing rules as shown in Table 2.3. When we refer to mid or head cut, we mean constructors $t\{x := u\}$ or tl. We refer to $l\{x := u\}$ and ll' as *auxiliary* mid or head cuts, respectively.

Herbelin's $\overline{\lambda}$-calculus is presented in Table 2.4. Rules $4i$ and $5j$ suggest that mid-cuts behave like explicit substitution. Hence the notation $t\{x := u\}$ and

Table 2.3: Cuts for Herbelin's sequent calculus

$$MidCut\,\frac{\Gamma;-\vdash v:A \quad \Gamma,x:A;-\vdash t:B}{\Gamma;-\vdash t\{x:=v\}:B}x\notin\Gamma$$

$$AuxMidCut\,\frac{\Gamma;-\vdash v:A \quad \Gamma,x:A;C\vdash l:B}{\Gamma;C\vdash l\{x:=v\}:B}x\notin\Gamma$$

$$HeadCut\,\frac{\Gamma;-\vdash t:A \quad \Gamma;A\vdash l:B}{\Gamma;-\vdash tl:B}$$

$$AuxHeadCut\,\frac{\Gamma;C\vdash l:A \quad \Gamma;A\vdash l':B}{\Gamma;C\vdash ll':B}$$

$l\{x := u\}$. Rules $3i$ suggest that cut ll' is an explicit append. Similarly to xl, a head cut tl is like the application of t to the list of arguments l.

This calculus is about cut-elimination at least in the following sense: every redex is a cut. Therefore, if t is cut-free (=no subterm is a cut) then t is normal (=irreducible). The converse is also true. If t has a cut, it has an innermost one. Such cut is a redex. Nevertheless, the exact cut-elimination procedure defined by the calculus is only provided in the proof of subject reduction. Then, rule 11 is the key step of cut elimination. Rules $4i$ and $5j$, as well as rule 20, perform stepwise right permutation. Rules 21 and $3i$ perform stepwise left permutation.

This suggests that mid-cuts are right permutable cuts. On the other hand, since the right cut formula of head-cuts is in the stoup, it seems that head-cuts are right permuted cuts. As observed in [Espírito Santo, 2000], this is not exactly true. The first problem is that we are not sure whether a contraction occurred or not in a dereliction xl. In the latter case, mid-cut $(xl)\{x := u\}$ is, in a sense, already right permuted. The second problem is that head cut $t[]$ is a right permutable cut that reduces to t by 20. This is why, in the following chapters, we will neither consider head-cuts of the form $t[]$ nor reduction rule 20. A third problem is that auxiliary cuts take the system outside the canonical fragment,

Table 2.4: The $\overline{\lambda}$-calculus

$$
\begin{array}{rrcl}
(Terms) & u,v,t & ::= & xl \,|\, \lambda x.t \,|\, tl \,|\, t\{x := v\} \\
(Lists) & l,l' & ::= & [] \,|\, t :: l \,|\, ll' \,|\, l\{x := v\}
\end{array}
$$

$$
\begin{array}{lrcl}
(11) & (\lambda x.t)(u :: l) & \rightarrow & t\{x := u\}l \\
 & & & \\
(20) & t[] & \rightarrow & t \\
 & & & \\
(21) & (xl)l' & \rightarrow & x(ll'),\ l' \neq [] \\
 & & & \\
(31) & (u :: l)l' & \rightarrow & u :: (ll') \\
(32) & []l & \rightarrow & l \\
 & & & \\
(41) & (xl)\{x := v\} & \rightarrow & vl\{x := v\} \\
(42) & (yl)\{x := v\} & \rightarrow & yl\{x := v\}, y \neq x \\
(43) & (\lambda y.u)\{x := v\} & \rightarrow & \lambda y.u\{x := v\} \\
 & & & \\
(51) & (u :: l)\{x := v\} & \rightarrow & u\{x := v\} :: l\{x := v\} \\
(52) & []\{x := v\} & \rightarrow & []
\end{array}
$$

because the formula in the stoup, although linear, is no longer necessarily main, and, therefore, a Herbelin's left inference is not necessarily canonical. This is why in the calculi we introduce auxiliary cuts are kept implicit.

Disregarding these minor problems, the cut-elimination procedure associated to $\overline{\lambda}$ is a stepwise right protocol. Mid-cuts are permuted to the right and this occasionally generates head-cuts (rule 41). Head-cuts are permuted to the left. However, a cut is never allowed to permute upwards past another cut (this is what we call a *inter-permutation* of cuts). So, the procedure is essentially an innermost strategy.

Herbelin observed that an inter-permutation of cuts like

$$(44) \quad (tl)\{x := v\} \quad \rightarrow \quad t\{x := v\}l\{x := v\}$$

was required if the cut-elimination procedure was to simulate full β-reduction. However, Herbelin did not consider this reduction rule because it breaks the proof of strong normalisation in [Herbelin, 1995]. In [Dyckhoff and Urban, 2001] it is shown that rule 44 may be added to the calculus without loss of strong normalisation, but further inter-permutations of cuts, among which is

$$(22) \quad (tl)l' \quad \rightarrow \quad t(ll') \ ,$$

are to be allowed for retaining confluence of the calculus.

In Chapter 3, we will design λ-calculi for the canonical fragment in a systematic way. Our main design decision is to adopt right protocols with increasing level of explicitness and stepwise character. We even define a fully explicit system, by making auxiliary cuts explicit. This system will be close to $\overline{\lambda}$ plus 44 and 22. That is, a systematic procedure chose which inter-permutations of cuts were to be admitted. Later on, we will see that, in our setting, 44 and 22 are enough for simulating full β-reduction and retaining confluence - although this was not a requirement impending on how we defined the calculi. Moreover, reduction rule 22 will prove crucial in the computational interpretation of the fragment (*e.g.* it is included verbatim in abstract machines). Actually, 22 is indispensable even for simulating full β reduction, as long as one maps natural deduction into sequent calculus not in the traditional way (Gentzen's mapping [Gentzen, 1935]), but according to a suitable generalisation of Prawitz's mapping [Prawitz, 1965].

What is manifest is that, after the breakthrough that constituted the identification of the canonical fragment, there did not follow the necessary study of cut elimination in this fragment, particularly the study of its relationship with normalisation. The following chapters provide contributions in that direction.

For future reference, we give the following.

Definition 3

$$
\begin{array}{rcll}
\overline{\lambda}_1 & = & \overline{\lambda} & \\
\overline{\lambda}_2 & = & \overline{\lambda}_1 + \{44\} & \\
\overline{\lambda}_3 & = & \overline{\lambda}_2 + \{22\} & = \overline{\lambda}_1 + \{22, 44\}
\end{array}
$$

The following result is due to [Dyckhoff and Urban, 2001].

Theorem 1 *If t is typable in $\overline{\lambda}_i$, then t is strongly normalising (any $i = 1, 2, 3$).*

We define compatible closure for the $\overline{\lambda}_i$-calculi.

Definition 4 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on $Lists$, the compatible closure $\rightarrow_R$ is the least pair of relations $\rightarrow$, the first on $Terms$ and containing the first relation of R, the second on $Lists$ and containing the second relation of R, closed under:*

$$Right\,\frac{t \to t'}{\lambda x.t \to \lambda x.t'} \quad Der\,\frac{l \to l'}{xl \to xl'}$$

$$HeadCut1\,\frac{t \to t'}{tl \to t'l} \quad HeadCut2\,\frac{l \to l'}{tl \to tl'}$$

$$MidCut1\,\frac{t \to t'}{t\{x := v\} \to t'\{x := v\}} \quad MidCut2\,\frac{v \to v'}{t\{x := v\} \to t\{x := v'\}}$$

$$Lft1\,\frac{u \to u'}{u :: l \to u' :: l} \quad Lft2\,\frac{l \to l'}{u :: l \to u :: l'}$$

$$AuxHeadCut1\,\frac{l_0 \to l_0'}{l_0 l_1 \to l_0' l_1} \quad AuxHeadCut2\,\frac{l_1 \to l_1'}{l_0 l_1 \to l_0 l_1'}$$

$$AuxMidCut1\,\frac{l \to l'}{l\{x := v\} \to l'\{x := v\}} \quad AuxMidCut2\,\frac{v \to v'}{l\{x := v\} \to l\{x := v'\}}$$

For instance, for defining $\to_{11}$, take $R = (11, \emptyset)$ in Definition 4. The definition of $\to_{3i}$ $(i = 1, 2, 3, 4)$ is by choosing $R = (3i, \emptyset)$. We could let

$$(3i)_i = 31 \cup 32$$

and, thus, $\to_{(3i)_i}$ (or, simply, $\to_{3i}$) is defined by taking $R = (\emptyset, (3i)_i)$.

Chapter 3

A fragment of sequent calculus

In this chapter we define four calculi of cut-elimination:

$$\begin{tikzcd}
\overline{\lambda}\mathcal{P}h\mathtt{x} \arrow[d, "(_)^{\circ}"'] \\
\lambda\mathcal{P}h\mathtt{x} \arrow[d, "(_)^{\flat}"'] \\
\lambda\mathcal{P}h \arrow[d, "(_)^{-}"'] \\
\lambda\mathcal{P}
\end{tikzcd}$$

In this diagram each arrow is a projection. All cut-elimination procedures associated to these calculi are right protocols, but with different levels of explicitness.

We start with $\lambda\mathcal{P}$, a calculus which only admits key-cuts and whose cut-elimination procedure is fully implicit, in a sense. Then we define $\lambda\mathcal{P}h$, which allows the more general right permuted cuts and includes an independent reduction rule for the complete left permutation of cuts. Next we define $\lambda\mathcal{P}h\mathtt{x}$, in which the complete right permutation of cuts is also separated from the key step and performed stepwise. Finally, we define a fully explicit system $\overline{\lambda}\mathcal{P}h\mathtt{x}$. All these calculi, except $\overline{\lambda}\mathcal{P}h\mathtt{x}$, are in the canonical fragment.

In this chapter we heavily rely on Chapter 2 for notation, terminology and motivation.

3.1 The $\lambda\mathcal{P}$-calculus

The $\lambda\mathcal{P}$-calculus[1] is presented in Table 3.1. Typing rules are in Table 3.2.

Besides constructors for the cut-free canonical fragment, there is a key-cut constructor $(\lambda x.t)(u \cdot l)$, and no other kind of cut. Reduction rules perform, in a sense that will be made precise later, a right protocol. But, since the only kind of explicit cut is key-cuts, most of the stages of cut-elimination are implicit, that is, performed in a single go by calls to meta-operators. These operators are *insert* and *append*, which perform complete left permutation, and *subst*, which performs complete permutation. This explains reduction rules $\beta 1$ and $\beta 2$, which start by performing the key step of cut elimination, but which are forced to immediately perform the permutation of the cuts generated by the key step. Since *subst* performs *complete* permutation, it has to immediately call *insert* whenever it generates a right permuted cut.

Constructor $(\lambda x.t)(u \cdot l)$ binds x in t. By variable convention, x occurs neither in u nor in l.

Definition 5 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on $Lists$, the compatible closure $\rightarrow_R$ is the least pair of relations $\rightarrow$, the first on $Terms$ and containing the first relation of R, the second on $Lists$ and containing the second relation of R, closed under:*

[1] Calculus $\lambda\mathcal{P}$ was defined for the first time, with minor differences, in [Espírito Santo, 2000], with the name λ_H.

Table 3.1: The $\lambda\mathcal{P}$-calculus

$$
\begin{array}{rrcl}
(Terms) & u,v,t & ::= & x \,|\, x(v\cdot l) \,|\, \lambda x.t \,|\, (\lambda x.t)(v\cdot l) \\
(Lists) & l,l' & ::= & [] \,|\, t :: l
\end{array}
$$

$$
\begin{array}{lrcl}
(\beta 1) & (\lambda x.t)(v\cdot []) & \rightarrow & subst(v,x,t) \\
(\beta 2) & (\lambda x.t)(v\cdot (u :: l)) & \rightarrow & insert(u,l,subst(v,x,t))
\end{array}
$$

where

$$
\begin{array}{rcl}
subst(v,x,x) & = & v \\
subst(v,x,y) & = & y,\ y\neq x \\
subst(v,x,x(u\cdot l)) & = & insert(subst(v,x,u),subst(v,x,l),v) \\
subst(v,x,y(u\cdot l)) & = & y(subst(v,x,u)\cdot subst(v,x,l)),\ y\neq x \\
subst(v,x,\lambda y.t) & = & \lambda y.subst(v,x,t) \\
subst(v,x,(\lambda y.t)(u\cdot l)) & = & (\lambda y.subst(v,x,t))(subst(v,x,u)\cdot subst(v,x,l))
\end{array}
$$

$$
\begin{array}{rcl}
subst(v,x,u :: l) & = & subst(v,x,u) :: subst(v,x,l) \\
subst(v,x,[]) & = & []
\end{array}
$$

$$
\begin{array}{rcl}
insert(u,l,x) & = & x(u\cdot l) \\
insert(u,l,x(u'\cdot l')) & = & x(u'\cdot append(l',u :: l)) \\
insert(u,l,\lambda x.t) & = & (\lambda x.t)(u :: l) \\
insert(u,l,(\lambda x.t)(u'\cdot l')) & = & (\lambda x.t)\,(u'\cdot append(l',u :: l))
\end{array}
$$

$$
\begin{array}{rcl}
append(t :: l,l') & = & t :: append(l,l') \\
append([],l') & = & l'
\end{array}
$$

Table 3.2: Typing rules for $\lambda\mathcal{P}$

$$Var\,\frac{}{\Gamma, x:A;-\vdash x:A} \qquad Right\,\frac{\Gamma, x:A;-\vdash t:B}{\Gamma;-\vdash \lambda x.t:A\supset B}x\notin\Gamma$$

$$Left\,\frac{\Gamma, x:A\supset B;-\vdash u:A \quad \Gamma, x:A\supset B;B\vdash l:C}{\Gamma, x:A\supset B;-\vdash x(u\cdot l):C}$$

$$KeyCut\,\frac{\Gamma, x:A;-\vdash t:B \quad \Gamma;-\vdash v:A \quad \Gamma;B\vdash l:C}{\Gamma;-\vdash(\lambda x.t)(v\cdot l):C}x\notin\Gamma$$

$$Ax\,\frac{}{\Gamma;A\vdash []:A} \qquad Lft\,\frac{\Gamma;-\vdash t:A \quad \Gamma;B\vdash l:C}{\Gamma;A\supset B\vdash t::l:C}$$

$$Left1\,\frac{u\to u'}{x(u\cdot l)\to x(u'\cdot l)} \qquad Left2\,\frac{l\to l'}{x(u\cdot l)\to x(u\cdot l')}$$

$$Right\,\frac{t\to t'}{\lambda x.t\to\lambda x.t'} \qquad KeyCut1\,\frac{t\to t'}{(\lambda x.t)(u\cdot l)\to(\lambda x.t')(u\cdot l)}$$

$$KeyCut2\,\frac{u\to u'}{(\lambda x.t)(u\cdot l)\to(\lambda x.t)(u'\cdot l)} \qquad KeyCut3\,\frac{l\to l'}{(\lambda x.t)(u\cdot l)\to(\lambda x.t)(u\cdot l')}$$

$$Lft1\,\frac{u\to u'}{u::l\to u'::l} \qquad Lft2\,\frac{l\to l'}{u::l\to u::l'}$$

For instance, for defining $\to_\beta$, take $R = (\beta 1\cup\beta 2,\emptyset)$ in Definition 5. That is, in $\lambda\mathcal{P}$ we set $\beta = (\beta 1\cup\beta 2,\emptyset)$. One can also define $\to_{\beta 1}$ (resp. $\to_{\beta 2}$) by taking $R = (\beta 1,\emptyset)$ (resp. $R = (\beta 2,\emptyset)$).

Admissible rules

Lemma 1 *In $\lambda\mathcal{P}$, let π_1 be a derivation of $\Gamma;C\vdash l:B$ and π_2 be a derivation of $\Gamma;B\vdash l':A$. The complete left permutation of π_2 over π_1 is a derivation of $\Gamma;C\vdash append(l,l'):A$. In particular, the following rule is admissible:*

$$\frac{\Gamma;C\vdash l:B \qquad \Gamma;B\vdash l':A}{\Gamma;C\vdash append(l,l'):A}$$

Proof : Let π_3 be the complete left permutation of π_2 over π_1. The proof is by induction on l.

Case $l = []$. Then $B = C$ and $\pi_3 = \pi_2$. Since $append(l, l') = l'$, we are done.

Case $l = t_1 :: l_1$. Then there are $\pi_1', \pi_1'', C_1, C_2$ such that π_1 has the form

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma; - \vdash t_1 : C_1\end{matrix} \qquad \begin{matrix}\pi_1'' \\ \vdots \\ \Gamma; C_2 \vdash l_1 : B\end{matrix}}{\Gamma; C_1 \supset C_2 \vdash t_1 :: l_1 : B}\,Lft$$

and $C = C_1 \supset C_2$. Derivation π_3 is

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma; - \vdash t_1 : C_1\end{matrix} \qquad \begin{matrix}\pi_3' \\ \vdots \\ \Gamma; C_2 \vdash append(l_1, l') : A\end{matrix}}{\Gamma; C_1 \supset C_2 \vdash t_1 :: append(l_1, l') : A}\,Lft$$

where π_3' is given by IH. Since $append(l, l') = t_1 :: append(l_1, l')$, we are done. ∎

Lemma 2 *In $\lambda\mathcal{P}$, let π_1 be a derivation of $\Gamma; - \vdash t : C \supset B$, π_2 be a derivation of $\Gamma; - \vdash u : C$ and π_3 a derivation of $\Gamma; B \vdash l : A$. The complete left permutation of π_2 and π_3 over π_1 is a derivation of $\Gamma; - \vdash insert(u, l, t) : A$. In particular, the following rule is admissible:*

$$\frac{\Gamma; - \vdash t : C \supset B \quad \Gamma; - \vdash u : C \qquad \Gamma; B \vdash l : A}{\Gamma; - \vdash insert(u, l, t) : A}$$

Proof : Let π_4 be the complete left permutation of π_2 and π_3 over π_1. The proof is by induction on t.

Case $t = x$. Then, there is Γ' such that π_1 is

$$\frac{}{\Gamma', x : C \supset B; - \vdash x : C \supset B}\,Var$$

and $\Gamma = \Gamma', x : C \supset B$. Derivation π_4 is

$$\dfrac{\begin{matrix}\pi_2 \\ \vdots \\ \Gamma', x : C \supset B; - \vdash u : C\end{matrix} \qquad \begin{matrix}\pi_3 \\ \vdots \\ \Gamma', x : C \supset B; B \vdash l : A\end{matrix}}{\Gamma', x : C \supset B; - \vdash x(u \cdot l) : A}\,Left$$

Since $insert(u, l, t) = x(u \cdot l)$, we are done.

Case $t = x(u' \cdot l')$. Then there are $\pi_1', \pi_1'', \Gamma', D, E$ such that π_1 has the form

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma', x : D \supset E; - \vdash u' : D\end{matrix} \qquad \begin{matrix}\pi_1'' \\ \vdots \\ \Gamma', x : D \supset E; E \vdash l' : C \supset B\end{matrix}}{\Gamma', x : D \supset E; - \vdash x(u' \cdot l') : C \supset B}\ Left$$

and $\Gamma = \Gamma', x : D \supset E$. Derivation π_4 is

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma; - \vdash u' : D\end{matrix} \qquad \dfrac{\begin{matrix}\pi_1'' \\ \vdots \\ \Gamma; E \vdash l' : C \supset B\end{matrix} \quad \pi_5}{\Gamma; E \vdash append(l', u :: l) : A}\ Lemma\ 1}{\Gamma; - \vdash x(u' \cdot append(l', u :: l)) : A}\ Left$$

where π_5 is

$$\dfrac{\begin{matrix}\pi_2 \\ \vdots \\ \Gamma; - \vdash u : C\end{matrix} \qquad \begin{matrix}\pi_3 \\ \vdots \\ \Gamma; B \vdash l : A\end{matrix}}{\Gamma; C \supset B \vdash u :: l : A}\ Lft$$

Since $insert(u, l, t) = x(u' \cdot append(l', u :: l))$, we are done.

Case $t = \lambda x.t'$: Then there is π_1' such that π_1 has the form

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma, x : C; - \vdash t' : B\end{matrix}}{\Gamma; - \vdash \lambda x.t' : C \supset B}\ Right$$

and $x \notin \Gamma$. Derivation π_4 is

$$\dfrac{\begin{matrix}\pi_1' \\ \vdots \\ \Gamma, x : C; - \vdash t' : B\end{matrix} \qquad \begin{matrix}\pi_2 \\ \vdots \\ \Gamma; - \vdash u : C\end{matrix} \qquad \begin{matrix}\pi_3 \\ \vdots \\ \Gamma; B \vdash l : A\end{matrix}}{\Gamma; - \vdash (\lambda x.t')(u :: l) : A}\ KeyCut$$

Since $insert(u, l, t) = (\lambda x.t')(u :: l)$, we are done.

Case $t = (\lambda x.t')(u' \cdot l')$: Then there are $\pi_1', \pi_1'', \pi_1''', D, E$ such that π_1 has the form

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma, x : D; - \vdash t' : E\end{array} \qquad \begin{array}{c}\pi_1''\\ \vdots\\ \Gamma; - \vdash u' : D\end{array} \qquad \begin{array}{c}\pi_1'''\\ \vdots\\ \Gamma; E \vdash l' : C \supset B\end{array}}{\Gamma; - \vdash (\lambda x.t')(u' \cdot l') : C \supset B}\, KeyCut$$

and $x \notin \Gamma$. Let π_5 be as in case $t = x(u' \cdot l')$. Derivation π_4 is

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma, x : D; - \vdash t' : E\end{array} \qquad \begin{array}{c}\pi_1''\\ \vdots\\ \Gamma; - \vdash u' : D\end{array} \qquad \dfrac{\begin{array}{c}\pi_1'''\\ \vdots\\ \Gamma; E \vdash l' : C \supset B\end{array} \qquad \pi_5}{\Gamma; E \vdash append(l', u :: l) : A}\, Lemma\ 1}{\Gamma; - \vdash (\lambda x.t')(u' \cdot append(l', u :: l)) : A}\, KeyCut$$

Since $insert(u, l, t) = (\lambda x.t')(u' \cdot append(l', u :: l))$, we are done. ■

Lemma 3 *In $\lambda\mathcal{P}$, let π_1 be a derivation of $\Gamma, x : B; - \vdash t : A$, π_2 a derivation of $\Gamma, x : B; C \vdash l : A$ and π a derivation of $\Gamma : - \vdash v : B$ such that $x \notin \Gamma$. Then, the complete permutations of π, over π_1 at x, and over π_2 at x, are derivations of $\Gamma; - \vdash subst(v, x, t) : A$ and of $\Gamma; C \vdash subst(v, x, l) : A$, respectively. In particular, the following rules are admissible:*

$$\dfrac{\Gamma; - \vdash v : B \qquad \Gamma, x : B; - \vdash t : A}{\Gamma; - \vdash subst(v, x, t) : A}\, x \notin \Gamma$$

$$\dfrac{\Gamma; - \vdash v : B \qquad \Gamma, x : B; C \vdash l : A}{\Gamma; C \vdash subst(v, x, l) : A}\, x \notin \Gamma$$

Proof : Let π_1^* and π_2^* be the complete permutations of π, over π_1 at x, and over π_2 at x, respectively. The proof is by simultaneous induction on t (with induction hypothesis IH1) and l (with induction hypothesis IH2).

Case $t = x$. Then $B = A$ and $\pi_1^* = \pi$. Since $subst(v, x, t) = v$, we are done.

Case $t = y \neq x$. Then there is Γ' such that π_1 is of the form

$$\frac{}{\Gamma', y : A, x : B; - \vdash y : A}\,Var$$

and $\Gamma = \Gamma', y : A$. Derivation π_1^* is

$$\frac{}{\Gamma', y : A; - \vdash y : A}\,Var$$

Since $subst(v, x, t) = y$, we are done.

Case $t = x(u' \cdot l')$: Then there are π_1', π_2', B_1, B_2 such that π_1 has the form

$$\frac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma, x : B_1 \supset B_2; - \vdash u' : B_1\end{array} \qquad \begin{array}{c}\pi_2'\\ \vdots\\ \Gamma, x : B_1 \supset B_2; B_2 \vdash l' : A\end{array}}{\Gamma, x : B_1 \supset B_2; - \vdash x(u' \cdot l') : A}\,Left$$

and $B = B_1 \supset B_2$. Derivation π_1^* is given by Lemma 2

$$\frac{\begin{array}{c}\pi\\ \vdots\\ \Gamma; - \vdash v : B_1 \supset B_2\end{array} \qquad \begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma; - \vdash subst(v, x, u') : B_1\end{array} \qquad \begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma; - \vdash subst(v, x, l') : A\end{array}}{\Gamma; - \vdash insert(subst(v, x, u'), subst(v, x, l'), v) : A}$$

where π_1^+ and π_2^+ are given by IH1 and IH2, respectively. Since $subst(v, x, t) = insert(subst(v, x, u'), subst(v, x, l'), v)$, we are done.

Case $t = y(u' \cdot l')$, $y \neq x$: Then there are $\pi_1', \pi_2', \Gamma', C_1, C_2$ such that π_1 has the form

$$\frac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma', y : C_1 \supset C_2, x : B; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2'\\ \vdots\\ \Gamma', y : C_1 \supset C_2, x : B; C_2 \vdash l' : A\end{array}}{\Gamma', y : C_1 \supset C_2, x : B; - \vdash y(u' \cdot l') : A}\,Left$$

and $\Gamma = \Gamma', y : C_1 \supset C_2$. Derivation π_1^* is the following $Left$ inference

$$\dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma', y : C_1 \supset C_2; - \vdash subst(v,x,u') : C_1\end{array} \qquad \begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma', y : C_1 \supset C_2; C_2 \vdash subst(v,x,l') : A\end{array}}{\Gamma', y : C_1 \supset C_2; - \vdash y(subst(v,x,u') \cdot subst(v,x,l')) : A}$$

where π_1^+ and π_2^+ are given by IH1 and IH2, respectively. Since $subst(v,x,t) = y(subst(v,x,u') \cdot subst(v,x,l'))$, we are done.

Case $t = \lambda y.t'$: Then there are π_1', A_1, A_2 such that π_1 has the form

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma, y : A_1, x : B; - \vdash t' : A_2\end{array}}{\Gamma, x : B; - \vdash \lambda y.t' : A_1 \supset A_2}\, Right\ ,$$

$A = A_1 \supset A_2$ and $y \notin \Gamma$. Derivation π_1^* is

$$\dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma, y : A_1; - \vdash subst(v,x,t') : A_2\end{array}}{\Gamma; - \vdash \lambda y.subst(v,x,t') : A_1 \supset A_2}\, Right$$

where π_1^+ is given by IH1. Since $subst(v,x,t) = \lambda y.subst(v,x,t')$, we are done.

Case $t = (\lambda y.t')(u' \cdot l')$. Then there are $\pi_1', \pi_1'', \pi_2', C_1, C_2$ such that π_1 has the form

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma, y : C_1, x : B; - \vdash t' : C_2\end{array} \qquad \begin{array}{c}\pi_1''\\ \vdots\\ \Gamma, x : B; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2'\\ \vdots\\ \Gamma, x : B; C_2 \vdash l' : A\end{array}}{\Gamma, x : B; - \vdash (\lambda y.t')(u' \cdot l') : A}\, KeyCut$$

Let us write s for $subst$. Derivation π_1^* is the following $KeyCut$ inference

$$\dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma, y : C_1; - \vdash s(v,x,t') : C_2\end{array} \qquad \begin{array}{c}\pi_1^{++}\\ \vdots\\ \Gamma; - \vdash s(v,x,u') : C_1\end{array} \qquad \begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma; C_2 \vdash s(v,x,l') : A\end{array}}{\Gamma, x : B; - \vdash (\lambda y.s(v,x,t'))(s(v,x,u') \cdot s(v,x,l')) : A}$$

where π_1^+ and π_1^{++} are given by IH1 and π_2^+ is given by IH2. Since $s(v,x,t) = (\lambda y.s(v,x,t'))(s(v,x,u') \cdot s(v,x,l'))$, we are done.

Case $l = []$: Then $C = A$ and π_2^* is

$$\frac{}{\Gamma; A \vdash [] : A}\,Ax$$

Since $subst(v,x,l) = []$, we are done.

Case $l = u' :: l'$: Then there are π_1', π_2', C_1, C_2 such that π_2 has the form

$$\frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : B; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma, x : B; C_2 \vdash l' : A\end{array}}{\Gamma, x : B; C_1 \supset C_2 \vdash u' :: l' : A}\,Lft$$

and $C = C_1 \supset C_2$. Derivation π_2^* is

$$\frac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma; - \vdash subst(v,x,u') : C_1\end{array} \qquad \begin{array}{c}\pi_2^+ \\ \vdots \\ \Gamma; C_2 \vdash subst(v,x,l') : A\end{array}}{\Gamma; C_1 \supset C_2 \vdash subst(v,x,u') :: subst(v,x,l') : A}\,Lft$$

where π_1^+ and π_2^+ are given by IH1 and IH2, respectively. Since $subst(v,x,l) = subst(v,x,u') :: subst(v,x,l')$, we are done. ■

Cut elimination in $\lambda\mathcal{P}$

In this subsection we show in what precise sense the reduction rules of $\lambda\mathcal{P}$ perform cut-elimination.

Rule $\beta 1$: $(\lambda x.t)(u \cdot []) \to subst(u,x,t)$.

$$\frac{\begin{array}{c}\pi_1 \\ \vdots \\ \Gamma, x : A; - \vdash t : B\end{array} \qquad \begin{array}{c}\pi_2 \\ \vdots \\ \Gamma; - \vdash u : A\end{array} \qquad \dfrac{}{\Gamma; B \vdash B}\,Ax}{\Gamma; - \vdash (\lambda x.t)(u \cdot []) : B}\,KeyCut$$

reduces to

$$\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \Gamma;-\vdash u:A\end{array}\qquad \begin{array}{c}\pi_1\\ \vdots\\ \Gamma,x:A;-\vdash t:B\end{array}}{\Gamma;-\vdash subst(u,x,t):B}\ \text{Lemma 3}$$

Rule $\beta 2$: $(\lambda x.t)(u\cdot(v::l))\to insert(v,l,subst(u,x,t))$.

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma,x:A;-\vdash t:B_1\supset B_2\end{array}\qquad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma;-\vdash u:A\end{array}\qquad \dfrac{\begin{array}{c}\pi_3\\ \vdots\\ \Gamma;-\vdash v:B_1\end{array}\qquad \begin{array}{c}\pi_4\\ \vdots\\ \Gamma;B_2\vdash l:C\end{array}}{\Gamma;B_1\supset B_2\vdash v::l:C}\,Lft}{\Gamma;-\vdash(\lambda x.t)(u\cdot(v::l)):C}\,KeyCut$$

reduces to

$$\dfrac{\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \Gamma;-\vdash u:A\end{array}\qquad \begin{array}{c}\pi_1\\ \vdots\\ \Gamma,x:A;-\vdash t:B_1\supset B_2\end{array}}{\Gamma;-\vdash subst(u,x,t):B_1\supset B_2}\,(1)\qquad \begin{array}{c}\pi_3\\ \vdots\\ \Gamma;-\vdash v:B_1\end{array}\qquad \begin{array}{c}\pi_4\\ \vdots\\ \Gamma;B_2\vdash l:C\end{array}}{\Gamma;-\vdash insert(v,l,subst(u,x,t)):C}\,(2)$$

where (1) is by Lemma 3 and (2) by Lemma 2.

Proposition 1 (Subject reduction) *In $\lambda\mathcal{P}$, if $\Gamma;-\vdash t:A$ and $t\to t'$, then $\Gamma;-\vdash t':A$*

Proof: The claim is proved together with the claim that if $\Gamma;B\vdash l:A$ and $l\to l'$, then $\Gamma;B\vdash l':A$, by simultaneous induction on $t\to t'$ and $l\to l'$. All cases but the base cases are routine, and the latter were done above. ■

Permutation of meta-operators

We prove some permutations of operators *subst*, *insert* and *append*.

Lemma 4 *For all u,u',l,l',l'' in $\lambda\mathcal{P}$:*
$append(append(l'',u'::l'),u::l)=append(l'',u'::append(l',u::l))$.

Proof: Straightforward induction on l''. ∎

Lemma 5 *For all* t, u, u', l, l' *in* $\lambda\mathcal{P}$*:*
$insert(u, l, insert(u', l', t)) = insert(u', append(l', u :: l), t)$.

Proof: Straightforward case analysis of t. One case requires Lemma 4. ∎

Lemma 6 *For all* v, u, l, l' *in* $\lambda\mathcal{P}$*:*
$subst(v, x, append(l', u :: l)) = append(subst(v, x, l'), subst(v, x, u) :: subst(v, x, l))$.

Proof: By straightforward induction on l'. ∎

Lemma 7 *For all* v, u, t, l *in* $\lambda\mathcal{P}$*:*
$subst(v, x, insert(u, l, t)) = insert(subst(v, x, u), subst(v, x, l), subst(v, x, t))$.

Proof: By case analysis of t. We write s for *subst*, i for *insert* and a for *append.*

Case $t = x$.

$$
\begin{aligned}
s(v, x, i(u, l, t)) &= s(v, x, i(u, l, x)) \\
&= s(v, x, x(u \cdot l)), \text{ by def. } insert, \\
&= i(s(v, x, u), s(v, x, l), v), \text{ by def. } subst, \\
&= i(s(v, x, u), s(v, x, l), s(v, x, x)), \text{ by def. } subst, \\
&= i(s(v, x, u), s(v, x, l), s(v, x, t)) \ .
\end{aligned}
$$

Case $t = y$, $y \neq x$.

$$
\begin{aligned}
s(v, x, i(u, l, t)) &= s(v, x, i(u, l, y)) \\
&= s(v, x, y(u \cdot l)), \text{ by def. } insert, \\
&= y(s(v, x, u) \cdot s(v, x, l)), \text{ by def. } subst, \\
&= i(s(v, x, u), s(v, x, l), y), \text{ by def. } subst, \\
&= i(s(v, x, u), s(v, x, l), s(v, x, y)) \\
&= i(s(v, x, u), s(v, x, l), s(v, x, t)) \ .
\end{aligned}
$$

Case $t = x(u' \cdot l')$.

$$\begin{aligned}
& s(v, x, i(u, l, t)) \\
= \;& s(v, x, i(u, l, x(u' \cdot l'))) \\
= \;& s(v, x, x(u' \cdot a(l', u :: l)), \text{ by def. } \mathit{insert}, \\
= \;& i(s(v, x, u'), s(v, x, a(l', u :: l)), v), \text{ by def. } \mathit{subst}, \\
= \;& i(s(v, x, u'), a(s(v, x, l'), s(v, x, u) :: s(v, x, l)), v), \text{ by Lemma 6}, \\
= \;& i(s(v, x, u), s(v, x, l), i(s(v, x, u'), s(v, x, l'), v)), \text{ by Lemma 5}, \\
= \;& i(s(v, x, u), s(v, x, l), s(v, x, x(u' \cdot l'))), \text{ by def. } \mathit{subst}, \\
= \;& i(s(v, x, u), s(v, x, l), s(v, x, t)) \ .
\end{aligned}$$

Case $t = y(u' \cdot l')$, $y \neq x$.

$$\begin{aligned}
& s(v, x, i(u, l, t)) \\
= \;& s(v, x, i(u, l, y(u' \cdot l'))) \\
= \;& s(v, x, y(u' \cdot a(l', u :: l)), \text{ by def. } \mathit{insert}, \\
= \;& y(s(v, x, u') \cdot s(v, x, a(l', u :: l))), \text{ by def. } \mathit{subst}, \\
= \;& y(s(v, x, u') \cdot a(s(v, x, l'), s(v, x, u) :: s(v, x, l))), \text{ by Lemma 6}, \\
= \;& i(s(v, x, u), s(v, x, l), y(s(v, x, u') \cdot s(v, x, l'))), \text{ by def. } \mathit{insert}, \\
= \;& i(s(v, x, u), s(v, x, l), s(v, x, y(u' \cdot l'))), \text{ by def. } \mathit{subst}, \\
= \;& i(s(v, x, u), s(v, x, l), s(v, x, t)) \ .
\end{aligned}$$

Case $t = \lambda y.t'$.

$$\begin{aligned}
s(v, x, i(u, l, t)) \;&=\; s(v, x, i(u, l, \lambda x.t')) \\
&=\; s(v, x, (\lambda y.t')(u \cdot l)), \text{ by def. } \mathit{insert}, \\
&=\; (\lambda y.s(v.x.t'))(s(v, x, u) \cdot s(v, x, l)), \text{ by def. } \mathit{subst}, \\
&=\; i(s(v, x, u), s(v, x, l), \lambda y.s(v, x, t')), \text{ by def. } \mathit{insert},
\end{aligned}$$

$$
\begin{aligned}
&= i(s(v,x,u), s(v,x,l)), s(v,x,\lambda y.t')), \text{ by def. } subst, \\
&= i(s(v,x,u), s(v,x,l), s(v,x,t)) \ .
\end{aligned}
$$

Case $t = (\lambda y.t')(u' \cdot l')$.

$$
\begin{aligned}
& s(v,x,i(u,l,t)) \\
= \ & s(v,x,i(u,l,(\lambda y.t')(u' \cdot l'))) \\
= \ & s(v,x,(\lambda y.t')(u' \cdot a(l', u :: l))), \text{ by def. } insert, \\
= \ & (\lambda y.s(v,x,t'))(s(v,x,u') \cdot s(v,x,a(l', u :: l))), \text{ by def. } subst, \\
= \ & (\lambda y.s(v,x,t'))(s(v,x,u') \cdot a(s(v,x,l'), s(v,x,u) :: s(v,x,l))), \text{ by Lemma 6}, \\
= \ & i(s(v,x,u), s(v,x,l), (\lambda y.s(v,x,t'))(s(v,x,u') \cdot s(v,x,l'))), \text{ by def. } insert, \\
= \ & i(s(v,x,u), s(v,x,l), s(v,x,(\lambda y.t')(u' \cdot l'))), \text{ by def. } subst, \\
= \ & i(s(v,x,u), s(v,x,l), s(v,x,t)) \ .
\end{aligned}
$$

■

The permutation of *subst* with itself might be called the *subst*-lemma, by analogy with the substitution lemma of λ-calculus.

Lemma 8 *Let $u, v, t, l \in \lambda\mathcal{P}$, $x \neq y$ and $y \notin FV(v)$. Then:*

1. $subst(v,x,subst(u,y,t)) = subst(subst(v,x,u), y, subst(v,x,t))$.
2. $subst(v,x,subst(u,y,l)) = subst(subst(v,x,u), y, subst(v,x,l))$.

Proof: By simultaneous induction on t and l, with induction hypotheses IH1 and IH2, respectively. We write s for *subst* and i for *insert*.

Case $t = x$.

$$
\begin{aligned}
s(v,x,s(u,y,t)) &= s(v,x,s(u,y,x)) \\
&= s(v,x,x), \text{ as } x \neq y, \\
&= v
\end{aligned}
$$

$$\begin{aligned} &= s(s(v,x,u),y,v), \text{ as } y \notin FV(v), \\ &= s(s(v,x,u),y,s(v,x,x)) \\ &= s(s(v,x,u),y,s(v,x,t)) \ . \end{aligned}$$

Case $t = y$.

$$\begin{aligned} s(v,x,s(u,y,t)) &= s(v,x,s(u,y,y)) \\ &= s(v,x,u) \\ &= s(s(v,x,u),y,y) \\ &= s(s(v,x,u),y,s(v,x,y)), \text{ as } x \neq y, \\ &= s(s(v,x,u),y,s(v,x,t)) \ . \end{aligned}$$

Case $t = z$, $z \neq x, y$.

$$\begin{aligned} s(v,x,s(u,y,t)) &= s(v,x,s(u,y,z)) \\ &= s(v,x,z), \text{ as } z \neq y, \\ &= z \\ &= s(s(v,x,u),y,z)), \text{ as } z \neq y, \\ &= s(s(v,x,u),y,s(v,x,z)), \text{ as } z \neq x, \\ &= s(s(v,x,u),y,s(v,x,t)) \ . \end{aligned}$$

Case $t = x(u' \cdot l')$.

$$\begin{aligned} & s(v,x,s(u,y,t)) \\ = \ & s(v,x,s(u,y,x(u' \cdot l'))) \\ = \ & s(v,x,x(s(u,y,u') \cdot s(u,y,l'))), \text{ as } x \neq y, \\ = \ & i(s(v,x,s(u,y,u')),s(v,x,s(u,y,l')),v) \\ = \ & i(s(s(v,x,u),y,s(v,x,u')),s(s(v,x,u),y,s(v,x,l')),v), \text{ by IH1,IH2}, \\ = \ & i(s(s(v,x,u),y,s(v,x,u')),s(s(v,x,u),y,s(v,x,l')),s(s(v,x,u),y,v)), \end{aligned}$$

$$\begin{aligned}
& \text{as } y \notin FV(v), \\
= \;& s(s(v,x,u),y,i(s(v,x,u'),s(v,x,l'),v)), \text{ by Lemma 7,} \\
= \;& s(s(v,x,u),y,s(v,x,x(u' \cdot l'))) \\
= \;& s(s(v,x,u),y,s(v,x,t)) \;\; .
\end{aligned}$$

Case $t = y(u' \cdot l')$.

$$\begin{aligned}
& s(v,x,s(u,y,t)) \\
= \;& s(v,x,s(u,y,y(u' \cdot l')) \\
= \;& s(v,x,i(s(u,y,u'),s(u,y,l'),u)) \\
= \;& i(s(v,x,s(u,y,u')),s(v,x,s(u,y,l')),s(v,x,u)), \text{ by Lemma 7,} \\
= \;& i(s(s(v,x,u),y,s(v,x,u')),s(s(v,x,u),y,s(v,x,l')),s(v,x,u)), \text{ by IH1,IH2,} \\
= \;& s(s(v,x,u),y,y(s(v,x,u') \cdot s(v,x,l'))) \\
= \;& s(s(v,x,u),y,s(v,x,y(u' \cdot l'))), \text{ as } x \neq y, \\
= \;& s(s(v,x,u),y,s(v,x,t)) \;\; .
\end{aligned}$$

Case $t = z(u' \cdot l'), z \neq x, y$.

$$\begin{aligned}
& s(v,x,s(u,y,t)) \\
= \;& s(v,x,s(u,y,z(u' \cdot l'))) \\
= \;& s(v,x,z(s(u,y,u') \cdot s(u,y,l'))), \text{ as } z \neq y, \\
= \;& z(s(v,x,s(u,y,u')) \cdot s(v,x,s(u,y,l'))), \text{ as } z \neq x, \\
= \;& z(s(s(v,x,u),y,s(v,x,u')) \cdot s(s(v,x,u),y,s(v,x,l'))), \text{ by IH1,IH2,} \\
= \;& s(s(v,x,u),y,z(s(v,x,u') \cdot s(v,x,l'))), \text{ as } z \neq y, \\
= \;& s(s(v,x,u),y,s(v,x,z(u' \cdot l'))), \text{ as } z \neq x, \\
= \;& s(s(v,x,u),y,s(v,x,t)) \;\; .
\end{aligned}$$

Case $t = \lambda z.t'$.

$$\begin{aligned}
s(v,x,s(u,y,t)) &= s(v,x,s(u,y,\lambda z.t')) \\
&= s(v,x,\lambda z.s(u,y,t')) \\
&= \lambda z.s(v,x,s(u,t,t')) \\
&= \lambda z.s(s(v,x,u),y,s(v,x,t')),\ \text{by IH1}, \\
&= s(s(v,x,u),y,\lambda z.s(v,x,t')) \\
&= s(s(v,x,u),y,s(v,x,\lambda z.t')) \\
&= s(s(v,x,u),y,s(v,x,t))\ .
\end{aligned}$$

Case $t = (\lambda z.t')(u' \cdot l')$.

$$\begin{aligned}
& s(v,x,s(u,y,t)) \\
= \ & s(v,x,s(u,y,(\lambda z.t')(u' \cdot l')) \\
= \ & s(v,x,(\lambda z.s(u,y,t'))(s(u,y,u') \cdot s(u,y,l'))) \\
= \ & (\lambda z.s(v,x,s(u,y,t')))(s(v,x,s(u,y,u')) \cdot s(v,x,s(u,y,l'))) \\
= \ & (\lambda z.s(s(v,x,u),y,s(v,x,t')))(s(s(v,x,u),y,s(v,x,u')) \cdot s(s(v,x,u),y,s(v,x,l'))), \\
& \text{by IH1,IH2}, \\
= \ & s(s(v,x,u),y,(\lambda z.s(v,x,t'))(s(v,x,u') \cdot s(v,x,l'))) \\
= \ & s(s(v,x,u),y,s(v,x,(\lambda z.t')(u' \cdot l'))) \\
= \ & s(s(v,x,u),y,s(v,x,t))\ .
\end{aligned}$$

Case $l = []$.

$$\begin{aligned}
s(v,x,s(u,y,l)) &= s(v,x,s(u,y,[])) \\
&= s(v,x,[]) \\
&= [] \\
&= s(s(v,x,u),y,[]) \\
&= s(s(v,x,u),y,s(v,x,[])) \\
&= s(s(v,x,u),y,s(v,x,l))\ .
\end{aligned}$$

Case $l = u' :: l'$.

$$
\begin{aligned}
s(v,x,s(u,y,l)) &= s(v,x,s(u,y,u' :: l')) \\
&= s(v,x,s(u,y,u') :: s(u,y,l')) \\
&= s(v,x,s(u,y,u')) :: s(v,x,s(u,y,l')) \\
&= s(s(v,x,u),y,s(v,x,u')) :: s(s(v,x,u),y,s(v,x,l')), \text{ by IH1,IH2} \\
&= s(s(v,x,u),y,s(v,x,u') :: s(v,x,l')) \\
&= s(s(v,x,u),y,s(v,x,u :: l')) \\
&= s(s(v,x,u),y,s(v,x,l)) \enspace .
\end{aligned}
$$

■

Appendability, insertability, substitutivity

Lemma 9 *In $\lambda\mathcal{P}$, if $l_2 \to l_2'$ then $append(l_1, l_2) \to append(l_1, l_2')$.*

Proof: By induction on l_1. ■

Lemma 10 *In $\lambda\mathcal{P}$:*

1. *(a) If $t \to t'$ then $insert(u,l,t) \to insert(u,l,t')$.*

   (b) If $l_1 \to l_1'$ then $append(l_1, l_2) \to append(l_1', l_2)$.

2. *If $u \to u'$ then $insert(u,l,t) \to insert(u',l,t)$.*

3. *If $l \to l'$ then $insert(u,l,t) \to insert(u,l',t)$.*

Proof: 1. By simultaneous induction on $t \to t'$ and $l_1 \to l_1'$. Cases according to Definition 5. We just do the base cases. The remaining cases are routine. We write s for *subst* and i for *insert*.

Case $\beta 1$.

$$\begin{array}{rcl} i(u, l, (\lambda x.t_0)(t_1 \cdot [])) & = & (\lambda x.t_0)(t_1 \cdot append([], u :: l)) \\ & = & (\lambda x.t_0)(t_1 \cdot (u :: l)) \\ & \rightarrow_{\beta 2} & i(u, l, s(t_1, x, t_0)) \ . \end{array}$$

Case $\beta 2$.

$$\begin{array}{rcl} i(u, l, (\lambda x.t_0)(t_1 \cdot (t_2 :: l_0))) & = & (\lambda x.t_0)(t_1 \cdot append(t_2 :: l_0, u :: l)) \\ & = & (\lambda x.t_0)(t_1 \cdot (t_2 :: append(l_0, u :: l))) \\ & \rightarrow_{\beta 2} & i(t_2, append(l_0, u :: l), s(t_1, x, t_0)) \\ & = & i(u, l, i(t_2, l_0, s(t_1, x, t_0))), \text{ by Lemma 5.} \end{array}$$

2. and 3. are by case analysis of t, using Lemma 9. ■

Lemma 11 *In* $\lambda\mathcal{P}$*:*

1. (a) *If* $t \rightarrow t'$ *then* $subst(u, x, t) \rightarrow subst(u, x, t')$.

   (b) *If* $l \rightarrow l'$ *then* $subst(u, x, l) \rightarrow subst(u, x, l')$.

2. (a) *If* $u \rightarrow u'$ *then* $subst(u, x, t) \rightarrow^* subst(u', x, t)$.

   (b) *If* $u \rightarrow u'$ *then* $subst(u, x, l) \rightarrow^* subst(u', x, l)$.

Proof: 1. By simultaneous induction on $t \rightarrow t'$ and $l \rightarrow l'$. Cases according to Definition 5. We just do the base cases. Case $Left1$ (resp. $Left2$) requires part 2. (resp. part 3.) of Lemma 10. The remaining cases are routine. We write s for $subst$ and i for $insert$.

Case $\beta 1$.

$$\begin{array}{rcl} s(u, x, (\lambda y.t_0)(t_1 \cdot [])) & = & (\lambda y.s(u, x, t_0))(s(u, x, t_1) \cdot []) \\ & \rightarrow_{\beta 1} & s(s(u, x, t_1), y, s(u, x, t_0)) \\ & = & s(u, x, s(t_1, y, t_0)), \text{ by Lemma 8.} \end{array}$$

Case $\beta 2$.

$$
\begin{array}{rl}
 & s(u,x,(\lambda y.t_0)(t_1 \cdot (t_2 :: l_0))) \\
= & (\lambda y.s(u,x,t_0))(s(u,x,t_1) \cdot (s(u,x,t_2) :: s(u,x,l_0))) \\
\rightarrow_{\beta 2} & i(s(u,x,t_2), s(u,x,l_0), s(s(u,x,t_1), y, s(u,x,t_0))) \\
= & i(s(u,x,t_2), s(u,x,l_0), s(u,x,s(t_1,y,t_0))), \text{ by Lemma 8,} \\
= & s(u,x,i(t_2,l_0,s(t_1,y,t_0))), \text{ by Lemma 7.}
\end{array}
$$

2. By simultaneous induction on t and l. Requires part 1. of Lemma 10. ■

3.2 Independent left permutation

The $\lambda\mathcal{P}h$-calculus is presented in Table 3.3. Typing rules are in Table 3.4.

Besides constructors for the cut-free canonical fragment, $\lambda\mathcal{P}h$ includes in its syntax a kind of cut, the right permuted cut $t(u \cdot l)$, which is more general that that found in $\lambda\mathcal{P}$. This construction subsumes both the key-cut $(\lambda x.t)(u \cdot l)$ and the left rule $x(u \cdot l)$ of $\lambda\mathcal{P}$. When $t(u \cdot l)$ does not fall under one of these sub-classes, it is left permutable, and there is a reduction rule h that performs the complete left permutation of such cuts. Notice that, in $\lambda\mathcal{P}h$, $x(u \cdot l)$ is a cut but is not a redex. Moreover, the cuts that are generated by the key step of cut-elimination are completely right permuted (but not completely permuted, as in $\lambda\mathcal{P}$). Therefore, there is an essential difference between *subst* in $\lambda\mathcal{P}$ and here.

Definition 6 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on $Lists$, the compatible closure $\rightarrow_R$ is the least pair of relations $\rightarrow$, the first on $Terms$ and containing the first relation of R, the second on $Lists$ and containing the second relation of R, closed under:*

Table 3.3: The $\lambda\mathcal{P}h$-calculus

$$
\begin{array}{rrcl}
(Terms) & u, v, t & ::= & x \,|\, \lambda x.t \,|\, t(u \cdot l) \\
(Lists) & l, l' & ::= & [] \,|\, t :: l
\end{array}
$$

$$
\begin{array}{rrcl}
(\beta 1) & (\lambda x.t)(u \cdot []) & \rightarrow & subst(u, x, t) \\
(\beta 2) & (\lambda x.t)(u \cdot (v :: l)) & \rightarrow & subst(u, x, t)(v \cdot l) \\
(h) & (t(u \cdot l))(u' \cdot l') & \rightarrow & t(u \cdot append(l, u' :: l'))
\end{array}
$$

where

$$
\begin{array}{rcl}
subst(v, x, x) & = & v \\
subst(v, x, y) & = & y \,, y \neq x \\
subst(v, x, \lambda y.t) & = & \lambda y.subst(v, x, t) \\
subst(v, x, t(u \cdot l)) & = & subst(v, x, t)(subst(v, x, u) \cdot subst(v, x, l))
\end{array}
$$

$$
\begin{array}{rcl}
subst(v, x, u :: l) & = & subst(v, x, u) :: subst(v, x, l) \\
subst(v, x, []) & = & []
\end{array}
$$

$$
\begin{array}{rcl}
append(t :: l, l') & = & t :: append(l, l') \\
append([], l') & = & l'
\end{array}
$$

Table 3.4: Typing rules for $\lambda\mathcal{P}h$

$$Var \frac{}{\Gamma, x : A; - \vdash x : A} \quad Right \frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B} x \notin \Gamma$$

$$HeadCut \frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; - \vdash t(u \cdot l) : C}$$

$$Ax \frac{}{\Gamma; A \vdash [] : A} \quad Lft \frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C}$$

$$Right \frac{t \to t'}{\lambda x.t \to \lambda x.t'} \quad HeadCut1 \frac{t \to t'}{t(u \cdot l) \to t'(u \cdot l)}$$

$$HeadCut2 \frac{u \to u'}{t(u \cdot l) \to t(u' \cdot l)} \quad HeadCut3 \frac{l \to l'}{t(u \cdot l) \to t(u \cdot l')}$$

$$Lft1 \frac{u \to u'}{u :: l \to u' :: l} \quad Lft2 \frac{l \to l'}{u :: l \to u :: l'}$$

For instance, for defining $\to_\beta$, take $R = (\beta 1 \cup \beta 2, \emptyset)$ in Definition 6. That is, in $\lambda\mathcal{P}h$ we set $\beta = (\beta 1 \cup \beta 2, \emptyset)$. One can again define $\to_{\beta 1}$ (resp. $\to_{\beta 2}$) by taking $R = (\beta 1, \emptyset)$ (resp. $R = (\beta 2, \emptyset)$), or define $\to_h$ by taking $R = (h, \emptyset)$.

Admissible rules

Lemma 12 *In $\lambda\mathcal{P}h$, let π_1 be a derivation of $\Gamma; C \vdash l : B$ and π_2 be a derivation of $\Gamma; B \vdash l' : A$. The complete left permutation of π_2 over π_1 is a derivation of $\Gamma; C \vdash append(l, l') : A$. In particular, the following rule is admissible:*

$$\frac{\Gamma; C \vdash l : B \qquad \Gamma; B \vdash l' : A}{\Gamma; C \vdash append(l, l') : A}$$

Proof: As in Lemma 1. ■

Lemma 13 *In $\lambda\mathcal{P}h$, let π_1 be a derivation of $\Gamma, x : B; - \vdash t : A$, π_2 a derivation of $\Gamma, x : B; C \vdash l : A$ and π a derivation of $\Gamma : - \vdash v : B$ such that $x \notin \Gamma$. Then, the complete right permutations of π, over π_1 at x, and over π_2 at x, are derivations of $\Gamma; - \vdash subst(v, x, t) : A$ and of $\Gamma; C \vdash subst(v, x, l) : A$, respectively. In particular, the following rules are admissible:*

$$\frac{\Gamma; - \vdash v : B \qquad \Gamma, x : B; - \vdash t : A}{\Gamma; - \vdash subst(v, x, t) : A}\, x \notin \Gamma$$

$$\frac{\Gamma; - \vdash v : B \qquad \Gamma, x : B; C \vdash l : A}{\Gamma; C \vdash subst(v, x, l) : A}\, x \notin \Gamma$$

Proof : Let π_1^* and π_2^* be the complete right permutations of π, over π_1 at x, and over π_2 at x, respectively. The proof is by simultaneous induction on t (with induction hypothesis IH1) and l (with induction hypothesis IH2).

Cases $t = x$, $t = y \neq x$, $t = \lambda y.t'$, $l = []$ and $l = u' :: l'$ exactly as in the proof of Lemma 3. The remaining case is

Case $t = t'(u' \cdot l')$. Then there are $\pi_1', \pi_1'', \pi_2', C_1, C_2$ such that π_1 has the form

$$\frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : B; - \vdash t' : C_1 \supset C_2\end{array} \qquad \begin{array}{c}\pi_1'' \\ \vdots \\ \Gamma, x : B; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma, x : B; C_2 \vdash l' : A\end{array}}{\Gamma, x : B; - \vdash t'(u' \cdot l') : A}\, HeadCut$$

Let us write s for $subst$. Derivation π_1^* is the following $HeadCut$ inference

$$\frac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma; - \vdash s(v, x, t') : C_1 \supset C_2\end{array} \qquad \begin{array}{c}\pi_1^{++} \\ \vdots \\ \Gamma; - \vdash s(v, x, u') : C_1\end{array} \qquad \begin{array}{c}\pi_2^+ \\ \vdots \\ \Gamma; C_2 \vdash s(v, x, l') : A\end{array}}{\Gamma, x : B; - \vdash s(v, x, t')(s(v, x, u') \cdot s(v, x, l')) : A}$$

where π_1^+ and π_1^{++} are given by IH1 and π_2^+ is given by IH2. Since $s(v, x, t) = s(v, x, t')(s(v, x, u') \cdot subst(v, x, l'))$, we are done.■

Cut elimination in $\lambda\mathcal{P}h$

In this subsection we show in what precise sense the reduction rules of $\lambda\mathcal{P}h$ perform cut-elimination.

Rule $\beta 1$: $(\lambda x.t)(u \cdot []) \rightarrow subst(u, x, t)$.

$$\dfrac{\dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma, x : A; - \vdash t : B\end{matrix}}{\Gamma; - \vdash \lambda x.t : A \supset B}\,\mathit{Right} \qquad \begin{matrix}\pi_2\\ \vdots\\ \Gamma; - \vdash u : A\end{matrix} \qquad \dfrac{}{\Gamma; B \vdash B}\,\mathit{Ax}}{\Gamma; - \vdash (\lambda x.t)(u \cdot []) : B}\,\mathit{HeadCut}$$

reduces to

$$\dfrac{\begin{matrix}\pi_2\\ \vdots\\ \Gamma; - \vdash u : A\end{matrix} \qquad \begin{matrix}\pi_1\\ \vdots\\ \Gamma, x : A; - \vdash t : B\end{matrix}}{\Gamma; - \vdash subst(u, x, t) : B}\,\text{Lemma 13}$$

Rule $\beta 2$: $(\lambda x.t)(u \cdot (v :: l)) \rightarrow subst(u, x, t)(v \cdot l)$.

$$\dfrac{\dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma, x : A; - \vdash t : B_1 \supset B_2\end{matrix}}{\Gamma; - \vdash \lambda x.t : A \supset B_1 \supset B_2}\,\mathit{Right} \qquad \begin{matrix}\pi_2\\ \vdots\\ \Gamma; - \vdash u : A\end{matrix} \qquad \dfrac{\begin{matrix}\pi_3\\ \vdots\\ \Gamma; - \vdash v : B_1\end{matrix} \quad \begin{matrix}\pi_4\\ \vdots\\ \Gamma; B_2 \vdash l : C\end{matrix}}{\Gamma; B_1 \supset B_2 \vdash v :: l : C}\,\mathit{Lft}}{\Gamma; - \vdash (\lambda x.t)(u \cdot (v :: l)) : C}\,\mathit{HeadCut}$$

reduces to

$$\dfrac{\dfrac{\begin{matrix}\pi_2\\ \vdots\\ \Gamma; - \vdash u : A\end{matrix} \qquad \begin{matrix}\pi_1\\ \vdots\\ \Gamma, x : A; - \vdash t : B_1 \supset B_2\end{matrix}}{\Gamma; - \vdash subst(u, x, t) : B_1 \supset B_2}\,(1) \qquad \begin{matrix}\pi_3\\ \vdots\\ \Gamma; - \vdash v : B_1\end{matrix} \qquad \begin{matrix}\pi_4\\ \vdots\\ \Gamma; B_2 \vdash l : C\end{matrix}}{\Gamma; - \vdash subst(u, x, t)(v \cdot l) : C}\,(2)$$

where (1) is by Lemma 13 and (2) is a $\mathit{HeadCut}$ inference.

Rule h: $t(u \cdot l)(u' \cdot l') \rightarrow t(u \cdot append(l, u' :: l'))$.

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma; - \vdash t : D \supset E\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma; - \vdash u : D\end{array} \quad \begin{array}{c}\pi_3\\ \vdots\\ \Gamma; E \vdash l : A \supset B\end{array}}{\Gamma; - \vdash t(u \cdot l) : A \supset B}(1) \quad \begin{array}{c}\pi_4\\ \vdots\\ \Gamma; - \vdash u' : A\end{array} \quad \begin{array}{c}\pi_5\\ \vdots\\ \Gamma; B \vdash l' : C\end{array}}{\Gamma; - \vdash t(u \cdot l)(u' \cdot l') : C}(2)$$

where both (1) and (2) are $HeadCut$ inferences, reduces to

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma; - \vdash t : D \supset E\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma; - \vdash u : D\end{array} \quad \dfrac{\begin{array}{c}\pi_3\\ \vdots\\ \Gamma; E \vdash l : A \supset B\end{array} \quad \dfrac{\begin{array}{c}\pi_4\\ \vdots\\ \Gamma; - \vdash u' : A\end{array} \quad \begin{array}{c}\pi_5\\ \vdots\\ \Gamma; B \vdash l' : C\end{array}}{\Gamma; A \supset B \vdash u' :: l' : C}Lft}{\Gamma; E \vdash append(l, u' :: l') : C}(3)}{\Gamma; - \vdash t(u \cdot l)(u' \cdot l') : C}HeadCut$$

where (3) is by Lemma 12.

Proposition 2 (Subject reduction) *In $\lambda\mathcal{P}h$, if $\Gamma; - \vdash t : A$ and $t \to t'$, then $\Gamma; - \vdash t' : A$*

Proof: The claim is proved together with the claim that if $\Gamma; B \vdash l : A$ and $l \to l'$, then $\Gamma; B \vdash l' : A$, by simultaneous induction on $t \to t'$ and $l \to l'$. All cases but the base cases are routine, and the latter were done above. ■

Relating $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$

We regard the terms of $\lambda\mathcal{P}$ as forming a subset of the terms of $\lambda\mathcal{P}h$. This inclusion is correct because typing rules $Left$ and $KeyCut$ of $\lambda\mathcal{P}$ may be seen as the particular cases of typing rule $HeadCut$ of $\lambda\mathcal{P}h$ in which the inference immediately above the leftmost premiss is a Var or $Right$ inference, respectively.

First, we show that $\lambda\mathcal{P}h$ simulates $\lambda\mathcal{P}$.

Lemma 14 *In $\lambda\mathcal{P}h$, either $t(u \cdot l) = insert(u, l, t)$ or $t(u \cdot l) \to_h insert(u, l, t)$, for all u, t, l in $\lambda\mathcal{P}$.*

Proof: Case analysis of t.

Case $t = x$. $t(u \cdot l) = x(u \cdot l) = insert(u, l, x) = insert(u, l, t)$.

Case $t = x(u_0 \cdot l_0)$. $t(u \cdot l) = (x(u_0 \cdot l_0))(u \cdot l) \to_h x(u_0 \cdot append(l_0, u :: l)) = insert(u, l, x(u_0 \cdot l_0)) = insert(u, l, t)$, as *append* in $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$ coincide for arguments in the latter calculus.

Cases $t = \lambda x.t_0$ and $t = (\lambda x.t_0)(u_0 \cdot l_0)$ are similar to cases $t = x$ and $t = x(u_0 \cdot l_0)$ respectively. ■

We need to compare *subst* in $\lambda\mathcal{P}h$ with *subst* in $\lambda\mathcal{P}$. In the following, when required, we write the former as *subst*$'$.

Lemma 15 *In $\lambda\mathcal{P}h$, the following holds:*

1. $subst'(u, x, t) \to_h^* subst(u, x, t)$*, for all* u*,* t *in* $\lambda\mathcal{P}$*.*

2. $subst'(u, x, l) \to_h^* subst(u, x, l)$*, for all* u*,* l *in* $\lambda\mathcal{P}$*.*

Proof: By simultaneous induction on t and l. The only interesting case is $t = x(v \cdot l)$. In this case,

$$\begin{array}{rcl} subst'(u,x,t) & = & subst'(u,x,x(v \cdot l)) \\ & = & u(subst'(u,x,v) \cdot subst'(u,x,l)) \\ & \to_h^* & u(subst(u,x,v) \cdot subst(u,x,l)), \text{ by IH1,IH2,} \\ & \to_h \textit{ or } = & insert(subst(u,x,v), subst(u,x,l), u), \text{ by Lemma 14,} \\ & = & subst(u,x,x(v \cdot l)) \\ & = & subst(u,x,t)\ . \end{array}$$

■

Proposition 3 *If* $t \to t'$ *in* $\lambda\mathcal{P}$*, then* $t \to^+ t'$ *in* $\lambda\mathcal{P}h$*.*

Table 3.5: From $\lambda\mathcal{P}h$ to $\lambda\mathcal{P}$

$$
\begin{array}{rcl}
x^- & = & x \\
(\lambda x.t)^- & = & \lambda x.t^- \\
(t(u \cdot l))^- & = & insert(u^-, l^-, t^-) \\
\\
([])^- & = & [] \\
(u :: l)^- & = & u^- :: l^-
\end{array}
$$

Proof: The claim is proved together with the claim that if $l \to l'$ in $\lambda\mathcal{P}$, then $l \to^+ l'$ in $\lambda\mathcal{P}h$, by simultaneous induction on $t \to t'$ and $l \to l'$. Cases according to Definition 5. We just show the base cases. The remaining cases are routine.

Case $\beta 1$. $(\lambda x.t)(u \cdot []) \to_{\beta 1} subst'(u, x, t) \to_h^* subst(u, x, t)$.

Case $\beta 2$. $(\lambda x.t)(u \cdot (v :: l)) \to_{\beta 2} subst'(u, x, t)(v \cdot l) \to_h^* subst(u, x, t)(v \cdot l)$.

In both cases, the last h-steps are by Lemma 15. ■

There is a translation $(\_)^- : \lambda\mathcal{P}h \to \lambda\mathcal{P}$ defined in Table 3.5. We now prove its correctness.

Proposition 4 (Correctness of $(\_)^-$)

1. *If $\lambda\mathcal{P}h$ derives $\Gamma; - \vdash t : A$ then $\lambda\mathcal{P}$ derives $\Gamma; - \vdash t^- : A$.*

2. *If $\lambda\mathcal{P}h$ derives $\Gamma; C \vdash l : A$ then $\lambda\mathcal{P}$ derives $\Gamma; C \vdash l^- : A$.*

Proof : Let π_1 be a derivation in $\lambda\mathcal{P}h$ of $\Gamma; - \vdash t : A$ and π_2 be a derivation in $\lambda\mathcal{P}h$ of $\Gamma; C \vdash l : A$. One proves by simultaneous induction on t (with induction hypothesis IH1) and l (with induction hypothesis IH2) that there are derivations π_1^* and π_2^* in $\lambda\mathcal{P}$ of $\Gamma; - \vdash t^- : A$ and $\Gamma; C \vdash l^- : A$ respectively. The only interesting case is $t = t_0(u_0 \cdot l_0)$. The remaining cases are routine.

Case $t = t_0(u_0 \cdot l_0)$: Then there are $\pi_1', \pi_1'', \pi_2', B, C$ such that π_1 has the form

$$\frac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma; - \vdash t_0 : B \supset C\end{array} \qquad \begin{array}{c}\pi_1''\\ \vdots\\ \Gamma; - \vdash u_0 : B\end{array} \qquad \begin{array}{c}\pi_2'\\ \vdots\\ \Gamma; C \vdash l_0 : A\end{array}}{\Gamma; - \vdash t_0(u_0 \cdot l_0) : A}\ HeadCut$$

Since $t^- = insert(u_0^-, l_0^-, t_0^-)$, we want a derivation π_1^* of $\Gamma; - \vdash insert(u_0^-, l_0^-, t_0^-) : A$. Take π_1^* as

$$\frac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma; - \vdash t_0^- : B \supset C\end{array} \qquad \begin{array}{c}\pi_1^{++}\\ \vdots\\ \Gamma; - \vdash u_0^- : B\end{array} \qquad \begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma; C \vdash l_0^- : A\end{array}}{\Gamma; - \vdash insert(u_0^-, l_0^-, t_0^-) : A}\ \text{Lemma 2}$$

where π_1^+ and π_1^{++} are given by IH1 and π_2^+ is given IH2. ■

Now we show some properties of $(\_)^-$.

Lemma 16 *$t^- = t$, for all t in $\lambda\mathcal{P}$.*

Proof: Immediate, by definition of *insert*. ■

Lemma 17 *$append(l, u' :: l')^- = append(l^-, u'^- :: l'^-)$, for all u', l', l in $\lambda\mathcal{P}h$.*

Proof: By a straightforward induction on l. ■

Lemma 18

1. *$(subst'(v, x, t))^- = subst(v^-, x, t^-)$, for all v, t in $\lambda\mathcal{P}h$.*

2. *$(subst'(v, x, l))^- = subst(v^-, x, l^-)$, for all v, l in $\lambda\mathcal{P}h$.*

Proof: By simultaneous induction on t and l, with induction hypotheses IH1 and IH2, respectively. *subst'* refers to *subst* in $\lambda\mathcal{P}h$. We just show the only interesting case. The remaining cases are routine.

Case $t = t_0(u_0 \cdot l_0)$.

$$
\begin{array}{ll}
& (subst'(v, x, t))^- \\
= & (subst'(v, x, t_0(u_0 \cdot l_0)))^- \\
= & (subst'(v, x, t_0)(subst'(v, x, u_0) \cdot subst'(v, x, l_0)))^- \\
= & insert(subst'(v, x, u_0)^-, subst'(v, x, l_0)^-, subst'(v, x, t_0)^-), \text{ by def. of } (\_)^-, \\
= & insert(subst(v^-, x, u_0^-), subst(v^-, x, l_0^-), subst(v^-, x, t_0^-)), \text{ by IH1,IH2}, \\
= & subst(v^-, x, insert(u_0^-, l_0^-, t_0^-), \text{ by Lemma 7}, \\
= & subst(v^-, x, (t_0(u_0 \cdot l_0))^-), \text{ by def. of } (\_)^-, \\
= & subst(v^-, x, t^-) \ .
\end{array}
$$

■

Proposition 5 *If $t_1 \to t_2$ in $\lambda\mathcal{P}h$, then either $t_1^- = t_2^-$ or $t_1^- \to t_2^-$ in $\lambda\mathcal{P}$.*

Proof: The claim is proved together with the claim that if $l_1 \to l_2$ in $\lambda\mathcal{P}h$, then either $l_1^- = l_2^-$ or $l_1^- \to l_2^-$ in $\lambda\mathcal{P}$, by simultaneous induction on $t_1 \to t_2$ and $l_1 \to l_2$. Cases according to Definition 6.

Case $\beta 1$. Similar to the following case.

Case $\beta 2$.

$$
\begin{array}{rcl}
((\lambda x.t)(u \cdot (v :: l)))^- & = & insert(u^-, (v :: l)^-, (\lambda x.t)^-) \\
& = & insert(u^-, (v^- :: l^-), \lambda x.t^-) \\
& = & (\lambda x.t^-)(u^- \cdot (v^- :: l^-)) \\
& \to_{\beta 2} & insert(v^-, l^-, subst(u^-, x, t^-)) \\
& = & insert(v^-, l^-, subst(u, x, t)^-), \text{ by Lemma 18}, \\
& = & (subst(u, x, t)(v \cdot l))^-, \text{ by def. of } (\_)^-.
\end{array}
$$

Case h.

$$
\begin{array}{rcl}
((t(u_1 \cdot l_1))(u_2 \cdot l_2))^- & = & insert(u_2^-, l_2^-, insert(u_1^-, l_1^-, t^-)), \text{ by def. of } (\_)^-, \\
 & = & insert(u_1^-, append(l_1^-, u_2^- :: l_2^-), t^-), \text{ by Lemma 5}, \\
 & = & insert(u_1^-, append(l_1, u_2 :: l_2)^-, t^-), \text{ by Lemma 17}, \\
 & = & (t(u_1 \cdot append(l_1, u_2 :: l_2)))^-, \text{ by def. of } (\_)^-.
\end{array}
$$

Case $HeadCut1$. Suppose that either $t_3^- = t_4^-$ or $t_3^- \rightarrow t_4^-$. We want either $(t_3(u \cdot l))^- = (t_4(u \cdot l))^-$ or $(t_3(u \cdot l))^- \rightarrow (t_4(u \cdot l))^-$. Now,

$$
\begin{array}{rcl}
(t_3(u \cdot l))^- & = & insert(u^-, l^-, t_3^-) \\
 & = \text{ or } \rightarrow & insert(u^-, l^-, t_4^-), \text{ by part 1 of Lemma 10}, \\
 & = & (t_4(u \cdot l))^- \ .
\end{array}
$$

Case $HeadCut2$. Similar, by part 2. of Lemma 10.

Case $HeadCut3$. Similar, by part 3. of Lemma 10.

The remaining cases follow by IH. ■

Corollary 1 *$\lambda\mathcal{P}h$ is a conservative extension of $\lambda\mathcal{P}$,* i.e. *$t \rightarrow^* t'$ in $\lambda\mathcal{P}$ iff $t \rightarrow^* t'$ in $\lambda\mathcal{P}h$, for all t, t' in $\lambda\mathcal{P}$.*

Proof: By Propositions 3, 5 and Lemma 16. ■

Proposition 6 *$t \rightarrow_h^* t^-$, all t in $\lambda\mathcal{P}h$.*

Proof: The claim is proved together with the claim that $l \rightarrow_h l^-$, all l in $\lambda\mathcal{P}h$, by simultaneous induction on t and l. The only interesting case is $t = t_0(u_0 \cdot l_0)$. In this case,

$$
\begin{array}{rcl}
t & = & t_0(u_0 \cdot l_0) \\
& \rightarrow_h^* & t_0^-(u_0^- \cdot l_0^-), \text{ by IH1,IH2,} \\
& \rightarrow_h \textit{ or } = & insert(u_0^-, l_0^-, t_0^-), \text{ by Lemma 14,} \\
& = & (t_0(u_0 \cdot l_0))^- \\
& = & t^- \ .
\end{array}
$$

■

Corollary 2 *If $\lambda\mathcal{P}$ is confluent, so is $\lambda\mathcal{P}h$.*

Proof: By Propositions 5 and 6. ■

Lemma 19 *If $t_1 \rightarrow_h t_2$ in $\lambda\mathcal{P}h$, then $t_1^- = t_2^-$.*

Proof: It suffices to look at the proof of Proposition 5. ■

Corollary 3 *In $\lambda\mathcal{P}h$, $\rightarrow_h$ is confluent.*

Proof: By Proposition 6 and Lemma 19. ■

Therefore, we may refer to the normal-form mapping $\downarrow_h$.

Corollary 4 *For all t in $\lambda\mathcal{P}h$, $t^- = \downarrow_h (t)$.*

Proof: From Proposition 6 and the fact that each t in $\lambda\mathcal{P}$ (as a term in $\lambda\mathcal{P}h$) is h-normal. ■

Permutation of meta-operators

We prove some permutations of operators *subst* and *append*.

Lemma 20 *For all* u, u', l, l', l'' *in* $\lambda\mathcal{P}h$*:*
$append(append(l'', u' :: l'), u :: l) = append(l'', u' :: append(l', u :: l))$.

Proof: Straightforward induction on l''. ■

Lemma 21 *For all* v, u, l, l' *in* $\lambda\mathcal{P}h$*:*
$subst(v, x, append(l', u :: l)) = append(subst(v, x, l'), subst(v, x, u) :: subst(v, x, l))$.

Proof: Exactly as in Lemma 6. ■

The following is the *subst* lemma for $\lambda\mathcal{P}h$.

Lemma 22 *Let* $u, v, t, l \in \lambda\mathcal{P}h$*,* $x \neq y$ *and* $y \notin FV(v)$*. Then:*

1. $subst(v, x, subst(u, y, t)) = subst(subst(v, x, u), y, subst(v, x, t))$.

2. $subst(v, x, subst(u, y, l)) = subst(subst(v, x, u), y, subst(v, x, l))$.

Proof: By simultaneous induction on t and l, with induction hypotheses IH1 and IH2, respectively. Cases $t = x$, $t = y$, $t = z \notin \{x, y\}$, $t = \lambda z.t'$, $l = []$ and $l = u' :: l'$ exactly as in the proof of Lemma 8. We write s for *subst* in the remaining case.

Case $t = t'(t' \cdot l')$.

$$
\begin{array}{cl}
 & s(v, x, s(u, y, t)) \\
= & s(v, x, s(u, y, t'(u' \cdot l'))) \\
= & s(v, x, s(u, y, t')(s(u, y, u') \cdot s(u, y, l'))) \\
= & s(v, x, s(u, y, t'))(s(v, x, s(u, y, u')) \cdot s(v, x, s(u, y, l'))) \\
= & s(s(v, x, u), y, s(v, x, t'))(s(s(v, x, u), y, s(v, x, u')) \cdot s(s(v, x, u), y, s(v, x, l'))),
\end{array}
$$

$$
\begin{aligned}
& \text{by IH1,IH2,} \\
= \;& s(s(v,x,u),y,s(v,x,t')(s(v,x,u') \cdot s(v,x,l'))) \\
= \;& s(s(v,x,u),y,s(v,x,t'(u' \cdot l'))) \\
= \;& s(s(v,x,u),y,s(v,x,t)) \ .
\end{aligned}
$$

■

Appendability and substitutivity

Lemma 23 *In $\lambda\mathcal{P}h$:*

1. *If $l_1 \to l_1'$, then $append(l_1, l_2) \to append(l_1', l_2)$.*
2. *If $l_2 \to l_2'$, then $append(l_1, l_2) \to append(l_1, l_2')$.*

Proof: 1. By straightforward induction on $l_1 \to l_1'$.

2. By straightforward induction on l_1. ■

Lemma 24 *In $\lambda\mathcal{P}h$:*

1. *(a) If $t \to t'$ then $subst(u, x, t) \to subst(u, x, t')$.*

   (b) If $l \to l'$ then $subst(u, x, l) \to subst(u, x, l')$.

2. *(a) If $u \to u'$ then $subst(u, x, t) \to^* subst(u', x, t)$.*

   (b) If $u \to u'$ then $subst(u, x, l) \to^* subst(u', x, l)$.

Proof: 1. By simultaneous induction on $t \to t'$ and $l \to l'$. Cases according to Definition 6. We just do the base cases. The remaining cases are routine. We write s for *subst*.

Case $\beta 1$.

$$
\begin{aligned}
s(u,x,(\lambda y.t_0)(t_1 \cdot [])) \quad &= \quad (\lambda y.s(u,x,t_0))(s(u,x,t_1) \cdot []) \\
&\to_{\beta 1} \quad s(s(u,x,t_1),y,s(u,x,t_0)) \\
&= \quad s(u,x,s(t_1,y,t_0)), \text{ by Lemma 22.}
\end{aligned}
$$

Case $\beta 2$.

$$\begin{array}{rl} & s(u,x,(\lambda y.t_0)(t_1\cdot(t_2::l_0))) \\ = & (\lambda y.s(u,x,t_0))(s(u,x,t_1)\cdot(s(u,x,t_2)::s(u,x,l_0))) \\ \to_{\beta 2} & s(s(u,x,t_1),y,s(u,x,t_0))(s(u,x,t_2)\cdot s(u,x,l_0)) \\ = & s(u,x,s(t_1,y,t_0))(s(u,x,t_2)\cdot s(u,x,l_0)),\ \text{by Lemma 22,} \\ = & s(u,x,s(t_1,y,t_0)(t_2\cdot l_0))\ \ . \end{array}$$

Case h.

$$\begin{array}{rl} & s(u,x,t_0(t_1\cdot l_1)(t_2\cdot l_2)) \\ = & s(u,x,t_0)(s(u,x,t_1)\cdot s(u,x,l_1))(s(u,x,t_2)\cdot s(u,x,l_2)) \\ \to_h & s(u,x,t_0)(s(u,x,t_1)\cdot \mathit{append}(s(u,x,l_1),s(u,x,t_2)::s(u,x,l_2))) \\ = & s(u,x,t_0)(s(u,x,t_1)\cdot s(u,x,\mathit{append}(l_1,t_2::l_2))),\ \text{by Lemma 21,} \\ = & s(u,x,t_0(t_1\cdot \mathit{append}(l_1,t_2::l_2)))\ \ . \end{array}$$

2. By simultaneous induction on t and l. ■

3.3 Explicit right permutation

The $\lambda\mathcal{P}h\mathtt{x}$-calculus is presented in Table 3.6. Typing rules are in Table 3.7.

The natural next step after the introduction of $\lambda\mathcal{P}h$ would be to separate right permutation from the key step of cut elimination, introducing a new constructor $t\{x := u\}$ for right permutable cuts, and keeping a meta-operator for the complete right permutation. That is, in addition to h, we would take rules $\beta 1$ and $\beta 2$ in $\lambda\mathcal{P}h$ and replace calls to *subst* by $t\{x := u\}$

$$\begin{array}{rcl} (\lambda x.t)(u\cdot[]) & \to & t\{x:=u\} \\ (\lambda x.t)(u\cdot(v::l)) & \to & t\{x:=u\}(v\cdot l) \end{array}$$

Table 3.6: The $\lambda\mathcal{P}h\mathtt{x}$-calculus

$$
\begin{array}{rrcl}
(Terms) & u, v, t & ::= & x \,|\, \lambda x.t \,|\, t(u \cdot l) \,|\, t\{x := v\} \\
(Lists) & l, l' & ::= & [] \,|\, t :: l
\end{array}
$$

$$
\begin{array}{lrcl}
(b1) & (\lambda x.t)(u \cdot []) & \to & t\{x := u\} \\
(b2) & (\lambda x.t)(u \cdot (v :: l)) & \to & t\{x := u\}(v \cdot l) \\
\\
(h) & (t(u \cdot l))(u' \cdot l') & \to & t(u \cdot append(l, u' :: l')) \\
\\
(\mathtt{x}1) & x\{x := v\} & \to & v \\
(\mathtt{x}2) & y\{x := v\} & \to & y, y \neq x \\
(\mathtt{x}3) & (\lambda y.u)\{x := v\} & \to & \lambda y.u\{x := v\} \\
(\mathtt{x}4) & (t(u \cdot l))\{x := v\} & \to & (t\{x := v\})((u\{x := v\}) \cdot sub(v, x, l))
\end{array}
$$

where

$$
\begin{array}{rcl}
append([], l') & = & l' \\
append(u :: l, l') & = & u :: append(l, l') \\
\\
sub(v, x, []) & = & [] \\
sub(v, x, u :: l) & = & (u\{x := v\}) :: sub(v, x, l)
\end{array}
$$

Table 3.7: Typing rules for $\lambda\mathcal{P}h\mathrm{x}$

$$Var\,\frac{}{\Gamma, x : A; - \vdash x : A} \qquad Right\,\frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B}\, x \notin \Gamma$$

$$HeadCut\,\frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; - \vdash t(u \cdot l) : C}$$

$$MidCut\,\frac{\Gamma; - \vdash v : A \quad \Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash t\{x := v\} : B}\, x \notin \Gamma$$

$$Ax\,\frac{}{\Gamma; A \vdash [] : A} \qquad Lft\,\frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C}$$

together with the rule[2]

$$t\{x := u\} \to subst(u, x, t) \ .$$

A calculus organised in this way has an architecture that is as close as possible to that of t-protocol. There is the key step and two "structural" steps, one for right permutation, the other for left permutation, being these permutations performed by meta-operators.

Nevertheless, we did not isolate such a calculus here and immediately took a step further, in that, not only constructor $t\{x := u\}$ is included and right permutation separated from the key step, but also right permutation becomes explicit, that is, performed in stepwise fashion by rules of the calculus. Therefore, there is no *subst* in this $\lambda\mathcal{P}h\mathrm{x}$. As it were, this calculus is a calculus of explicit *subst*.

Constructor $t\{x := u\}$ binds x in t. By variable convention, x does not occur in u.

[2] This is calculus λ_H^+ of [Espírito Santo, 2000]

Definition 7 (Compatible closure) *Given a pair R of binary relations, the first on Terms and the second on Lists, the compatible closure $\rightarrow_R$ is the least pair of relations $\rightarrow$, the first on Terms and containing the first relation of R, the second on Lists and containing the second relation of R, closed under:*

$$Right\,\frac{t \rightarrow t'}{\lambda x.t \rightarrow \lambda x.t'} \qquad HeadCut1\,\frac{t \rightarrow t'}{t(u \cdot l) \rightarrow t'(u \cdot l)}$$

$$HeadCut2\,\frac{u \rightarrow u'}{t(u \cdot l) \rightarrow t(u' \cdot l)} \qquad HeadCut3\,\frac{l \rightarrow l'}{t(u \cdot l) \rightarrow t(u \cdot l')}$$

$$MidCut1\,\frac{t \rightarrow t'}{t\{x := v\} \rightarrow t'\{x := v\}} \qquad MidCut2\,\frac{v \rightarrow v'}{t\{x := v\} \rightarrow t\{x := v'\}}$$

$$Lft1\,\frac{u \rightarrow u'}{u :: l \rightarrow u' :: l} \qquad Lft2\,\frac{l \rightarrow l'}{u :: l \rightarrow u :: l'}$$

For instance, for defining $\rightarrow_b$, take $R = (b1 \cup b2, \emptyset)$ in Definition 7. That is, in $\lambda\mathcal{P}h\mathtt{x}$ we set $b = (b1 \cup b2, \emptyset)$. One can define $\rightarrow_{b1}$ (resp. $\rightarrow_{b2}$) by taking $R = (b1, \emptyset)$ (resp. $R = (b2, \emptyset)$), or define $\rightarrow_h$ by taking $R = (h, \emptyset)$. The definition of $\rightarrow_{\mathtt{x}i}$ ($i = 1, 2, 3, 4$) is by choosing $R = (\mathtt{x}i, \emptyset)$. We will also let

$$\mathtt{x} = \mathtt{x1} \cup \mathtt{x2} \cup \mathtt{x3} \cup \mathtt{x4}$$

and, thus, $\rightarrow_{\mathtt{x}}$ is defined by taking $R = (\mathtt{x}, \emptyset)$.

Admissible rules

Lemma 25 *In $\lambda\mathcal{P}h\mathtt{x}$, let π_1 be a derivation of $\Gamma; C \vdash l : B$ and π_2 be a derivation of $\Gamma; B \vdash l' : A$. The complete left permutation of π_2 over π_1 is a derivation of $\Gamma; C \vdash append(l, l') : A$. In particular, the following rule is admissible:*

$$\frac{\Gamma; C \vdash l : B \qquad \Gamma; B \vdash l' : A}{\Gamma; C \vdash append(l, l') : A}$$

Proof: As in Lemma 1. ■

Lemma 26 *In $\lambda\mathcal{P}h\mathrm{x}$ the following rule is admissible:*

$$\frac{\Gamma; - \vdash v : B \qquad \Gamma, x : B; C \vdash l : A}{\Gamma; C \vdash sub(v, x, l) : A}\, x \notin \Gamma$$

Proof : Let π_2 be a derivation of $\Gamma, x : B; C \vdash l : A$ such that $x \notin \Gamma$. We prove by induction on l that, for any derivation π_1 of $\Gamma : - \vdash v : B$, there is a derivation π_2^* of $\Gamma; - \vdash sub(v, x, l) : A$.

Case $l = []$: Then $C = A$ and $sub(v, x, l) = []$. Hence we want a derivation π_2^* of $\Gamma; A \vdash [] : A$. Take π_2^* as an application of the Ax rule.

Case $l = u' :: l'$: Then there are π_1', π_2', C_1, C_2 such that π_2 has the form

$$\frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : B; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma, x : B; C_2 \vdash l' : A\end{array}}{\Gamma, x : B; C_1 \supset C_2 \vdash u' :: l' : A}\, Lft$$

and $C = C_1 \supset C_2$. Since $sub(v, x, l) = (u'\{x := v\}) :: subst(v, x, l')$, we want a derivation π_2^* of $\Gamma; C_1 \supset C_2 \vdash (u'\{x := v\}) :: subst(v, x, l') : A$. Take π_2^* as

$$\frac{\dfrac{\begin{array}{c}\pi_1 \\ \vdots \\ \Gamma; - \vdash v : B\end{array} \qquad \begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : B; - \vdash u' : C_1\end{array}}{\Gamma; - \vdash u'\{x := v\} : C_1}\, MidCut \qquad \begin{array}{c}\pi_2^+ \\ \vdots \\ \Gamma; C_2 \vdash subst(v, x, l') : A\end{array}}{\Gamma; C_1 \supset C_2 \vdash (u'\{x := v\}) :: subst(v, x, l') : A}\, Lft$$

where π_2^+ is given by IH. ■

Cut elimination in $\lambda\mathcal{P}h\mathrm{x}$

We now see in what precise sense reduction rules of $\lambda\mathcal{P}h\mathrm{x}$ correspond to cut elimination steps.

Rules $\beta1$ and $\beta2$. As for $\lambda\mathcal{P}h$, except that calls to Lemma 13 are replaced by $MidCut$ inferences.

Rule h. Exactly as for $\lambda\mathcal{P}h$, using Lemma 25 instead of Lemma 12.

Rules $\mathtt{x}1, \mathtt{x}2$. Standard cut-elimination steps reducing a cut whose right subderivation is an axiom.

Rules $\mathtt{x}3, \mathtt{x}4$. Standard cut-elimination steps, permuting the cut upwards past the last inference of the right subderivation. In the case of $\mathtt{x}4$, Lemma 26 is used.

Proposition 7 (Subject reduction) *In $\lambda\mathcal{P}h\mathtt{x}$, if $\Gamma; - \vdash t : A$ and $t \to t'$, then $\Gamma; - \vdash t' : A$*

Proof: The claim is proved together with the claim that if $\Gamma; B \vdash l : A$ and $l \to l'$, then $\Gamma; B \vdash l' : A$, by simultaneous induction on $t \to t'$ and $l \to l'$. All cases but the base cases are routine, and the latter were sketched above. ■

Relating $\lambda\mathcal{P}h\mathtt{x}$ and $\lambda\mathcal{P}h$

First, we show that $\lambda\mathcal{P}h\mathtt{x}$ simulates $\lambda\mathcal{P}h$.

Lemma 27 *In $\lambda\mathcal{P}h\mathtt{x}$:*

1. $t\{x := v\} \to_{\mathtt{x}}^{+} subst(v, x, t)$, *for all t, v in $\lambda\mathcal{P}h$.*
2. $sub(v, x, l) \to_{\mathtt{x}}^{*} subst(v, x, l)$, *for all l, v in $\lambda\mathcal{P}h$.*

Proof: By a straightforward, simultaneous induction on t and l. ■

Proposition 8 *If $t \to t'$ in $\lambda\mathcal{P}h$, then $t \to^{+} t'$ in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: The claim is proved together with the claim that if $l \to l'$ in $\lambda\mathcal{P}h$, then $l \to^{+} l'$ in $\lambda\mathcal{P}h\mathtt{x}$, by simultaneous induction on $t \to t'$ and $l \to l'$. Cases according to Definition 6. We only show the base cases. The remaining cases are routine.

Case $\beta 1$. $(\lambda x.t)(u \cdot []) \to_{b1} t\{x := v\} \to_{R}^{+} subst(v, x, t)$, the last reduction being by Lemma 27.

Case $\beta 2$. $(\lambda x.t)(u \cdot (v :: l)) \to_{b2} t\{x := v\}(v \cdot l) \to_{R}^{+} subst(v, x, t)(v \cdot l)$, the last reduction being again by Lemma 27.

Table 3.8: From $\lambda\mathcal{P}h\mathsf{x}$ to $\lambda\mathcal{P}h$

$$
\begin{array}{rcl}
x^\flat & = & x \\
(\lambda x.t)^\flat & = & \lambda x.t^\flat \\
(t(u\cdot l))^\flat & = & t^\flat(u^\flat \cdot l^\flat) \\
(t\{x:=v\})^\flat & = & subst(v^\flat, x, t^\flat) \\
\\
([])^\flat & = & [] \\
(u::l)^\flat & = & u^\flat :: l^\flat
\end{array}
$$

Case h. $(t(u\cdot l))(u'\cdot l') \to_h t(u\cdot append(l, u'::l'))$ in $\lambda\mathcal{P}h\mathsf{x}$ and we are done because *append* in $\lambda\mathcal{P}h\mathsf{x}$ and $\lambda\mathcal{P}h$ coincide for arguments in the latter calculus. ■

We now define a mapping $(\_)^\flat$ from $\lambda\mathcal{P}h\mathsf{x}$ to $\lambda\mathcal{P}h$. The definition is given in Table 3.8 and simply amounts to replace each mid-cut by the corresponding application of operator *subst* of $\lambda\mathcal{P}h$.

Proposition 9 (Correctness of $(\_)^\flat$)

1. *If $\lambda\mathcal{P}h\mathsf{x}$ derives $\Gamma; - \vdash t : A$ then $\lambda\mathcal{P}h$ derives $\Gamma; - \vdash t^\flat : A$.*

2. *If $\lambda\mathcal{P}h\mathsf{x}$ derives $\Gamma; C \vdash l : A$ then $\lambda\mathcal{P}h$ derives $\Gamma; C \vdash l^\flat : A$.*

Proof : Let π_1 be a derivation in $\lambda\mathcal{P}h\mathsf{x}$ of $\Gamma; - \vdash t : A$ and π_2 be a derivation in $\lambda\mathcal{P}h\mathsf{x}$ of $\Gamma; C \vdash l : A$. One proves by simultaneous induction on t (with induction hypothesis IH1) and l (with induction hypothesis IH2) that there are derivations π_1^* and π_2^* in $\lambda\mathcal{P}h$ of $\Gamma; - \vdash t^\flat : A$ and $\Gamma; C \vdash l^\flat : A$ respectively. The only interesting case is $t = t_0\{x := u_0\}$. The remaining cases are routine.

Case $t = t_0\{x := u_0\}$: Then there are π_1', π_1'', B such that π_1 has the form

$$\dfrac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma; - \vdash u_0 : B\end{array} \qquad \begin{array}{c}\pi_1'' \\ \vdots \\ \Gamma, x : B; - \vdash t_0 : A\end{array}}{\Gamma; - \vdash t_0\{x := u_0\} : A}\ MidCut$$

Since $t^\flat = subst(u_0^\flat, x, t_0^\flat)$, we want a derivation π_1^* of $\Gamma; - \vdash subst(u_0^\flat, x, t_0^\flat) : A$. Take π_1^* as

$$\dfrac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma; - \vdash u_0^\flat : B\end{array} \qquad \begin{array}{c}\pi_1^{++} \\ \vdots \\ \Gamma, x : B; - \vdash t_0^\flat : A\end{array}}{\Gamma; - \vdash subst(u_0^\flat, x, t_0^\flat) : A}\ \text{Lemma 13}$$

where π_1^+ and π_1^{++} are given by IH1. ■

Now we show some properties of $(\_)^\flat$.

Lemma 28 *$t^\flat = t$, for all t in $\lambda\mathcal{P}h$.*

Proof: Immediate. ■

Lemma 29

1. *$sub(v, x, l)^\flat = subst(v^\flat, x, l^\flat)$, for all v, l in $\lambda\mathcal{P}h\mathtt{x}$.*
2. *$append(l_1, l_2)^\flat = append(l_1^\flat, l_2^\flat)$, for all l_1, l_2 in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: By straightforward inductions on l and l_1. ■

Proposition 10 *If $t_1 \to t_2$ in $\lambda\mathcal{P}h\mathtt{x}$, then $t_1^\flat \to^* t_2^\flat$ in $\lambda\mathcal{P}h$.*

Proof: The claim is proved together with the claim that if $l_1 \to l_2$ in $\lambda\mathcal{P}h\mathtt{x}$, then $l_1^\flat \to^* l_2^\flat$ in $\lambda\mathcal{P}h$, by simultaneous induction on $t_1 \to t_2$ and $l_1 \to l_2$. Cases according to Definition 7.

Case $b1$.

$$\begin{array}{rcl}((\lambda x.t)(u \cdot []))^\flat & = & (\lambda x.t^\flat)(u^\flat \cdot []) \\ & \rightarrow_{\beta 1} & subst(u^\flat, x, t^\flat) \\ & = & (t\{x := u\})^\flat \ .\end{array}$$

Case $b2$.

$$\begin{array}{rcl}((\lambda x.t)(u \cdot (v :: l)))^\flat & = & (\lambda x.t^\flat)(u^\flat \cdot (v^\flat :: l^\flat)) \\ & \rightarrow_{\beta 2} & subst(u^\flat, x, t^\flat)(v^\flat \cdot l^\flat) \\ & = & (t\{x := u\}(v \cdot l))^\flat \ .\end{array}$$

Case h.

$$\begin{array}{rcl}((t(u_1 \cdot l_1))(u_2 \cdot l_2))^\flat & = & (t^\flat(u_1^\flat \cdot l_1^\flat))(u_2^\flat \cdot l_2^\flat) \\ & \rightarrow_h & t^\flat(u_1^\flat \cdot append(l_1^\flat, u_2^\flat :: l_2^\flat)) \\ & = & t^\flat(u_1^\flat \cdot append(l_1^\flat, (u_2 :: l_2)^\flat)) \\ & = & t^\flat(u_1^\flat \cdot append(l_1, (u_2 :: l_2))^\flat), \text{ by Lemma 29,} \\ & = & (t(u_1 \cdot append(l_1, u_2 :: l_2)))^\flat\end{array}$$

Case x1. $(x\{x := v\})^\flat = subst(v^\flat, x, x) = v^\flat$.
Case x2. $(y\{x := v\})^\flat = subst(v^\flat, x, y) = y = y^\flat$.
Case x3.

$$\begin{array}{rcl}((\lambda y.t)\{x := v\})^\flat & = & subst(v^\flat, x, \lambda y.t^\flat) \\ & = & \lambda y.subst(v^\flat, x, t^\flat) \\ & = & (\lambda y.t\{x := v\})^\flat \ .\end{array}$$

Case x4.

$$
\begin{array}{ll}
 & (t(u \cdot l))\{x := v\})^\flat \\
= & subst(v^\flat, x, t^\flat(u^\flat \cdot l^\flat)) \\
= & subst(v^\flat, x, t^\flat)(subst(v^\flat, x, u^\flat) \cdot subst(v^\flat, x, l^\flat)) \\
= & subst(v^\flat, x, t^\flat)(subst(v^\flat, x, u^\flat) \cdot sub(v, x, l)^\flat), \text{ by Lemma 29,} \\
= & (t\{x := v\})^\flat((u\{x := v\})^\flat \cdot sub(v, x, l)^\flat) \\
= & ((t\{x := v\})(u\{x := v\} \cdot sub(v, x, l)))^\flat \ .
\end{array}
$$

Case $MidCut1$. Suppose $t_3^\flat \to^* t_4^\flat$. We want $(t_3\{x := u\})^\flat \to^* (t_4\{x := u\})^\flat$. Now,

$$
\begin{array}{rcl}
(t_3\{x := u\})^\flat & = & subst(u^\flat, x, t_3^\flat) \\
 & \to^* & subst(u^\flat, x, t_4^\flat), \text{ by part 1. of Lemma 24,} \\
 & = & (t_4\{x := u\})^\flat \ .
\end{array}
$$

Case $MidCut2$. Similar, by part 2. of Lemma 24.

The remaining cases are by IH. ■

Corollary 5 *$\lambda\mathcal{P}h\mathtt{x}$ is a conservative extension of $\lambda\mathcal{P}h$,* i.e. *$t \to^* t'$ in $\lambda\mathcal{P}h$ iff $t \to^* t'$ in $\lambda\mathcal{P}h\mathtt{x}$, for all t, t' in $\lambda\mathcal{P}h$.*

Proof: By Propositions 8 and 10 and Lemma 28. ■

Proposition 11 *$t \to_{\mathtt{x}}^* t^\flat$, for all t in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: The claim is proved together with the claim that $l \to_{\mathtt{x}}^* l^\flat$, for all l in $\lambda\mathcal{P}h\mathtt{x}$, by simultaneous induction on t and l. The only interesting case is $t = t_0\{x := v_0\}$. In this case,

$$
\begin{array}{rcl}
t & = & t_0\{x := v_0\} \\
& \to^*_{\mathtt{x}} & t_0^\flat\{x := v_0^\flat\}, \text{ by IH1,IH2,} \\
& \to^+_{\mathtt{x}} & subst(v_0^\flat, x, t_0^\flat), \text{ by Lemma 27,} \\
& = & (t_0\{x := v_0\})^\flat \\
& = & t^\flat
\end{array}
$$

The cases $t = x$ and $l = []$ are immediate and the remaining cases are by IH. ■

Corollary 6 *If $\lambda\mathcal{P}h$ is confluent, so is $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: By Propositions 10 and 11. ■

Lemma 30 *If $t_1 \to_{\mathtt{x}} t_2$ in $\lambda\mathcal{P}h\mathtt{x}$, then $t_1^\flat = t_2^\flat$.*

Proof: It suffices to look at the proof of Proposition 10. ■

Corollary 7 *In $\lambda\mathcal{P}h\mathtt{x}$, $\to_{\mathtt{x}}$ is confluent.*

Proof: By Proposition 11 and Lemma 30. ■

Therefore, we may refer to the normal-form mapping $\downarrow_{\mathtt{x}}$.

Corollary 8 *For all t in $\lambda\mathcal{P}h\mathtt{x}$, $t^\flat = \downarrow_{\mathtt{x}}(t)$.*

Proof: From Proposition 11 and the fact that each t in $\lambda\mathcal{P}h$ (as a term in $\lambda\mathcal{P}h\mathtt{x}$) is $\mathtt{x}$-normal. ■

Appendability and substitutivity

Lemma 31 *In $\lambda\mathcal{P}h\mathtt{x}$:*

1. *If $l_1 \to l_1'$, then $append(l_1, l_2) \to append(l_1', l_2)$.*

2. *If $l_2 \to l_2'$, then $append(l_1, l_2) \to append(l_1, l_2')$.*

Proof: Exactly as in Lemma 23. ■

Lemma 32 *In $\lambda\mathcal{P}h\mathtt{x}$:*

1. *If $l \to l'$, then $sub(u, x, l) \to sub(u, x, l')$.*

2. *If $u \to u'$, then $sub(u, x, l) \to sub(u', x, l)$.*

Proof: 1. By straightforward induction on $l \to l'$.

2. By straightforward induction on l. ■

3.4 A fully explicit system

The calculi $\lambda\mathcal{P}$, $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}h\mathtt{x}$ are the systems that will deserve our attention in the following chapters. Nevertheless, we conclude this chapter by identifying one further calculus, named $\overline{\lambda}\mathcal{P}h\mathtt{x}$, that happens to be outside the canonical fragment. The point is that $\overline{\lambda}\mathcal{P}h\mathtt{x}$ is a fully explicit system, in the sense that the whole cut-elimination procedure is done by means of local rules, without appeal to meta-operators. It is obtained from $\lambda\mathcal{P}h\mathtt{x}$ simply by making explicit operators *append* and *sub*. The corresponding new constructors are $l(u \cdot l')$ and $l\{x := u\}$. The latter binds x in l. By variable convention, x does not occur in u.

As we will be dealing with fully explicit system, we briefly compare $\overline{\lambda}\mathcal{P}h\mathtt{x}$ with Herbelin's original $\overline{\lambda}$-calculus. It will become clear that a fully explicit right protocol defined over the canonical fragment requires inter-permutations 44 and 22 to be added to $\overline{\lambda}$. Moreover, the simulation of $\overline{\lambda}\mathcal{P}h\mathtt{x}$ by $\overline{\lambda}_3$ will allow the reuse of the strong normalisability of $\overline{\lambda}_3$.

The $\overline{\lambda}\mathcal{P}h\mathtt{x}$-calculus is defined on Table 3.9. Typing rules are in Table 3.10.

Table 3.9: The $\overline{\lambda}\mathcal{P}h\mathtt{x}$-calculus

$$\begin{array}{lrcl}
(Terms) & u, v, t & ::= & x \,|\, \lambda x.t \,|\, t(u \cdot l) \,|\, t\{x := v\} \\
(Lists) & l, l' & ::= & [] \,|\, t :: l \,|\, l(u \cdot l') \,|\, l\{x := u\}
\end{array}$$

$$\begin{array}{lrcl}
(b1) & (\lambda x.t)(u \cdot []) & \rightarrow & t\{x := u\} \\
(b2) & (\lambda x.t)(u \cdot (v :: l)) & \rightarrow & t\{x := u\}(v \cdot l) \\
 & & & \\
(\overline{h}) & (t(u \cdot l))(u' \cdot l') & \rightarrow & t(u \cdot (l(u' \cdot l'))) \\
 & & & \\
(\mathtt{x}1) & x\{x := v\} & \rightarrow & v \\
(\mathtt{x}2) & y\{x := v\} & \rightarrow & y, y \neq x \\
(\mathtt{x}3) & (\lambda y.u)\{x := v\} & \rightarrow & \lambda y.u\{x := v\} \\
(\overline{\mathtt{x}}4) & (t(u \cdot l))\{x := v\} & \rightarrow & (t\{x := v\})((u\{x := v\}) \cdot (l\{x := v\})) \\
 & & & \\
(\overline{h}1) & [](u' \cdot l') & \rightarrow & u' :: l' \\
(\overline{h}2) & (u :: l)(u' \cdot l') & \rightarrow & u :: (l(u' \cdot l')) \\
 & & & \\
(\overline{\mathtt{x}}41) & []\{x := v\} & \rightarrow & [] \\
(\overline{\mathtt{x}}42) & (u :: l)\{x := v\} & \rightarrow & (u\{x := v\}) :: (l\{x := v\})
\end{array}$$

Table 3.10: Typing rules for $\overline{\lambda}\mathcal{P}h\mathrm{x}$

$$Var\,\frac{}{\Gamma, x:A; - \vdash x:A} \qquad Right\,\frac{\Gamma, x:A; - \vdash t:B}{\Gamma; - \vdash \lambda x.t : A \supset B}\, x \notin \Gamma$$

$$HeadCut\,\frac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad \Gamma; B \vdash l : C}{\Gamma; - \vdash t(u \cdot l) : C}$$

$$MidCut\,\frac{\Gamma; - \vdash v : A \quad \Gamma, x:A; - \vdash t : B}{\Gamma; - \vdash t\{x := v\} : B}\, x \notin \Gamma$$

$$Ax\,\frac{}{\Gamma; A \vdash [] : A} \qquad Lft\,\frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C}$$

$$AuxHeadCut\,\frac{\Gamma; D \vdash l : A \supset B \quad \Gamma; - \vdash u' : A \quad \Gamma; B \vdash l' : C}{\Gamma; D \vdash l(u' \cdot l') : C}$$

$$AuxMidCut\,\frac{\Gamma; - \vdash v : A \quad \Gamma, x:A; C \vdash l : B}{\Gamma; C \vdash l\{x := v\} : B}\, x \notin \Gamma$$

Definition 8 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on $Lists$, the compatible closure $\to_R$ is the least pair of relations $\to$, the first on $Terms$ and containing the first relation of R, the second on $Lists$ and containing the second relation of R, closed under:*

$$Right\,\frac{t \to t'}{\lambda x.t \to \lambda x.t'} \quad HeadCut1\,\frac{t \to t'}{t(u \cdot l) \to t'(u \cdot l)}$$

$$HeadCut2\,\frac{u \to u'}{t(u \cdot l) \to t(u' \cdot l)} \quad HeadCut3\,\frac{l \to l'}{t(u \cdot l) \to t(u \cdot l')}$$

$$MidCut1\,\frac{t \to t'}{t\{x := v\} \to t'\{x := v\}} \quad MidCut2\,\frac{v \to v'}{t\{x := v\} \to t\{x := v'\}}$$

$$Lft1\,\frac{u \to u'}{u :: l \to u' :: l} \quad Lft2\,\frac{l \to l'}{u :: l \to u :: l'} \quad AuxHeadCut1\,\frac{l_0 \to l_0'}{l_0(u \cdot l) \to l_0'(u \cdot l)}$$

$$AuxHeadCut2\,\frac{u \to u'}{l_0(u \cdot l) \to l_0(u' \cdot l)} \quad AuxHeadCut3\,\frac{l \to l'}{l_0(u \cdot l) \to l_0(u \cdot l')}$$

$$AuxMidCut1\,\frac{l \to l'}{l\{x := v\} \to l'\{x := v\}} \quad AuxMidCut2\,\frac{v \to v'}{l\{x := v\} \to l\{x := v'\}}$$

Relating $\overline{\lambda}\mathcal{P}h\mathrm{x}$ and $\lambda\mathcal{P}h\mathrm{x}$

First, we show that $\overline{\lambda}\mathcal{P}h\mathrm{x}$ simulates $\lambda\mathcal{P}h\mathrm{x}$.

Lemma 33 *In $\overline{\lambda}\mathcal{P}h\mathrm{x}$, the following holds:*

1. $l(u' \cdot l') \to^+_{\overline{h}1,\overline{h}2} append(l, u' :: l')$, *all* l, u', l' *in* $\lambda\mathcal{P}h\mathrm{x}$.
2. $l\{x := v\} \to^+_{\overline{\mathrm{x}}41,\overline{\mathrm{x}}42} sub(v, x, l)$, *all* l, v *in* $\lambda\mathcal{P}h\mathrm{x}$.

Proof: Both by straightforward induction on l. ■

Proposition 12 *If $t \to t'$ in $\lambda\mathcal{P}h\mathrm{x}$, then $t \to^+ t'$ in $\overline{\lambda}\mathcal{P}h\mathrm{x}$.*

Proof: The claim is proved together with the claim that, if $l \to l'$ in $\lambda\mathcal{P}h\mathrm{x}$, then $l \to^+ l'$ in $\overline{\lambda}\mathcal{P}h\mathrm{x}$, by simultaneous induction on $t \to t'$ and $l \to l'$. Cases

Table 3.11: From $\overline{\lambda}\mathcal{P}h\mathtt{x}$ to $\lambda\mathcal{P}h\mathtt{x}$

$$
\begin{array}{rcl}
x^\circ & = & x \\
(\lambda x.t)^\circ & = & \lambda x.t^\circ \\
(t(u\cdot l))^\circ & = & t^\circ(u^\circ \cdot l^\circ) \\
(t\{x:=v\})^\circ & = & t^\circ\{x:=v^\circ\} \\
\\
([])^\circ & = & [] \\
(u::l)^\circ & = & u^\circ :: l^\circ \\
(l_1(u\cdot l_2))^\circ & = & append(l_1^\circ, u^\circ :: l_2^\circ) \\
(l\{x:=v\})^\circ & = & sub(v^\circ, x, l^\circ)
\end{array}
$$

according to Definition 7. We just show the interesting base cases, the remaining being routine.

Case h. $(t(u\cdot l))(u'\cdot l') \to_{\overline{h}} t(u\cdot(l(u'\cdot l'))) \to^+_{\overline{h}1,\overline{h}2} t(u\cdot(append(l,u'::l')))$, where the steps $\to_{\overline{h}1,\overline{h}2}$ are by Lemma 33.

Case $\mathtt{x}4$. $(t(u\cdot l))\{x:=v\} \to_{\overline{\mathtt{x}}4} (t\{x:=v\})(u\{x:=v\}\cdot l\{x:=v\}) \to^+_{\overline{\mathtt{x}}41,\overline{\mathtt{x}}42} (t\{x:=v\})(u\{x:=v\}\cdot sub(v,x,l))$, where the steps $\to^+_{\overline{\mathtt{x}}41,\overline{\mathtt{x}}42}$ are by Lemma 33. ■

We now consider a mapping $(\_)^\circ$ from $\overline{\lambda}\mathcal{P}h\mathtt{x}$ to $\lambda\mathcal{P}h\mathtt{x}$. Its definition is given in Table 3.11. Auxiliary cuts are translated to calls to *append* and *sub*.

Proposition 13 (Correctness of $(\_)^\circ$)

1. *If $\overline{\lambda}\mathcal{P}h\mathtt{x}$ derives $\Gamma;-\vdash t:A$ then $\lambda\mathcal{P}h\mathtt{x}$ derives $\Gamma;-\vdash t^\circ:A$.*

2. *If $\overline{\lambda}\mathcal{P}h\mathtt{x}$ derives $\Gamma;C\vdash l:A$ then $\lambda\mathcal{P}h\mathtt{x}$ derives $\Gamma;C\vdash l^\circ:A$.*

Proof : Let π_1 be a derivation in $\overline{\lambda}\mathcal{P}h\mathtt{x}$ of $\Gamma;-\vdash t:A$ and π_2 be a derivation in $\overline{\lambda}\mathcal{P}h\mathtt{x}$ of $\Gamma;C\vdash l:A$. One proves by simultaneous induction on t (with induction hypothesis IH1) and l (with induction hypothesis IH2) that there are derivations π_1^* and π_2^* in $\lambda\mathcal{P}h\mathtt{x}$ of $\Gamma; - \vdash t^\circ : A$ and $\Gamma; C \vdash l^\circ : A$ respectively. We only do the interesting cases. The remaining cases are routine.

Case $l = l_0(u_0 \cdot l_1)$. Then there are $\pi_1', \pi_2', \pi_2'', B_1, B_2$ such that π_2 has the form

$$\dfrac{\begin{array}{c}\pi_2'\\ \vdots\\ \Gamma; C \vdash l_0 : B_1 \supset B_2\end{array} \qquad \begin{array}{c}\pi_1'\\ \vdots\\ \Gamma; \vdash u_0 : B_1\end{array} \qquad \begin{array}{c}\pi_2''\\ \vdots\\ \Gamma; B_2 \vdash l_1 : A\end{array}}{\Gamma; C \vdash l_0(u_0 \cdot l_1) : A}\ AuxHeadCut$$

Since $(l_0(u_0 \cdot l_1))^\circ = append(l_0^\circ, u_0^\circ :: l_1^\circ)$, we want a derivation π_2^* of $\Gamma; C \vdash append(l_0^\circ, u_0^\circ :: l_1^\circ) : A$. Take π_2^* as

$$\dfrac{\begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma; C \vdash l_0^\circ : B_1 \supset B_2\end{array} \qquad \dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma; \vdash u_0^\circ : B_1\end{array} \qquad \begin{array}{c}\pi_2^{++}\\ \vdots\\ \Gamma; B_2 \vdash l_1^\circ : A\end{array}}{\Gamma; B_1 \supset B_2 \vdash u_0^\circ :: l_1^\circ : A}\ Lft}{\Gamma; C \vdash append(l_0^\circ, u_0^\circ :: l_1^\circ) : A}\ Lemma\ 25$$

where π_1^+ is given by IH1 and π_2^+, π_2^{++} are given by IH2.

Case $l = l_0\{x := u_0\}$. Then, there are π_1', π_2', B such that π_2 has the form

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ \Gamma; - \vdash u_0 : B\end{array} \qquad \begin{array}{c}\pi_2'\\ \vdots\\ \Gamma, x : B; C \vdash l_0 : A\end{array}}{\Gamma; C \vdash l_0\{x := u_0\} : A}\ AuxMidCut$$

and $x \notin \Gamma$. Since $(l_0\{x := u_0\})^\circ = sub(u_0^\circ, x, l_0^\circ)$, we want a derivation π_2^* of $\Gamma; C \vdash sub(u_0^\circ, x, l_0^\circ) : A$. Take π_2^* as

$$\dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma; - \vdash u_0^\circ : B\end{array} \qquad \begin{array}{c}\pi_2^+\\ \vdots\\ \Gamma, x : B; C \vdash l_0^\circ : A\end{array}}{\Gamma; C \vdash sub(u_0^\circ, x, l_0^\circ) : A}\ Lemma\ 26$$

where π_1^+ and π_2^+ are given by IH1 and IH2, respectively. ∎

Lemma 34 *$t^\circ = t$, for all t in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: Immediate. ■

Proposition 14 *If $t_1 \to t_2$ in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, then $t_1^\circ \to^* t_2^\circ$ in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: The claim is proved together with the claim that if $l_1 \to l_2$ in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, then $l_1^\circ \to^* l_2^\circ$ in $\lambda\mathcal{P}h\mathtt{x}$, by simultaneous induction on $t_1 \to t_2$ and $l_1 \to l_2$. Cases according to Definition 8.

Cases $b1$, $b2$, $\overline{h}$, $\mathtt{x}1$, $\mathtt{x}2$, $\mathtt{x}3$ and $\overline{\mathtt{x}}4$. One step of these in $\overline{\lambda}\mathcal{P}h\mathtt{x}$ is mapped by $(\_)^\circ$ to a step of the same kind in $\lambda\mathcal{P}h\mathtt{x}$.

Cases $\overline{h}1$, $\overline{h}2$, $\overline{\mathtt{x}}41$ and $\overline{\mathtt{x}}42$. One step of these is collapsed in $\lambda\mathcal{P}h\mathtt{x}$ by $(\_)^\circ$.

Case $AuxHeadCut1$. Follows by part 1. of Lemma 31.

Cases $AuxHeadCut2$ and $AuxHeadCut3$. Follow by part 2. of Lemma 31.

Case $AuxMidCut1$. Follows by part 1. of Lemma 32.

Case $AuxMidCut2$. Follows by part 2. of Lemma 32.

All the remaining cases follow by IH. ■

Corollary 9 *$\overline{\lambda}\mathcal{P}h\mathtt{x}$ is a conservative extension of $\lambda\mathcal{P}h\mathtt{x}$,* i.e. *$t \to^* t'$ in $\lambda\mathcal{P}h\mathtt{x}$ iff $t \to^* t'$ in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, for all t, t' in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: By Propositions 12 and 14 and Lemma 34. ■

Proposition 15 *$t \to_R^* t^\circ$, for all t in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, and $R = \overline{h}1, \overline{h}2, \overline{\mathtt{x}}41, \overline{\mathtt{x}}42$.*

Proof: The claim is proved together with the claim that $l \to_R^* l^\circ$, for all l in $\overline{\lambda}\mathcal{P}h\mathtt{x}$ and same R, by simultaneous induction on t and l. Only two cases deserve attention.

Case $l = l_1(u_0 \cdot l_2)$. Then

$$\begin{array}{rcl} l & = & l_1(u_0 \cdot l_2) \\ & \to_R^* & l_1^\circ(u_0^\circ \cdot l_2^\circ), \text{ by IH1, IH2,} \\ & \to_{\overline{h}1,\overline{h}2}^+ & append(l_1^\circ, u_0^\circ :: l_2^\circ), \text{ by Lemma 33,} \\ & = & (l_1(u_0 :: l_2))^\circ \\ & = & l^\circ \ . \end{array}$$

Case $l = l_0\{x := v_0\}$. Then

$$\begin{array}{rcl} l & = & l_0\{x := v_0\} \\ & \to_R^* & l_0^\circ\{x := v_0^\circ\}, \text{ by IH1, IH2,} \\ & \to_{\overline{\mathtt{x}}41,\overline{\mathtt{x}}42}^+ & sub(v_0^\circ, x, l_0^\circ), \text{ by Lemma 33,} \\ & = & (l_0\{x := v_0\})^\circ \\ & = & l^\circ \ . \end{array}$$

■

Corollary 10 *If $\lambda\mathcal{P}h\mathtt{x}$ is confluent, so is $\overline{\lambda}\mathcal{P}h\mathtt{x}$.*

Proof: By Propositions 14 and 15. ■

Lemma 35 *If $t_1 \to_R t_2$ in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, $R = \overline{h}1, \overline{h}2, \overline{\mathtt{x}}41, \overline{\mathtt{x}}42$, then $t_1^\circ = t_2^\circ$.*

Proof: It suffices to look at the proof of Proposition 14. ■

Corollary 11 *In $\overline{\lambda}\mathcal{P}h\mathtt{x}$, $\to_R$ is confluent ($R = \overline{h}1, \overline{h}2, \overline{\mathtt{x}}41, \overline{\mathtt{x}}42$).*

Proof: By Proposition 15 and Lemma 35. ■

Therefore, we may refer to the normal-form mapping $\downarrow_R$.

Table 3.12: From $\overline{\lambda}\mathcal{P}h\mathtt{x}$ to $\overline{\lambda}_3$

$$
\begin{array}{rcl}
x^{\wr} & = & x[] \\
(\lambda x.t)^{\wr} & = & \lambda x.t^{\wr} \\
(t(u \cdot l))^{\wr} & = & t^{\wr}(u^{\wr} :: l^{\wr}) \\
(t\{x := v\})^{\wr} & = & t^{\wr}\{x := v^{\wr}\} \\
\\
([])^{\wr} & = & [] \\
(u :: l)^{\wr} & = & u^{\wr} :: l^{\wr} \\
(l_1(u_0 \cdot l_2))^{\wr} & = & l_1^{\wr}(u_0^{\wr} :: l_2^{\wr}) \\
(l\{x := v\})^{\wr} & = & l^{\wr}\{x := v^{\wr}\}
\end{array}
$$

Corollary 12 *For all t in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, $t^{\circ} = \downarrow_R (t)$ ($R = \overline{h}1, \overline{h}2, \overline{\mathtt{x}}41, \overline{\mathtt{x}}42$).*

Proof: From Proposition 15 and the fact that each t in $\lambda\mathcal{P}h\mathtt{x}$ (when regarded as a term in $\overline{\lambda}\mathcal{P}h\mathtt{x}$) is R-normal. ■

Comparison with Herbelin's system

The $\overline{\lambda}\mathcal{P}h\mathtt{x}$-calculus is sufficiently close to the original $\overline{\lambda}$-calculus to allow an easy comparison. At the level of syntax, the difference is that the former has constructors x, $t(u \cdot l)$ and $l(u \cdot l')$, whereas the latter has xl, tl and ll'. Hence, $\overline{\lambda}$ seems a little bigger. However, in order to simulate reduction rules $\overline{h}$ and $\overline{\mathtt{x}}4$ of $\overline{\lambda}\mathcal{P}h\mathtt{x}$, one needs to adjoin permutations 44 and 22 to $\overline{\lambda}$. A mapping from $\overline{\lambda}\mathcal{P}h\mathtt{x}$ to $\overline{\lambda}_3$ is suggested in Table 3.12.

Proposition 16 (Correctness of $(\_)^{\wr}$)

1. *If $\overline{\lambda}\mathcal{P}h\mathtt{x}$ derives $\Gamma; - \vdash t : A$ then $\overline{\lambda}_3$ derives $\Gamma; - \vdash t^{\wr} : A$.*

2. If $\overline{\lambda}\mathcal{P}h\mathtt{x}$ *derives* $\Gamma; C \vdash l : A$ *then* $\overline{\lambda}_3$ *derives* $\Gamma; C \vdash l^\wr : A$.

Proof : This is by the usual simultaneous induction. Here we are going to be sketchier. We show how to "simulate" in $\overline{\lambda}_3$ typing rules Var, $HeadCut$ and $AuxHeadCut$ of $\overline{\lambda}\mathcal{P}h\mathtt{x}$.

Var:

$$\dfrac{\dfrac{}{\Gamma; A \vdash [] : A}\,Ax}{\Gamma, x : A; - \vdash x[] : A}\,Der$$

$HeadCut$:

$$\dfrac{\Gamma; - \vdash t^\wr : B \supset C \qquad \dfrac{\Gamma; - \vdash u^\wr : B \quad \Gamma; C \vdash l^\wr : A}{\Gamma; B \supset C \vdash u^\wr :: l^\wr : A}\,Lft}{\Gamma; - \vdash t^\wr(u^\wr :: l^\wr) : A}\,HeadCut$$

$AuxHeadCut$:

$$\dfrac{\Gamma; D \vdash l_1^\wr : B \supset C \qquad \dfrac{\Gamma; - \vdash u_0^\wr : B \quad \Gamma; C \vdash l_2^\wr : A}{\Gamma; B \supset C \vdash u_0^\wr :: l_2^\wr : A}\,Lft}{\Gamma; D \vdash l_1^\wr(u_0^\wr :: l_2^\wr) : A}\,AuxHeadCut$$

■

Proposition 17 *If* $t_1 \rightarrow t_2$ *in* $\overline{\lambda}\mathcal{P}h\mathtt{x}$, *then* $t_1^\wr \rightarrow^+ t_2^\wr$ *in* $\overline{\lambda}_3$.

Proof: The claim is proved together with the claim that, if $l \rightarrow l'$ in $\overline{\lambda}\mathcal{P}h\mathtt{x}$, then $l \rightarrow^+ l'$ in $\overline{\lambda}_3$, by simultaneous induction on $t \rightarrow t'$ and $l \rightarrow l'$. Cases according to Definition 8. We just show three base cases. Non-base case are routine.

Case $\overline{h}$.

$$\begin{array}{rcl}
((t(u_1 \cdot l_1))(u_2 \cdot l_2))^\wr & = & (t^\wr(u_1^\wr :: l_1^\wr))(u_2^\wr :: l_2^\wr) \\
& \rightarrow_{22} & t^\wr((u_1^\wr :: l_1^\wr)(u_2^\wr :: l_2^\wr)) \\
& \rightarrow_{31} & t^\wr(u_1^\wr :: (l_1^\wr(u_2^\wr :: l_2^\wr)) \\
& = & (t(u_1 \cdot (l_1(u_2 \cdot l_2))))^\wr \ .
\end{array}$$

Case $\mathtt{x}1$.

$$
\begin{aligned}
(x\{x := u\})^{\wr} &= (x[])\{x := u^{\wr}\} \\
&\rightarrow_{41} u^{\wr}([]\{x := u^{\wr}\}) \\
&\rightarrow_{52} u^{\wr}[] \\
&\rightarrow_{20} u^{\wr} \ .
\end{aligned}
$$

Case $\overline{\mathtt{x}}4$.

$$
\begin{aligned}
(t(u \cdot l)\{x := v\})^{\wr} &= t^{\wr}(u^{\wr} :: l^{\wr})\{x := v^{\wr}\} \\
&\rightarrow_{44} (t^{\wr}\{x := v^{\wr}\})(u^{\wr} :: l^{\wr})\{x := v^{\wr}\} \\
&\rightarrow_{51} (t^{\wr}\{x := v^{\wr}\})((u^{\wr}\{x := v^{\wr}\}) :: (l^{\wr}\{x := v^{\wr}\})) \\
&= ((t\{x := v\})(u\{x := v\} \cdot l\{x := v\}))^{\wr} \ .
\end{aligned}
$$

■

Hence, strong normalisability of typable terms may flow from $\overline{\lambda}_3$ to $\overline{\lambda}\mathcal{P}h\mathtt{x}$.

Corollary 13 *Let $\mathcal{S} \in \{\lambda\mathcal{P}, \lambda\mathcal{P}h, \lambda\mathcal{P}h\mathtt{x}, \overline{\lambda}\mathcal{P}h\mathtt{x}\}$. If t is typable in $\mathcal{S}$, then t is strongly normalising.*

Proof: From Theorem 1 and Propositions 16, 17, 12, 8 and 3. ■

Chapter 4

Normalisation as cut-elimination

The goal of this chapter is to prove that $\lambda\mathcal{P}$ is isomorphic to λ [1]. We do this by defining an intermediate calculus, named $\lambda\mathcal{N}$, and the following isomorphisms

$$\lambda\mathcal{P} \underset{\Theta}{\overset{\Psi}{\leftrightarrows}} \lambda\mathcal{N} \underset{|\_|}{\overset{\mathcal{N}}{\leftrightarrows}} \lambda$$

The calculus $\lambda\mathcal{N}$ may be seen as a presentation of λ with a separation between the normal subcalculus and a single constructor for β-redexes. The true nature of $\lambda\mathcal{N}$ will only become clear in the next chapter.

4.1 A presentation of λ

A somewhat unusual treatment of λ-calculus is presented in Table 4.1. We name this calculus the $\lambda\mathcal{N}$-calculus. Typing rules are in Table 4.2. This is a presentation in which the normal fragment is obtained by simply omitting one constructor, very much in the style of sequent calculus. Another characteristic is a distinction between *applicative terms* $app(A)$ and *applications* $A \in Apps$. Actually, there are three kinds of applications. Two of them are value applications (xN and $(\lambda x.M)N$); we call the third (AN) an *applicative* application (because A is not a value).

[1] The isomorphism between $\lambda\mathcal{P}$ and λ was announced for the first time in [Espírito Santo, 2000].

Table 4.1: The $\lambda\mathcal{N}$-calculus

$$\begin{array}{rrcl}
(Terms) & M, N & ::= & x \mid \lambda x.M \mid app(A) \\
(Apps) & A & ::= & xN \mid (\lambda x.M)N \mid AN
\end{array}$$

$$\begin{array}{lrcl}
(\beta 1) & app((\lambda x.M)N) & \to & M[N/x] \\
(\beta 2) & ((\lambda x.M)N)N' & \to & M[N/x]@N'
\end{array}$$

where

$$\begin{array}{rcl}
x[N/x] & = & N \\
y[N/x] & = & y,\ y \neq x \\
(\lambda y.M)[N/x] & = & \lambda y.M[N/x] \\
(app(A))[N/x] & = & app(A[N/x])
\end{array}$$

$$\begin{array}{rcl}
(xM)[N/x] & = & N@M[N/x] \\
(yM)[N/x] & = & yM[N/x],\ y \neq x \\
((\lambda y.M)M')[N/x] & = & (\lambda y.M[N/x])M'[N/x] \\
(AM)[N/x] & = & A[N/x]M[N/x]
\end{array}$$

$$\begin{array}{rcl}
x@N & = & xN \\
(\lambda x.M)@N & = & (\lambda x.M)N \\
(app(A))@N & = & AN
\end{array}$$

Table 4.2: Typing rules for $\lambda\mathcal{N}$

$$Var\,\frac{}{\Gamma, x : B \vdash x : B} \quad VElim\,\frac{\Gamma, x : B \supset C \vdash N : B}{\Gamma, x : B \supset C \vdash xN : C}$$

$$Intro\,\frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x.M : B \supset C}\,x \notin \Gamma \quad Redex\,\frac{\Gamma, x : B \vdash M : C \quad \Gamma \vdash N : B}{\Gamma \vdash (\lambda x.M)N : C}\,x \notin \Gamma$$

$$App\,\frac{\Gamma \vdash A : B}{\Gamma \vdash app(A) : B} \quad AElim\,\frac{\Gamma \vdash A : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash AN : C}$$

The syntactic class *Apps* is ranged over by A. Unfortunately A also ranges over types. So, when typing $\lambda\mathcal{N}$-terms, we will only use the meta-variables B, C, D. Care is needed for distinguishing between the two operators of substitution: for each N, x, there are substitution operators $\_[N/x] : Terms \to Terms$ and $\_[N/x] : Apps \to Apps$.

A surprisingly interesting exercise is to define β in this setting. The problem with $(\lambda x.M)N \to M[N/x]$ is that the redex is in *Apps* whereas the *contractum* is in $Terms$. We can fix this by proposing

$$(\beta 1) \quad app((\lambda x.M)N) \quad \to \quad M[N/x] \ .$$

However, with $\beta 1$ alone, we cannot reduce $app(((\lambda x.M)N)N')$. The solution is to also consider the notion of reduction

$$(\beta 2) \quad ((\lambda x.M)N)N' \quad \to \quad M[N/x]@N' \ .$$

Here the operator $@ : Terms \times Terms \to Apps$ (see again Table 4.1) makes available the *application* between two $\lambda\mathcal{N}$-terms, a non-primitive construction in $\lambda\mathcal{N}$. Relation $\beta 1$ is a relation on $Terms$ whereas relation $\beta 2$ is a relation on *Apps*.

Definition 9 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on Apps, the compatible closure $\to_R$ is the least*

$$
\begin{array}{rcl}
\mathcal{N}x & = & x \\
\mathcal{N}(\lambda x.M) & = & \lambda x.\mathcal{N}(M) \\
\mathcal{N}(MN) & = & app(\mathcal{N}(M)@\mathcal{N}(N))
\end{array}
$$

Table 4.3: From λ to $\lambda\mathcal{N}$.

pair of relations $\to$, the first on $Terms$ and containing the first relation of R, the second on $Apps$ and containing the second relation of R, closed under:

$$
Intro\,\frac{M \to M'}{\lambda x.M \to \lambda x.M'} \qquad App\,\frac{A \to A'}{app(A) \to app(A')}
$$

$$
VElim\,\frac{N \to N'}{xN \to xN'}
$$

$$
Redex1\,\frac{M \to M'}{(\lambda x.M)N \to (\lambda x.M')N} \qquad Redex2\,\frac{N \to N'}{(\lambda x.M)N \to (\lambda x.M)N'}
$$

$$
AElim1\,\frac{A \to A'}{AN \to A'N} \qquad AElim2\,\frac{N \to N'}{AN \to AN'}
$$

For instance, for defining $\to_\beta$, take $R = (\beta1, \beta2)$ in Definition 9. That is, in $\lambda\mathcal{N}$ we set

$$
\beta = (\beta1, \beta2) \ .
$$

One can also define $\to_{\beta1}$ (resp. $\to_{\beta2}$) by taking $R = (\beta1, \emptyset)$ (resp. $R = (\emptyset, \beta2)$.

We now prove that $\lambda\mathcal{N}$ and λ are isomorphic.

First we define a mapping $\mathcal{N} : \lambda \to \lambda\mathcal{N}$ in Table 4.3. If $\mathcal{N}(V) = V'$ (where V is a value) and $\mathcal{N}(N_i) = N_i'$, then $\mathcal{N}$ sends $VN_1...N_k$ to $app(V'N_1'...N_k')$. For instance, $\mathcal{N}((xy)z) = app(app(\mathcal{N}x@\mathcal{N}y)@\mathcal{N}z) = app(app(xy)@z) = app((xy)z)$, whereas $\mathcal{N}(x(yz)) = app(\mathcal{N}x@app(\mathcal{N}y@\mathcal{N}z)) = app(x@app(yz)) = app(x\,app(yz))$.

Lemma 36 *If $\lambda\mathcal{N}$ derives $\Gamma \vdash M : C \supset B$ and $\Gamma \vdash N : C$, then $\lambda\mathcal{N}$ derives $\Gamma \vdash M@N : B$.*

Proof: Let π and π_0 be derivations of $\Gamma \vdash M : C \supset B$ and $\Gamma \vdash N : C$, respectively. We prove by case analysis of M that there is a derivation π^* of $\Gamma \vdash M@N : B$.

Case $M = x$. Then $M@N = xN$ and π has the form

$$\frac{}{\Gamma', x : C \supset B \vdash x : C \supset B}\,Var$$

and $\Gamma = \Gamma', x : C \supset B$. We want a derivation of $\Gamma', x : C \supset B \vdash xN : B$ Take π^* as

$$\frac{\begin{array}{c}\pi_0\\ \vdots\\ \Gamma', x : C \supset B \vdash N : C\end{array}}{\Gamma', x : C \supset B \vdash xN : B}\,VElim$$

Case $M = \lambda x.M_0$. Then $M@N = (\lambda x.M_0)N$ and π has the form

$$\frac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma, x : C \vdash M_0 : B\end{array}}{\Gamma \vdash \lambda x.M_0 : C \supset B}\,Intro$$

We want a derivation of $\Gamma \vdash (\lambda x.M_0)N : B$. Take π^* as

$$\frac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma, x : C \vdash M_0 : B\end{array} \qquad \begin{array}{c}\pi_0\\ \vdots\\ \Gamma \vdash N : C\end{array}}{\Gamma \vdash (\lambda x.M_0)N : B}\,Redex$$

Case $M = app(A)$. Then $M@N = AN$ and π has the form

$$\frac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \vdash A : C \supset B\end{array}}{\Gamma \vdash app(A) : C \supset B}\,App$$

We want a derivation of $\Gamma \vdash AN : B$. Take π^* as

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \vdash A : C \supset B\end{array} \qquad \begin{array}{c}\pi_0\\ \vdots\\ \Gamma \vdash N : C\end{array}}{\Gamma \vdash AN : B}\ AElim$$

■

Proposition 18 (Correctness) *If λ derives $\Gamma \vdash M : B$ then $\lambda\mathcal{N}$ derives $\Gamma \vdash \mathcal{N}(M) : B$.*

Proof: Let π be a derivation of $\Gamma \vdash M : B$ in λ. One proves by induction on M that there is a derivation π^* of $\Gamma \vdash \mathcal{N}(M) : B$ in $\lambda\mathcal{N}$. Cases $M = x$ and $M = \lambda x.M_0$ are straightforward. Let $M = M_0N_0$. Then π as the form

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \vdash M_0 : C \supset B\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma \vdash N_0 : C\end{array}}{\Gamma \vdash M_0N_0 : B}\ Elim$$

Since $\mathcal{N}(M) = app(\mathcal{N}(M_0)@\mathcal{N}(N_0))$, take π^* as

$$\dfrac{\dfrac{\begin{array}{c}\pi_1^*\\ \vdots\\ \Gamma \vdash M_0^- : C \supset B\end{array} \qquad \begin{array}{c}\pi_2^*\\ \vdots\\ \Gamma \vdash N_0^- : C\end{array}}{\Gamma \vdash \mathcal{N}(M_0)@\mathcal{N}(N_0)}\ \text{Lemma 36}}{\Gamma \vdash app(\mathcal{N}(M_0)@\mathcal{N}(N_0))}\ App$$

where π_1^*, π_2^* are given by IH. ■

We now define, in Table 4.4, the inverse mapping, from $\lambda\mathcal{N}$ to λ. Mapping $|\_|$ (absolute value) sends the different kinds of application in $\lambda\mathcal{N}$ to application in λ and erases *app*. It is a forgetful mapping.

Proposition 19 (Correctness) *The following holds:*

1. *If $\lambda\mathcal{N}$ derives $\Gamma \vdash M : B$, then λ derives $\Gamma \vdash |M| : B$.*

$$
\begin{array}{rcl}
|x| & = & x \\
|\lambda x.M| & = & \lambda x.|M| \\
|app(A)| & = & |A| \\
\\
|xN| & = & x|N| \\
|(\lambda x.M)N| & = & (\lambda x.|M|)|N| \\
|AN| & = & |A||N|
\end{array}
$$

Table 4.4: From $\lambda\mathcal{N}$ to λ.

2. *If* $\lambda\mathcal{N}$ *derives* $\Gamma \vdash A : B$*, then* λ *derives* $\Gamma \vdash |A| : B$*.*

Proof: Let π_1 and π_2 be derivations in $\lambda\mathcal{N}$ of $\Gamma \vdash M : B$ and $\Gamma \vdash A : B$, respectively. We prove, by simultaneous induction on M and A (with induction hypothesis IH1 and IH2, respectively) that there are in λ derivations π_1^+ and π_2^+ of $\Gamma \vdash |M| : B$ and $\Gamma \vdash |A| : B$, respectively.

Case $M = x$. Immediate.

Case $M = \lambda x.M_0$. Immediate, by IH1.

Case $M = app(A)$. Then π_1 has the shape

$$
\dfrac{\begin{array}{c}\pi_2' \\ \vdots \\ \Gamma \vdash A : B\end{array}}{\Gamma \vdash app(A) : B}\,App
$$

Since $|M| = |A|$, we want a derivation π_1^+ of $\Gamma \vdash |A| : B$. By IH2, there is a derivation $\pi_2'^+$ of $\Gamma \vdash |A| : B$. Take $\pi_1^+ = \pi_2'^+$.

Case $A = xN$. Then π_2 has the shape

$$\dfrac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma', x : C \supset B \vdash N : C\end{array}}{\Gamma', x : C \supset B \vdash xN : B}\,VElim$$

where $\Gamma = \Gamma', x : C \supset B$. Since $|A| = x|N|$, we want a derivation π_2^+ of $\Gamma', x : C \supset B \vdash x|N| : B$. By IH1, there is a derivation $\pi_1'^+$ of $\Gamma', x : C \supset B \vdash |N| : C$. Take π_2^+ as

$$\dfrac{\dfrac{}{\Gamma, x : C \supset B \vdash x : C \supset B}\,Var \qquad \begin{array}{c}\pi_1'^+ \\ \vdots \\ \Gamma', x : C \supset B \vdash |N| : C\end{array}}{\Gamma', x : C \supset B \vdash x|N| : B}\,Elim$$

Case $A = (\lambda x.M)N$. Then π_2 has the shape

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma, x : C \vdash M : B\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma \vdash N : C\end{array}}{\Gamma \vdash (\lambda x.M)N : B}\,Redex$$

Since $|A| = (\lambda x.|M|)|N|$, we want a derivation π_2^+ of $\Gamma \vdash (\lambda x.|M|)|N| : B$. Using IH1 twice, we build π_2^+ as the traditional *Intro* followed by *Elim*.

Case $A = A_0N$. Follows easily by IH1 and IH2. ■

We now prove the isomorphism between $\lambda\mathcal{N}$ and λ at the level of terms.

Proposition 20 *$\mathcal{N}|M| = M$ and $\mathcal{N}|A| = app(A)$, for all M and A in $\lambda\mathcal{N}$.*

Proof: By simultaneous induction on M and A, with induction hypotheses IH1 and IH2, respectively.

Case $M = x$. $\mathcal{N}|M| = \mathcal{N}|x| = \mathcal{N}(x) = x = M$.

Case $M = \lambda x.M_0$.

$$\begin{array}{rcl}\mathcal{N}|M| & = & \mathcal{N}|\lambda x.M_0| \\ & = & \mathcal{N}(\lambda x.|M_0|),\text{ by def. of } |\_|,\end{array}$$

$$
\begin{aligned}
&= \lambda x.\mathcal{N}|M_0|, \text{ by def. of } \mathcal{N}, \\
&= \lambda x.M_0, \text{ by IH1}, \\
&= M \ .
\end{aligned}
$$

Case $M = app(A)$.

$$
\begin{aligned}
\mathcal{N}|M| &= \mathcal{N}|app(A)| \\
&= \mathcal{N}|A|, \text{ by def. } |\_|, \\
&= app(A), \text{ by IH2}, \\
&= M \ .
\end{aligned}
$$

Case $A = xN$.

$$
\begin{aligned}
\mathcal{N}|A| &= \mathcal{N}|xN| \\
&= \mathcal{N}(x|N|), \text{ by def. of } |\_|, \\
&= app(\mathcal{N}(x)@\mathcal{N}|N|), \text{ by def. of } \mathcal{N}, \\
&= app(x@\mathcal{N}|N|), \text{ by def. of } \mathcal{N}, \\
&= app(x@N), \text{ by IH1}, \\
&= app(xN), \text{ by def. of } @, \\
&= app(A) \ .
\end{aligned}
$$

Case $A = (\lambda x.M)N$.

$$
\begin{aligned}
\mathcal{N}|A| &= \mathcal{N}|(\lambda x.M)N| \\
&= \mathcal{N}((\lambda x.|M|)|N|), \text{ by def. of } |\_|, \\
&= app(\mathcal{N}(\lambda x.|M|)@\mathcal{N}|N|), \text{ by def. of } \mathcal{N}, \\
&= app((\lambda.\mathcal{N}|M|)@\mathcal{N}|N|), \text{ by def. of } \mathcal{N}, \\
&= app((\lambda x.M)@N), \text{ by IH1}, \\
&= app((\lambda x.M)N), \text{ by def. of } @, \\
&= app(A) \ .
\end{aligned}
$$

Case $A = A_0N_0$.

$$
\mathcal{N}|A| = \mathcal{N}|A_0N_0|
$$

$$
\begin{aligned}
&= \mathcal{N}(|A_0||N_0|), \text{ by def. of } |\_|,\\
&= app(\mathcal{N}|A_0|@\mathcal{N}|N_0|), \text{ by def. of } \mathcal{N},\\
&= app(app(A_0)@N_0), \text{ by IH1 and IH2},\\
&= app(A_0N_0), \text{ by def. of } @,\\
&= app(A)\ .
\end{aligned}
$$

■

Lemma 37 $|M@N| = |M||N|$, *for all* M, N *in* $\lambda\mathcal{N}$.

Proof: By case analysis of M. Variables and λ-abstractions are uninteresting. If $M = app(A)$ then

$$
\begin{aligned}
|M@N| &= |app(A)@N|\\
&= |AN|, \text{ by def. of } @,\\
&= |A||N|, \text{ by def. of } |\_|,\\
&= |app(A)||N|, \text{ by def. of } |\_|,\\
&= |M||N|.
\end{aligned}
$$

■

Proposition 21 $|\mathcal{N}(M)| = M$, *for all* M *in* λ.

Proof: By induction on M. Variables and λ-abstractions are straightforward. If $M = M_0N_0$, then

$$
\begin{aligned}
|\mathcal{N}(M)| &= |\mathcal{N}(M_0N_0)|\\
&= |app(\mathcal{N}(M_0)@\mathcal{N}(N_0))|, \text{ by def. of } \mathcal{N},\\
&= |(\mathcal{N}(M_0)@\mathcal{N}(N_0))|, \text{ by def of } |\_|,
\end{aligned}
$$

$$\begin{aligned} &= |\mathcal{N}(M_0)||\mathcal{N}(N_0)|, \text{ by Lemma 37,} \\ &= M_0N_0, \text{ by I.H.,} \\ &= M \ . \end{aligned}$$

■

Now it comes the proof of the isomorphism at the level of reduction.

Lemma 38 *The following holds:*

1. $|M[N/x]| = |M|[|N|/x]$, *for all* M, N *in* $\lambda\mathcal{N}$.
2. $|A[N/x]| = |A|[|N|/x]$, *for all* A, N *in* $\lambda\mathcal{N}$.

Proof: By simultaneous induction on M and A, with induction hypotheses IH1 and IH2, respectively . There are only two interesting cases.

Case $M = app(A)$. Then

$$\begin{aligned} |M[N/x]| &= |app(A)[N/x]| \\ &= |app(A[N/x])|, \text{ by def. of } \_[N/x] \text{ in } \lambda\mathcal{N}, \\ &= |A[N/x]|, \text{ by def. of } |\_|, \\ &= |A|[|N|/x], \text{ by IH2} \\ &= |app(A)|[|N|/x], \text{ by def. of } |\_|, \\ &= |M|[|N|/x] \ . \end{aligned}$$

Case $A = xM$. Then,

$$\begin{aligned} |A[N/x]| &= |(xM)[N/x]| \\ &= |N@M[N/x]|, \text{ by def. of } \_[N/x], \\ &= |N||M[N/x]|, \text{ by Lemma 37,} \\ &= |N|(|M|[|N|/x]), \text{ by IH1,} \end{aligned}$$

$$
\begin{aligned}
&= (x|M|)[|N|/x], \text{ by def. of } \_[N/x] \text{ in } \lambda, \\
&= |xM|[|N|/x], \text{ by def. of } |\_|, \\
&= |A|[|N|/x] \; .
\end{aligned}
$$

■

Corollary 14 *$\mathcal{N}(M[N/x]) = \mathcal{N}(M)[\mathcal{N}(N)/x]$, for all M, N in λ.*

Proof:

$$
\begin{aligned}
\mathcal{N}(M[N/x]) &= \mathcal{N}(|\mathcal{N}(M)|[|\mathcal{N}(N)|/x]), \text{ by Proposition 21,} \\
&= \mathcal{N}|\mathcal{N}(M)[\mathcal{N}(N)/x]|, \text{ by Lemma 38,} \\
&= \mathcal{N}(M)[\mathcal{N}(N)/x], \text{ by Proposition 20.}
\end{aligned}
$$

■

The following is the first half of the isomorphism.

Theorem 2 *If $M_1 \rightarrow_\beta M_2$ in $\lambda\mathcal{N}$, then $|M_1| \rightarrow_\beta |M_2|$ in λ.*

Proof: The claim is proved together with the claim that if $A_1 \rightarrow_\beta A_2$ in $\lambda\mathcal{N}$, then $|A_1| \rightarrow_\beta |A_2|$ in λ, by simultaneous induction on $M_1 \rightarrow_\beta M_2$ and $A_1 \rightarrow_\beta A_2$ (with induction hypotheses IH1 and IH2, respectively). Cases according to Definition 9.

Case $\beta 1$:

$$
\begin{aligned}
|app((\lambda x.M)N)| &= (\lambda x.|M|)|N|, \text{ by def. of } |\_|, \\
&\rightarrow_\beta |M|[|N|/x] \\
&= |M[N/x]|, \text{ by Lemma 38.}
\end{aligned}
$$

Case $\beta 2$:

$$\begin{array}{rcl} |((\lambda x.M)N_1)N_2| & = & ((\lambda x.|M|)|N_1|)|N_2|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & |M|[|N_1|/x]|N_2| \\ & = & |M[N_1/x]||N_2|, \text{ by Lemma 38}, \\ & = & |M[N_1/x]@N_2|, \text{ by Lemma 37}. \end{array}$$

Case *Intro*: Suppose $|M_1| \rightarrow_\beta |M_2|$ (IH1). Then,

$$\begin{array}{rcl} |\lambda x.M_1| & = & \lambda x.|M_1|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & \lambda x.|M_2| \ (*) \\ & = & |\lambda x.M_2|, \text{ by def. of } |\_|, \end{array}$$

where step (*) is by IH1 and closure of $\rightarrow_\beta$ in λ under *Intro*.

Case *App*: Suppose $|A_1| \rightarrow_\beta |A_2|$ (IH2). Then,

$$\begin{array}{rcl} |app(A_1)| & = & |A_1|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & |A_2|, \text{ by IH2}, \\ & = & |app(A_2)|, \text{ by def. of } |\_|. \end{array}$$

Case *VElim*: Suppose $|N_1| \rightarrow_\beta |N_2|$ (IH1). Then,

$$\begin{array}{rcl} |xN_1| & = & x|N_1|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & x|N_2| \ (*) \\ & = & |xN_2|, \text{ by def. of } |\_|, \end{array}$$

where step (*) is by IH1 and closure of $\rightarrow_\beta$ in λ under *Elim*2.

Case *Redex*1: Suppose $|M_1| \rightarrow_\beta |M_2|$ (IH1). Then,

$$\begin{array}{rcl} |(\lambda x.M_1)N| & = & (\lambda x.|M_1|)|N|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & (\lambda x.|M_2|)|N| \ (*) \\ & = & |((\lambda x.M_2)N|, \text{ by def. of } |\_|, \end{array}$$

where step (*) is by IH1 and closure of $\rightarrow_\beta$ in λ under *Intro* and *Elim*1.

Case *Redex*2: Similarly, by IH1 and closure of $\rightarrow_\beta$ in λ under *Elim*2.

Case *AElim*1: Suppose $|A_1| \rightarrow_\beta |A_2|$ (IH2). Then,

$$\begin{array}{rcl} |A_1 N| & = & |A_1||N|, \text{ by def. of } |\_|, \\ & \rightarrow_\beta & |A_2||N| \ (*) \\ & = & |A_2 N|, \text{ by def. of } |\_|, \end{array}$$

where step (*) is by IH2 and closure of $\rightarrow_\beta$ in λ under *Elim*1.

Case *AElim*2: Similarly, by IH1 and closure of $\rightarrow_\beta$ in λ under *Elim*2. ■

Lemma 39 *The following holds in $\lambda\mathcal{N}$:*

1. *If $M \rightarrow_\beta M'$, then $M@N \rightarrow_\beta M'@N$.*

2. *If $N \rightarrow_\beta N'$, then $M@N \rightarrow_\beta M@N'$.*

Proof: 1. Suppose $M \rightarrow_\beta M'$. We proceed by case analysis of M.

Case $M = x$. Vacuous.

Case $M = \lambda x.M_0$. Hence there is M_0' such that $M' = \lambda x.M_0'$ and $M_0 \rightarrow_\beta M_0'$. Then, $M@N = (\lambda x.M_0)@N = (\lambda x.M_0)N \rightarrow_\beta (\lambda x.M_0')N = (\lambda x.M_0')@N = M'@N$ (here we used the fact that $\rightarrow_\beta$ in $\lambda\mathcal{N}$ is closed under *Redex*1).

Case $M = app(A)$. There are two subcases.

Subcase 1: $A \rightarrow_\beta A'$ and $M' = app(A')$. Then $M@N = app(A)@N = AN \rightarrow_\beta A'N = app(A')@N = M'@N$ (here we used the fact that $\rightarrow_\beta$ in $\lambda\mathcal{N}$ is closed under *AElim*1).

Subcase 2: $A = (\lambda x.M_0)N_0$ and $M' = M_0[N_0/x]$. Then, $M@N = app(A)@N = AN = ((\lambda x.M_0)N_0)N \rightarrow_\beta M_0[N_0/x]@N = M'@N$.

2. Suppose $N \rightarrow_\beta N'$. We proceed by case analysis of M.

Case $M = x$. Then, $M@N = x@N = xN \rightarrow_\beta xN' = x@N' = M@N'$ (here we used the fact that $\rightarrow_\beta$ in $\lambda\mathcal{N}$ is closed under *VElim*).

Case $M = \lambda x.M_0$. Then, $M@N = (\lambda x.M_0)@N = (\lambda x.M_0)N \rightarrow_\beta (\lambda x.M_0)N' = (\lambda x.M_0)@N' = M@N'$ (here we used the fact that $\rightarrow_\beta$ in $\lambda\mathcal{N}$ is closed under *Redex*2).

Case $M = app(A)$. Then, $M@N = app(A)@N = AN \rightarrow_\beta AN' = app(A)@N' = M@N'$ (here we used the fact that $\rightarrow_\beta$ in $\lambda\mathcal{N}$ is closed under *AElim*2). ■

The second half of the isomorphism is:

Theorem 3 *If $M_1 \rightarrow_\beta M_2$ in λ, then $\mathcal{N}(M_1) \rightarrow_\beta \mathcal{N}(M_2)$ in $\lambda\mathcal{N}$.*

Proof: By induction on $M_1 \rightarrow_\beta M_2$. Cases according to Definition 1.

Case β:

$$\begin{array}{rcl} \mathcal{N}(\lambda x.M)N & = & app((\lambda x.\mathcal{N}(M))@\mathcal{N}(N)), \text{ by def. of } \mathcal{N}, \\ & = & app((\lambda x.\mathcal{N}(M))\mathcal{N}(N)), \text{ by def. of } @, \\ & \rightarrow_\beta & \mathcal{N}(M)[\mathcal{N}(N)/x] \\ & = & \mathcal{N}(M[N/x]), \text{ by Corollary 14.} \end{array}$$

Case *Intro*: suppose $\mathcal{N}(M_1) \rightarrow_\beta \mathcal{N}(M_2)$ (IH). Then,

$$\begin{array}{rcl} \mathcal{N}(\lambda x.M_1) & = & \lambda x.\mathcal{N}(M_1), \text{ by def. of } \mathcal{N}, \\ & \rightarrow_\beta & \lambda x.\mathcal{N}(M_2) \ (*) \\ & = & \mathcal{N}(\lambda x.M_2), \text{ by def. of } \mathcal{N}, \end{array}$$

where step (*) is by IH and closure of $\rightarrow_\beta$ in $\lambda\mathcal{N}$ under *Intro*.

Case *Elim*1: suppose $\mathcal{N}(M_1) \rightarrow_\beta \mathcal{N}(M_2)$ (IH). Then,

$$\begin{array}{rcl} \mathcal{N}(M_1 N) & = & app(\mathcal{N}(M_1)@\mathcal{N}(N)), \text{ by def. of } \mathcal{N}, \\ & \rightarrow_\beta & app(\mathcal{N}(M_2)@\mathcal{N}(N)) \ (*) \\ & = & \mathcal{N}(M_2 N), \text{ by def. of } \mathcal{N}, \end{array}$$

where step (*) is by IH and 1. of Lemma 39 and closure of $\to_\beta$ in $\lambda\mathcal{N}$ under *App*.

Case *Elim*2: Similarly, by IH and 2. of Lemma 39 and closure of $\to_\beta$ in $\lambda\mathcal{N}$ under *App*. ■

Corollary 15 (Isomorphism)

1. *$M_1 \to_\beta M_2$ in λ iff $\mathcal{N}(M_1) \to_\beta \mathcal{N}(M_2)$ in $\lambda\mathcal{N}$.*

2. *$M_1 \to_\beta M_2$ in $\lambda\mathcal{N}$ iff $|M_1| \to_\beta |M_2|$ in λ.*

Corollary 16

1. *$\lambda\mathcal{N}$ is confluent.*

2. *If M is typable in $\lambda\mathcal{N}$, then M is strongly normalising.*

3. *$\lambda\mathcal{N}$ satisfies subject reduction.*

Proof: Because these properties hold of λ and may be easily transferred from λ to $\lambda\mathcal{N}$ with the help of $\mathcal{N}$ and $|\_|$. ■

4.2 Mappings Ψ and Θ

Translations Ψ and Θ between $\lambda\mathcal{N}$ and $\lambda\mathcal{P}$ are given in Tables 4.5 and 4.6. The idea of Ψ (recall Chapter 2) is to "turn the main branch upside down". Roughly, if $\Psi(V) = v$ (where V is some value) and $\Psi(N_i) = u_i$, then Ψ sends $app(VN_1N_2...N_k)$ to $v(u_1 \cdot [u_2, ..., u_k])$. Θ does precisely the inverse.

The following propositions were firstly proved for the cut-free fragment in [Dyckhoff and Pinto, 1998].

Proposition 22 (Correctness of Ψ)

1. *If $\lambda\mathcal{N}$ derives $\Gamma \vdash M : B$ then $\lambda\mathcal{P}$ derives $\Gamma; - \vdash \Psi(M) : B$.*

$$
\begin{array}{rcl}
\Psi(x) & = & x \\
\Psi(\lambda x.M) & = & \lambda x.\Psi M \\
\Psi(app(A)) & = & \Psi'(A, []) \\
\\
\Psi'(xN, l) & = & x(\Psi N \cdot l) \\
\Psi'((\lambda x.M)N, l) & = & (\lambda x.\Psi M)(\Psi N \cdot l) \\
\Psi'(AN, l) & = & \Psi'(A, \Psi N :: l)
\end{array}
$$

Table 4.5: From $\lambda\mathcal{N}$ to $\lambda\mathcal{P}$

$$
\begin{array}{rcl}
\Theta(x) & = & x \\
\Theta(x(u \cdot l)) & = & \Theta'(x\Theta u, l) \\
\Theta(\lambda x.t) & = & \lambda x.\Theta t \\
\Theta((\lambda x.t)(u \cdot l)) & = & \Theta'((\lambda x.\Theta t)\Theta u, l) \\
\\
\Theta'(A, []) & = & app(A) \\
\Theta'(A, u :: l) & = & \Theta'(A\Theta u, l)
\end{array}
$$

Table 4.6: From $\lambda\mathcal{P}$ to $\lambda\mathcal{N}$.

2. *If $\lambda\mathcal{N}$ derives $\Gamma \vdash A : C$ and $\lambda\mathcal{P}$ derives $\Gamma; C \vdash l : B$ then $\lambda\mathcal{P}$ derives $\Gamma; - \vdash \Psi'(A, l) : B$.*

Proof: We prove by simultaneous induction on M and A (with induction hypotheses IH1 and IH2, respectively) that: i) if π_1 is a derivation in $\lambda\mathcal{N}$ of $\Gamma \vdash M : B$, then there is in $\lambda\mathcal{P}$ a derivation π_1^* of $\Gamma; - \vdash \Psi(M) : B$; and ii) if π_2 is a derivation in $\lambda\mathcal{N}$ of $\Gamma \vdash A : C$, then, for all l such that $\lambda\mathcal{P}$ derives $\Gamma; C \vdash l : B$, there is in $\lambda\mathcal{P}$ a derivation π_2^* of $\Gamma; - \vdash \Psi'(A, l) : B$.

Case $M = x$: Then there is Γ' such that π_1 has the form

$$\frac{}{\Gamma', x : B \vdash x : B}\ Var$$

and $\Gamma = \Gamma', x : B$. Since $\Psi(M) = \Psi(x) = x$, we want a derivation π_1^* of $\Gamma', x : B; - \vdash x : B$. Take π_1^* as one application of the Var rule.

Case $M = \lambda x.M'$: Then there are π_1', B_1, B_2 such that π_1 has the form

$$\frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : B_1 \vdash M' : B_2\end{array}}{\Gamma \vdash \lambda x.M' : B_1 \supset B_2}\ Intro\ ,$$

$B = B_1 \supset B_2$ and $x \notin \Gamma$. Since $\Psi(M) = \Psi(\lambda x.M') = \lambda x.\Psi(M')$, we want a derivation π_1^* of $\Gamma; - \vdash \lambda x.M' : B_1 \supset B_2$. Take π_1^* as

$$\frac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma, x : B_1; - \vdash \Psi(M') : B_2\end{array}}{\Gamma; - \vdash \lambda x.M' : B_1 \supset B_2}\ Right$$

where π_1^+ is given by IH1.

Case $M = app(A)$: Then there is π_2' such that π_1 has the form

$$\frac{\begin{array}{c}\pi_2 \\ \vdots \\ \Gamma \vdash A : B\end{array}}{\Gamma \vdash app(A) : B}\ App$$

Since $\Psi(M) = \Psi(app(A)) = \Psi'(A, [])$ we want a derivation π_1^* of $\Gamma; - \vdash \Psi'(A, []) : B$. Now $\lambda\mathcal{P}$ derives $\Gamma; B \vdash [] : B$ and, hence, by IH2, there is a derivation π_2^+ of $\Gamma; - \vdash \Psi'(A, []) : B$. Take $\pi_1^* = \pi_2^+$.

Case $A = xN$: The there are π_1', Γ', D such that π_2 as the form

$$\dfrac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma', x : D \supset C \vdash N : D\end{array}}{\Gamma', x : D \supset C \vdash xN : C}\ VElim$$

and $\Gamma = \Gamma', x : D \supset C$. Let π_3 be a derivation in $\lambda\mathcal{P}$ of $\Gamma; C \vdash l : B$. Since $\Psi'(A, l) = \Psi'(xN, l) = x(\Psi N \cdot l)$, we want a derivation π_2^* of $\Gamma', x : D \supset C; - \vdash x(\Psi N \cdot l) : B$. Take π_2^* as

$$\dfrac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma', x : D \supset C; - \vdash \Psi N : D\end{array} \qquad \begin{array}{c}\pi_3 \\ \vdots \\ \Gamma; C \vdash l : B\end{array}}{\Gamma', x : D \supset C; - \vdash x(\Psi N \cdot l) : B}\ Left$$

where π_1^+ is given by IH1.

Case $A = (\lambda x.M)N$: Then there are π_1', π_1'', D such that π_2 has the form

$$\dfrac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma, x : D \vdash M : C\end{array} \qquad \begin{array}{c}\pi_1'' \\ \vdots \\ \Gamma \vdash N : D\end{array}}{\Gamma \vdash (\lambda x.M)N : C}\ Redex$$

and $x \notin \Gamma$. Let π_3 be a derivation in $\lambda\mathcal{P}$ of $\Gamma; C \vdash l : B$. Since $\Psi'(A, l) = \Psi'((\lambda x.M)N, l) = (\lambda x.\Psi M)(\Psi N \cdot l)$, we want a derivation π_2^* of $\Gamma; - \vdash (\lambda x.\Psi M)(\Psi N \cdot l) : B$. Take π_2^* as

$$\dfrac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma, x : D; - \vdash \Psi(M) : C\end{array} \qquad \begin{array}{c}\pi_1^{++} \\ \vdots \\ \Gamma; - \vdash \Psi(N) : D\end{array} \qquad \begin{array}{c}\pi_3 \\ \vdots \\ \Gamma; C \vdash l : B\end{array}}{\Gamma; - \vdash (\lambda x.\Psi M)(\Psi N \cdot l) : B}\ KeyCut$$

where π_1^+ and π_1^{++} are given by IH1.

Case $A = A'N$: Then there are π_1', π_2', D such that π_2 has the form

$$\dfrac{\begin{array}{c}\pi_2' \\ \vdots \\ \Gamma \vdash A' : D \supset C\end{array} \qquad \begin{array}{c}\pi_1' \\ \vdots \\ \Gamma \vdash N : D\end{array}}{\Gamma \vdash A'N : C}\ AElim$$

Let π_3 be a derivation in $\lambda\mathcal{P}$ of $\Gamma; C \vdash l : B$. Since $\Psi'(A, l) = \Psi'(A'N, l) = \Psi'(A, \Psi N :: l)$, we want a derivation π_2^* of $\Gamma; - \vdash \Psi'(A, \Psi N :: l) : B$. Observe that

$$\dfrac{\begin{array}{c}\pi_1^+\\ \vdots\\ \Gamma; - \vdash \Psi N : D\end{array} \qquad \begin{array}{c}\pi_3\\ \vdots\\ \Gamma; C \vdash l : B\end{array}}{\Gamma; D \supset C \vdash \Psi N :: l : B}\, Lft$$

is a derivation in $\lambda\mathcal{P}$ of $\Gamma; D \supset C \vdash \Psi N :: l : B$, where π_1^+ is given by IH1. Hence, by IH2, there is a derivation π_2^+ of $\Gamma; - \vdash \Psi'(A, \Psi N :: l) : B$. Take $\pi_2^* = \pi_2^+$. ■

Proposition 23 (Correctness of Θ)

1. *If $\lambda\mathcal{P}$ derives $\Gamma; - \vdash t : B$ then $\lambda\mathcal{N}$ derives $\Gamma \vdash \Theta t : B$.*

2. *If $\lambda\mathcal{N}$ derives $\Gamma \vdash A : C$ and $\lambda\mathcal{P}$ derives $\Gamma; C \vdash l : B$ then $\lambda\mathcal{N}$ derives $\Gamma \vdash \Theta'(A, l) : B$.*

Proof: We prove by simultaneous induction on t and l (with induction hypotheses referred to by IH1 and IH2, respectively) that i) if π_1 is a derivation in $\lambda\mathcal{P}$ of $\Gamma; - \vdash t : B$, then there is in $\lambda\mathcal{N}$ a derivation π_1^* of $\Gamma \vdash \Theta t : B$; and ii) if π_2 is a derivation in $\lambda\mathcal{P}$ of $\Gamma; C \vdash l : B$, then for all A such that $\lambda\mathcal{N}$ derives $\Gamma \vdash A : C$, there is in $\lambda\mathcal{N}$ a derivation π_2^* of $\Gamma \vdash \Theta'(A, l) : B$.

Case $l = []$: Then $B = C$. Let π_3 be a derivation of $\Gamma \vdash A : C$. Since $\Theta'(A, l) = \Theta'(A, []) = app(A)$, we want a derivation π_2^* of $\Gamma \vdash app(A) : B$. Take π_2^* as

$$App\, \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ \Gamma \vdash A : B\end{array}}{\Gamma \vdash app(A) : B}\ .$$

Case $l = u' :: l'$: Then there are π_1', π_2', C_1, C_2 such that π_2 has the form

$$Lft \frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma; - \vdash u' : C_1\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma; C_2 \vdash l' : B\end{array}}{\Gamma; C_1 \supset C_2 \vdash u' :: l' : B}$$

and $C = C_1 \supset C_2$. Let π_3 be a derivation of $\Gamma \vdash A : C_1 \supset C_2$. Since $\Theta'(A, l) = \Theta'(A, u' :: l') = \Theta'(A\Theta u', l')$, we want a derivation π_2^* of $\Gamma \vdash \Theta'(A\Theta u', l') : B$. Observe that

$$AElim \frac{\begin{array}{c}\pi_3 \\ \vdots \\ \Gamma \vdash A : C_1 \supset C_2\end{array} \qquad \begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma \vdash \Theta u' : C_1\end{array}}{\Gamma \vdash A\Theta u' : C_2}$$

is a derivation in $\lambda\mathcal{N}$ of $\Gamma \vdash A\Theta u' : C_2$, where π_1^+ is given by IH1. Hence, by IH2, there is a derivation π_2^+ of $\Gamma \vdash \Theta'(A\Theta u', l') : B$. Take $\pi_2^* = \pi_2^+$.

Case $t = x$: Then there is Γ' such that π_1 has the form

$$Var \frac{}{\Gamma', x : B; - \vdash x : B}$$

and $\Gamma = \Gamma', x : B$. Since $\Theta t = \Theta(x) = x$, we want a derivation π_1^* of $\Gamma', x : B \vdash x : B$. Just take π_1^* as an application of the Var rule.

Case $t = x(u \cdot l)$: Then there are $\pi_1', \pi_2', \Gamma', D$ such that π has the form

$$Left \frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma', x : D \supset C; - \vdash u : D\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma', x : D \supset C; C \vdash l : B\end{array}}{\Gamma', x : D \supset C; - \vdash x(u \cdot l) : B} \; .$$

and $\Gamma = \Gamma', x : D \supset C$. Since $\Theta t = \Theta(x(u \cdot l)) = \Theta'(x\Theta u, l)$, we want a derivation π_1^* of $\Gamma', x : D \supset C \vdash \Theta'(x\Theta u, l) : B$. Observe that

$$VElim \frac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma', x : D \supset C \vdash \Theta u : D\end{array}}{\Gamma', x : D \supset C \vdash x\Theta u : C}$$

is a derivation in $\lambda\mathcal{N}$ of $\Gamma', x : D \supset C \vdash x\Theta u : C$, where π_1^+ is given by IH1. By IH2, there is a derivation π_2^+ of $\Gamma', x : D \supset C \vdash \Theta'(x\Theta u, l) : B$. Take $\pi_1^* = \pi_2^+$.

Case $t = \lambda x.t'$: Then there are π_1', B_1, B_2 such that π_1 has the form

$$\begin{array}{c} \pi_1' \\ \vdots \\ \dfrac{\Gamma, x : B_1; - \vdash t' : B_2}{\Gamma; - \vdash \lambda x.t' : B_1 \supset B_2}\,\mathit{Right} \end{array}$$

and $B = B_1 \supset B_2$. Since $\Theta t = \Theta(\lambda x.t') = \lambda x.\Theta t'$, we want a derivation π_1^* of $\Gamma \vdash \lambda x.\Theta t' : B_1 \supset B_2$. Take π_1^* as

$$\begin{array}{c} \pi_1^+ \\ \vdots \\ \dfrac{\Gamma, x : B_1 \vdash \Theta t' : B_2}{\Gamma \vdash \lambda x.\Theta t' : B_1 \supset B_2}\,\mathit{Intro} \end{array}$$

where π_1^+ is given by IH1.

Case $t = (\lambda x.t')(u' \cdot l')$: Then there are $\pi_1', \pi_1'', \pi_2', D, E$ such that π_1 has the form

$$\mathit{KeyCut}\,\dfrac{\begin{array}{c} \pi_1' \\ \vdots \\ \Gamma, x : D; - \vdash t' : E \end{array} \qquad \begin{array}{c} \pi_1'' \\ \vdots \\ \Gamma; - \vdash u' : D \end{array} \qquad \begin{array}{c} \pi_2' \\ \vdots \\ \Gamma; E \vdash l' : B \end{array}}{\Gamma; - \vdash (\lambda x.t')(u' \cdot l') : B}$$

and $x \notin \Gamma$. Since $\Theta t = \Theta((\lambda x.t')(u'cdotl')) = \Theta'((\lambda x.\Theta t')\Theta u', l')$, we want a derivation π_1^* of $\Gamma \vdash \Theta'((\lambda x.\Theta t')\Theta u', l') : B$. Observe that

$$\mathit{Redex}\,\dfrac{\begin{array}{c} \pi_1^+ \\ \vdots \\ \Gamma, x : D \vdash \Theta t' : E \end{array} \qquad \begin{array}{c} \pi_1^{++} \\ \vdots \\ \Gamma \vdash \Theta u' : D \end{array}}{\Gamma \vdash (\lambda x.\Theta t')\Theta u' : E}$$

is a derivation in $\lambda\mathcal{N}$ of $\Gamma \vdash (\lambda x.\Theta t')\Theta u' : E$, where π_1^+ and π_1^{++} are given by IH1. Hence, by IH2, there is a derivation π_2^+ of $\Gamma \vdash \Theta'((\lambda x.\Theta t')\Theta u', l') : B$. Take $\pi_1^* = \pi_2^+$. ■

4.3 The isomorphism theorem

We prove that Ψ and Θ are mutually inverse. This establishes the isomorphism between $\lambda\mathcal{P}$ and $\lambda\mathcal{N}$ at the level of proofs. The method is taken from [Dyckhoff and Pinto, 1998].

Proposition 24 *$\Theta \circ \Psi = id$ and $\Theta \circ \Psi' = \Theta'$.*

Proof: We prove $\Theta\Psi M = M$ and $\Theta\Psi'(A, l) = \Theta'(A, l)$ by simultaneous induction on M (respec. A) with induction hypothesis IH1 (respec. IH2).

Cases $M = x$ and $M = \lambda x.M'$ are straightforward.

Case $M = app(A)$:

$$\begin{aligned} \Theta\Psi M &= \Theta\Psi(app(A)) \\ &= \Theta\Psi'(A, []) \\ &= \Theta'(A, []), \text{ by IH2,} \\ &= app(A) \ . \end{aligned}$$

Case $A = xM$:

$$\begin{aligned} \Theta\Psi'(A, l) &= \Theta\Psi'(xM, l) \\ &= \Theta(x(\Psi M \cdot l)) \\ &= \Theta'(x(\Theta\Psi M), l) \\ &= \Theta'(xM, l), \text{ by IH1.} \end{aligned}$$

Case $A = (\lambda x.M)N$:

$$\begin{aligned} \Theta\Psi'(A, l) &= \Theta\Psi'((\lambda x.M)N, l) \\ &= \Theta((\lambda x.\Psi M)(\Psi N \cdot l)) \\ &= \Theta'((\lambda x.\Theta\Psi M)(\Theta\Psi N), l) \\ &= \Theta'((\lambda x.M)N, l), \text{ by IH1.} \end{aligned}$$

Case $A = A'M$:

$$\begin{array}{rcl} \Theta\Psi'(A,l) & = & \Theta\Psi'(A'M,l) \\ & = & \Theta\Psi'(A',\Psi M :: l) \\ & = & \Theta'(A',\Psi M :: l),\ \text{by IH2,} \\ & = & \Theta'(A'(\Theta\Psi M),l) \\ & = & \Theta'(A'M,l),\ \text{by IH1,} \\ & = & \Theta'(A,l)\ \ . \end{array}$$

■

Proposition 25 *$\Psi \circ \Theta = id$ and $\Psi \circ \Theta' = \Psi'$.*

Proof: We prove $\Psi\Theta t = t$ and $\Psi\Theta'(A,l) = \Psi'(A,l)$ by simultaneous induction on t (respec. l) with induction hypothesis IH1 (respec. IH2).

Cases $t = x$ and $t = \lambda x.t'$ are straightforward.

Case $t = x(u \cdot l)$:

$$\begin{array}{rcl} \Psi\Theta t & = & \Psi\Theta(x(u \cdot l)) \\ & = & \Psi\Theta'(x\Theta u, l) \\ & = & \Psi'(x\Theta u, l),\ \text{by IH2,} \\ & = & x(\Psi\Theta u \cdot l) \\ & = & x(u \cdot l),\ \text{by IH1,} \\ & = & t\ \ . \end{array}$$

Case $t = (\lambda x.t')(u \cdot l)$:

$$\begin{array}{rcl} \Psi\Theta t & = & \Psi\Theta((\lambda x.t')(u \cdot l)) \\ & = & \Psi\Theta'((\lambda x.\Theta t)\Theta u, l) \\ & = & \Psi'((\lambda x.\Theta t)\Theta u, l),\ \text{by IH2,} \end{array}$$

$$\begin{aligned}
&= (\lambda x.\Psi\Theta t)(\Psi\Theta u \cdot l) \\
&= (\lambda x.t')(u \cdot l), \text{ by IH1,} \\
&= t \ .
\end{aligned}$$

Case $l = []$:

$$\begin{aligned}
\Psi\Theta'(A, l) &= \Psi\Theta'(A, []) \\
&= \Psi(app(A)) \\
&= \Psi'(A, []) \\
&= \Psi'(A, l) \ .
\end{aligned}$$

Case $l = u :: l'$:

$$\begin{aligned}
\Psi\Theta'(A, l) &= \Psi\Theta'(A, u :: l') \\
&= \Psi\Theta'(A\Theta u, l) \\
&= \Psi'(A\Theta u, l), \text{ by IH2,} \\
&= \Psi'(A, \Psi\Theta u :: l) \\
&= \Psi'(A, u :: l), \text{ by IH1.}
\end{aligned}$$

■

Now we establish some preliminary properties that will be useful later in proving the isomorphism between the normalisation procedures in $\lambda\mathcal{N}$ and $\lambda\mathcal{P}$. The first results relate *insert* and *append* in $\lambda\mathcal{P}$ with @ in $\lambda\mathcal{N}$.

Lemma 40

1. $insert(\Psi N, l, \Psi M) = \Psi'(M@N, l)$, *for all* M, N *in* $\lambda\mathcal{N}$ *and* l *in* $\lambda\mathcal{P}$.

2. $insert(u', l', \Psi'(A, l)) = \Psi'(A, append(l, u' :: l'))$, *for all* A *in* $\lambda\mathcal{N}$ *and* u', l, l' *in* $\lambda\mathcal{P}$.

Proof: By a simultaneous induction on M and A, with induction hypotheses IH1 and IH2, respectively.

Case $M = x$.

$$\begin{aligned}
insert(\Psi N, l, \Psi M) &= insert(\Psi N, l, \Psi x) \\
&= insert(\Psi N, l, x) \\
&= x(\Psi N \cdot l) \\
&= \Psi'(xN, l) \\
&= \Psi'(x@N, l) \\
&= \Psi'(M@N, l) \ .
\end{aligned}$$

Case $M = \lambda x.M'$.

$$\begin{aligned}
insert(\Psi N, l, \Psi M) &= insert(\Psi N, l, \Psi(\lambda x.M')) \\
&= insert(\Psi N, l, \lambda x.\Psi M') \\
&= (\lambda x.\Psi M')(\Psi N \cdot l) \\
&= \Psi'((\lambda x.M')N, l) \\
&= \Psi'((\lambda x.M')@N, l) \\
&= \Psi'(M@N, l) \ .
\end{aligned}$$

Case $M = app(A)$.

$$\begin{aligned}
insert(\Psi N, l, \Psi M) &= insert(\Psi N, l, \Psi(app(A))) \\
&= insert(\Psi N, l, \Psi'(A, [])) \\
&= \Psi'(A, append([], \Psi N :: l)), \text{ by IH2,} \\
&= \Psi'(A, \Psi N :: l) \\
&= \Psi'(AN, l) \\
&= \Psi'(app(A)@N, l) \\
&= \Psi'(M@N, l) \ .
\end{aligned}$$

Case $A = xM$:

$$\begin{aligned}
insert(u', l', \Psi'(A, l)) &= insert(u', l', \Psi'(xM, l)) \\
&= insert(u', l', x(\Psi M \cdot l)) \\
&= x(\Psi M \cdot append(l, u' :: l')) \\
&= \Psi'(xM, append(l, u' :: l')) \\
&= \Psi'(A, append(l, u' :: l')) \ .
\end{aligned}$$

Case $A = (\lambda x.M)N$:

$$\begin{aligned}
insert(u', l', \Psi'(A, l)) &= insert(u', l', \Psi'((\lambda x.M)N, l)) \\
&= insert(u', l', (\lambda x.\Psi M)(\Psi N \cdot l)) \\
&= (\lambda x.\Psi M)(\Psi N \cdot append(l, u' :: l')) \\
&= \Psi'((\lambda x.M)N, append(l, u' :: l')) \\
&= \Psi'(A, append(l, u' :: l')) \ .
\end{aligned}$$

Case $A = A'M$:

$$\begin{aligned}
insert(u', l', \Psi'(A, l)) &= insert(u', l', \Psi'(A'M, l)) \\
&= insert(u', l', \Psi'(A', \Psi M :: l)) \\
&= \Psi'(A', append(\Psi M :: l, u' :: l')), \text{ by IH2,} \\
&= \Psi'(A', \Psi M :: append(l, u' :: l')) \\
&= \Psi'(A'M, append(l, u' :: l')) \\
&= \Psi'(A, append(l, u' :: l')) \ .
\end{aligned}$$

■

The following is an immediate consequence of part 2. of Lemma 40, when $l = []$. Compare with equation $\Psi(app(A)) = \Psi'(A, [])$, which belongs to the definition of Ψ.

Corollary 17 $insert(u, l, \Psi(app(A))) = \Psi'(A, u :: l)$, *for all* A *in* $\lambda\mathcal{N}$ *and* u, l *in* $\lambda\mathcal{P}$.

Corollary 18

1. $\Theta(insert(u, l, t)) = \Theta'(\Theta t @ \Theta u, l)$, *for all* u, t, l *in* $\lambda\mathcal{P}$.

2. $\Theta'(A, append(l, u' :: l')) = \Theta'(\Theta'(A, l) @ \Theta u', l')$, *for all* A *in* $\lambda\mathcal{N}$ *and* u', l, l' *in* $\lambda\mathcal{P}$.

Proof: 1.

$$\begin{aligned}
\Theta(insert(u, l, t)) &= \Theta(insert(\Psi\Theta u, \Psi\Theta l, \Psi\Theta t)), \text{ by Proposition 25,} \\
&= \Theta\Psi'(\Theta t @ \Theta u, l), \text{ by Lemma 40,} \\
&= \Theta'(\Theta t @ \Theta u, l), \text{ by Proposition 24.}
\end{aligned}$$

2.

$$\begin{aligned}
\Theta'(A, append(l, u' :: l')) &= \Theta\Psi'(A, append(l, u' :: l')), \text{ by Proposition 24,} \\
&= \Theta(insert(u', l', \Psi'(A, l))), \text{ by Lemma 40,} \\
&= \Theta(insert(\Psi\Theta u', l', \Psi\Theta'(A, l))), \text{ by Proposition 25,} \\
&= \Theta\Psi'(\Theta'(A, l) @ \Theta u', l'), \text{ by Lemma 40,} \\
&= \Theta'(\Theta'(A, l) @ \Theta u', l'), \text{ by Proposition 24.}
\end{aligned}$$

■

Next results relate substitution in $\lambda\mathcal{N}$ with the operator *subst* of $\lambda\mathcal{P}$.

Lemma 41

1. $\Psi(M[N/x]) = subst(\Psi N, x, \Psi M)$, *all* M, N *in* $\lambda\mathcal{N}$.

2. $subst(\Psi N, x, \Psi'(A, l)) = \Psi'(A[N/x], subst(\Psi N, x, l))$, *all* N, A *in* $\lambda\mathcal{N}$, l *in* $\lambda\mathcal{P}$.

Proof: By simultaneous induction on M and A with induction hypothesis referred to as IH1 and IH2, respectively. Whenever convenient, we write s for $subst$.

Cases $M = x$ and $M = \lambda y.M'$ are straightforward.

Case $M = app(A)$:

$$\begin{array}{rcl} \Psi(M[N/x]) & = & \Psi(app(A)[N/x]) \\ & = & \Psi(app(A[N/x])), \text{ by def. of } \_[N/x], \\ & = & \Psi'(A[N/x], []), \text{ by def. of } \Psi, \\ & = & \Psi'(A[N/x], subst(\Psi N, x, [])), \text{ by def. of } subst, \\ & = & subst(\Psi N, x, \Psi'(A, [])), \text{ by IH2}, \\ & = & subst(\Psi N, x, \Psi(app(A))), \text{ by def. of } \Psi, \\ & = & subst(\Psi N, x, \Psi(M)) \ . \end{array}$$

Case $A = xM$:

$$\begin{array}{rl} & subst(\Psi N, x, \Psi'(A, l)) \\ = & subst(\Psi N, x, \Psi'(xM, l)) \\ = & subst(\Psi N, x, x(\Psi M \cdot l)), \text{ by def. of } \Psi, \\ = & insert(subst(\Psi N, x, \Psi M), subst(\Psi N, x, l), \Psi N), \text{ by def. of } subst, \\ = & insert(\Psi(M[N/x]), subst(\Psi N, x, l), \Psi N), \text{ by IH1}, \\ = & \Psi'(N@M[N/x], subst(\Psi N, x, l)), \text{ by Lemma 40}, \\ = & \Psi'((xM)[N/x], subst(\Psi N, x, l)), \text{ by def. of } \_[N/x], \\ = & \Psi'(A[N/x], subst(\Psi N, x, l)) \ . \end{array}$$

Case $A = (\lambda y.M)M'$:

$$\begin{array}{rl} & s(\Psi N, x, \Psi'(A, l)) \\ = & s(\Psi N, x, \Psi'((\lambda y.M)M', l)) \\ = & s(\Psi N, x, (\lambda y.\Psi M)(\Psi M' \cdot l)), \text{ by def. of } \Psi, \end{array}$$

$$\begin{aligned}
&= (\lambda y.s(\Psi N, x, \Psi M))(s(\Psi N, x, \Psi M') \cdot s(\Psi N, x, l))), \text{ by def. of } subst,\\
&= (\lambda y.\Psi(M[N/x]))(\Psi(M'[N/x]) \cdot s(\Psi N, x, l)), \text{ by IH1},\\
&= \Psi'((\lambda y.M[N/x])(M'[N/x]), s(\Psi N, x, l)), \text{ by def. of } \Psi,\\
&= \Psi'(((\lambda y.M)M')[N/x], s(\Psi N, x, l)), \text{ by def. of } _[N/x],\\
&= \Psi'(A[N/x], s(\Psi N, x, l)) \ .
\end{aligned}$$

Case $A = A'M$:

$$\begin{aligned}
& subst(\Psi N, x, \Psi'(A, l))\\
&= subst(\Psi N, x, \Psi'(A'M, l))\\
&= subst(\Psi N, x, \Psi'(A', \Psi M :: l)), \text{ by def. of } \Psi,\\
&= \Psi'(A'[N/x], subst(\Psi N, x, \Psi M :: l)), \text{ by IH2},\\
&= \Psi'(A'[N/x], subst(\Psi N, x, \Psi M) :: subst(\Psi N, x, l))), \text{ by def. of } subs,\\
&= \Psi'(A'[N/x], \Psi(M[N/x]) :: subst(\Psi N, x, l))), \text{ by IH1},\\
&= \Psi'(A'[N/x]M[N/x], subst(\Psi N, x, l))), \text{ by def. of } \Psi,\\
&= \Psi'((A'M)[N/x], subst(\Psi N, x, l)), \text{ by def. of } _[N/x],\\
&= \Psi'(A[N/x], subst(\Psi N, x, l)) \ .
\end{aligned}$$

■

Corollary 19

1. $\Theta(subst(v, x, t)) = \Theta t[\Theta v/x]$, *all* t, v *in* $\lambda\mathcal{P}$.
2. $\Theta'(A[\Theta u/x], subst(u, x, l)) = \Theta'(A, l)[\Theta u/x]$, *for all* A *in* $\lambda\mathcal{N}$ *and* u, l *in* $\lambda\mathcal{P}$.

Proof: 1.

$$\begin{aligned}
\Theta(subst(v, x, t)) &= \Theta(subst(\Psi\Theta v, x, \Psi\Theta t)), \text{ by Proposition 25},\\
&= \Theta\Psi(\Theta t[\Theta v/x]), \text{ by Lemma 41},\\
&= \Theta t[\Theta v/x], \text{ by Proposition 24}.
\end{aligned}$$

2.

$$\begin{aligned}
& \Theta'(A[\Theta u/x], subst(u,x,l)) \\
= \; & \Theta\Psi'(A[\Theta u/x], subst(\Psi\Theta u,x,l)), \text{ by Propositions 24 and 25,} \\
= \; & \Theta(subst(\Psi\Theta u,x,\Psi'(A,l))), \text{ by Lemma 41,} \\
= \; & \Theta(subst(\Psi\Theta u,x,\Psi\Theta'(A,l))), \text{ by Proposition 25,} \\
= \; & \Theta(\Psi(\Theta'(A,l)[\Theta u/x])), \text{ by Lemma 41,} \\
= \; & \Theta'(A,l)[\Theta u/x], \text{ by Proposition 24.}
\end{aligned}$$

■

Lemma 42 *In $\lambda\mathcal{P}$, if $l \rightarrow_{\beta i} l'$, then $\Psi'(A,l) \rightarrow_{\beta i} \Psi'(A,l')$ (for all A in $\lambda\mathcal{N}$, $i \in \{1,2\}$).*

Proof: By induction on A.

Case $A = xN$. $\Psi'(xN,l) = x(\Psi N \cdot l) \rightarrow_{\beta i} x(\Psi N \cdot l') = \Psi'(xN,l')$, where the reduction step is by $l \rightarrow_{\beta i} l'$ and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $Left2$.

Case $A = (\lambda x.M)N$. Similarly, but by closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $KeyCut3$.

Case $A = A'N$. $\Psi'(A'N,l) = \Psi'(A,\Psi N :: l) \rightarrow_{\beta i} \Psi'(A,\Psi N :: l') = \Psi'(A'N,l')$, where the reduction step is by I.H., as $l \rightarrow_{\beta i} l'$ and $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ is closed under $Lft2$. ■

Lemma 43 *In $\lambda\mathcal{N}$, if $A \rightarrow_{\beta i} A'$, then $\Theta'(A,l) \rightarrow_{\beta i} \Theta'(A',l)$ (for all l in $\lambda\mathcal{P}$, $i \in \{1,2\}$).*

Proof: By induction on l.

Case $l = []$. $\Theta'(A,[]) = app(A) \rightarrow_{\beta i} app(A') = \Theta'(A',[])$, where the reduction step is by $A \rightarrow_{\beta i} A'$ and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{N}$ under App.

Case $l = u :: l'$. $\Theta'(A,u :: l') = \Theta'(A\Theta u,l') \rightarrow_{\beta i} \Theta'(A'\Theta u,l') = \Theta'(A',u :: l')$, where the reduction step is by I.H., as $A \rightarrow_{\beta i} A'$ and $\rightarrow_{\beta i}$ in $\lambda\mathcal{N}$ is closed under

*AElim*1. ■

The first half of the promised isomorphism of normalisation procedures is the following

Theorem 4 *Let $i \in \{1, 2\}$. If $M \to_{\beta i} M'$ in $\lambda\mathcal{N}$ then $\Psi M \to_{\beta i} \Psi M'$ in $\lambda\mathcal{P}$.*

Proof: We prove the claim and also that

$$\text{if } A \to_{\beta i} A' \text{ in } \lambda\mathcal{N}, \text{ then } \Psi'(A, l) \to_{\beta i} \Psi'(A', l) \text{ in } \lambda\mathcal{P}, \text{ for all } l \text{ in } \lambda\mathcal{P},$$

by simultaneous induction on $M \to_{\beta i} M'$ and $A \to_{\beta i} A'$. Cases correspond to closure rules, according to Definition 9. We prove both cases $i = 1, 2$ at the same time.

Case $\beta 1$:

$$\begin{array}{rcl} \Psi(app((\lambda x.M)N)) & = & \Psi'((\lambda x.M)N, []), \text{ by def. of } \Psi, \\ & = & (\lambda x.\Psi M)(\Psi N \cdot []), \text{ by def. of } \Psi, \\ & \to_{\beta 1} & subst(\Psi N, x, \Psi M) \\ & = & \Psi(M[N/x]), \text{ by Lemma 41.} \end{array}$$

Case $\beta 2$:

$$\begin{array}{rcl} \Psi'(((\lambda x.M)N)N', l) & = & \Psi'(((\lambda x.M)N), \Psi N' :: l), \text{ by def. of } \Psi, \\ & = & (\lambda x.\Psi M)(\Psi N \cdot (\Psi N' :: l)), \text{ by def. of } \Psi, \\ & \to_{\beta 2} & insert(\Psi N', l, subst(\Psi N, x, \Psi M)) \\ & = & insert(\Psi N', l, \Psi(M[N/x])), \text{ by Lemma 41,} \\ & = & \Psi'(M[N/x]@N', l), \text{ by Lemma 40.} \end{array}$$

Case *Intro*: Suppose $\Psi(M) \to_{\beta i} \Psi(M')$ (IH1).

$$\begin{array}{rcl}\Psi(\lambda x.M) & = & \lambda x.\Psi(M), \text{ by def. of } \Psi, \\ & \rightarrow_{\beta i} & \lambda x.\Psi(M') \ (*) \\ & = & \Psi(\lambda x.M'), \text{ by def. of } \Psi,\end{array}$$

where the reduction (*) step is by IH1 and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under *Right*.

Case *App*: Suppose $\Psi'(A, l') \rightarrow_{\beta i} \Psi'(A', l')$, all l' (IH2).

$$\begin{array}{rcl}\Psi(app(A)) & = & \Psi'(A, []), \text{ by def. of } \Psi, \\ & \rightarrow_{\beta i} & \Psi'(A', []), \text{by IH2}, \\ & = & \Psi(app(A')), \text{ by def. of } \Psi.\end{array}$$

Case *VElim*: Suppose $\Psi M \rightarrow_{\beta i} \Psi M'$ (IH1).

$$\begin{array}{rcl}\Psi'(xM, l) & = & x(\Psi M \cdot l), \text{ by def. of } \Psi, \\ & \rightarrow_{\beta i} & x(\Psi M' \cdot l) \ (*) \\ & = & \Psi'(xM', l), \text{ by def. of } \Psi,\end{array}$$

where the reduction step (*) is by IH1 and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $Left1$.

Case $Redex1$: Suppose $\Psi M \rightarrow_{\beta i} \Psi M'$ (IH1).

$$\begin{array}{rcl}\Psi'((\lambda x.M)N, l) & = & (\lambda x.\Psi M)(\Psi N \cdot l), \text{ by def. of } \Psi, \\ & \rightarrow_{\beta i} & (\lambda x.\Psi M')(\Psi N \cdot l) \ (*) \\ & = & \Psi'((\lambda x.M')N, l), \text{ by def. of } \Psi,\end{array}$$

where the reduction step (*) is by IH1 and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $KeyCut1$.

Case $Redex2$: Similarly, but by closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $KeyCut2$.

Case $AElim1$: Suppose $\Psi(A, l') \rightarrow_{\beta i} \Psi(A', l')$, all l' (IH2).

$$
\begin{array}{rcl}
\Psi'(AM, l) & = & \Psi'(A, \Psi M :: l), \text{ by def. of } \Psi, \\
& \rightarrow_{\beta i} & \Psi'(A', \Psi M :: l), \text{ by IH2}, \\
& = & \Psi'(A'M, l), \text{ by def. of } \Psi.
\end{array}
$$

Case $AElim2$: Suppose $\Psi M \rightarrow_{\beta i} \Psi M'$ (IH1).

$$
\begin{array}{rcl}
\Psi'(AM, l) & = & \Psi'(A, \Psi M :: l), \text{ by def. of } \Psi, \\
& \rightarrow_{\beta i} & \Psi'(A, \Psi M' :: l) \ (*) \\
& = & \Psi'(AM', l), \text{ by def. of } \Psi,
\end{array}
$$

where the reduction step (*) is by Lemma 42, IH1 and closure of $\rightarrow_{\beta i}$ in $\lambda\mathcal{P}$ under $Lft1$. ■

The second half of the isomorphism is as follows.

Theorem 5 *Let $i \in \{1, 2\}$. If $t \rightarrow_{\beta i} t'$ in $\lambda\mathcal{P}$ then $\Theta t \rightarrow_{\beta i} \Theta t'$ in $\lambda\mathcal{N}$.*

Proof: We prove the claim and also that

if $l \rightarrow_{\beta i} l'$ in $\lambda\mathcal{P}$, then $\Theta'(A, l) \rightarrow_{\beta i} \Theta(A, l')$ in $\lambda\mathcal{N}$, for all A in $\lambda\mathcal{N}$,

by simultaneous induction on $t \rightarrow_{\beta i} t'$ and $l \rightarrow_{\beta i} l'$. Cases correspond to closure rules, according to Definition 5. We prove both cases $i = 1, 2$ at the same time.

Case $\beta 1$:

$$
\begin{array}{rcl}
\Theta((\lambda x.t)(v \cdot [])) & = & \Theta'((\lambda x.\Theta t)\Theta v, []), \text{ by def. of } \Theta, \\
& = & app((\lambda x.\Theta t)\Theta v), \text{ by def. of } \Theta, \\
& \rightarrow_{\beta 1} & \Theta t[\Theta v/x] \\
& = & \Theta(subst(v, x, t)), \text{ by Corollary 19.}
\end{array}
$$

Case $\beta 2$:

$$
\begin{array}{rcl}
\Theta((\lambda x.t)(v \cdot (u :: l))) & = & \Theta'((\lambda x.\Theta t)\Theta v, u :: l), \text{ by def. of } \Theta, \\
& = & \Theta'(((\lambda x.\Theta t)\Theta v)\Theta u, l), \text{ by def. of } \Theta, \\
& \to_{\beta 2} & \Theta'((\Theta t[\Theta v/x])@\Theta u, l), \text{ by Lemma 43}, \\
& = & \Theta'(\Theta(subst(v, x, t))@\Theta u, l), \text{ by Corollary 19}, \\
& = & \Theta(insert(u, l, subst(v, x, t))), \text{ by Corollary 18}.
\end{array}
$$

Case $Left1$: Suppose $\Theta u \to_{\beta i} \Theta u'$ (IH1).

$$
\begin{array}{rcl}
\Theta(x(u \cdot l)) & = & \Theta'(x\Theta u, l), \text{ by def. of } \Theta, \\
& \to_{\beta i} & \Theta'(x\Theta u', l) \ (*) \\
& = & \Theta(x(u' \cdot l)), \text{ by def. of } \Theta,
\end{array}
$$

where the reduction step (*) is by Lemma 43, IH1 and closure of $\to_{\beta i}$ in $\lambda\mathcal{N}$ under $VElim$.

Case $Left2$: Suppose $\Theta'(A, l) \to_{\beta i} \Theta'(A, l')$, all A (IH2).

$$
\begin{array}{rcl}
\Theta(x(u \cdot l)) & = & \Theta'(x\Theta u, l), \text{ by def. of } \Theta, \\
& \to_{\beta i} & \Theta'(x\Theta u, l'), \text{ by IH2}, \\
& = & \Theta(x(u \cdot l')), \text{ by def. of } \Theta.
\end{array}
$$

Case $Right$: Suppose $\Theta t \to_{\beta i} \Theta t'$ (IH1).

$$
\begin{array}{rcl}
\Theta(\lambda x.t) & = & \lambda x.\Theta t, \text{ by def. of } \Theta, \\
& \to_{\beta i} & \lambda x.\Theta t' \ (*) \\
& = & \Theta(\lambda x.t'), \text{ by def. of } \Theta,
\end{array}
$$

where the reduction step (*) is by IH1 and closure of $\to_{\beta i}$ in $\lambda\mathcal{N}$ under $Intro$.

Case $KeyCut1$: Suppose $\Theta t \to_{\beta i} \Theta t'$ (IH1).

$$\begin{array}{rcl}\Theta((\lambda x.t)(u\cdot l)) & = & \Theta'((\lambda x.\Theta t)\Theta u, l), \text{ by def. of } \Theta,\\ & \to_{\beta i} & \Theta'((\lambda x.\Theta t')\Theta u, l) \ (^*)\\ & = & \Theta((\lambda x.t')(u :: l)), \text{ by def. of } \Theta,\end{array}$$

where the reduction step (*) is by Lemma 43, IH1 and closure of $\to_{\beta i}$ in $\lambda\mathcal{N}$ under *Redex*1.

Case *KeyCut*2: Similarly, but by closure of $\to_{\beta i}$ in $\lambda\mathcal{N}$ under *Redex*2.

Case *KeyCut*3 Suppose $\Theta'(A, l) \to_{\beta i} \Theta'(A, l')$, all A (IH2).

$$\begin{array}{rcl}\Theta((\lambda x.t)(u\cdot l)) & = & \Theta'((\lambda x.\Theta t)\Theta u, l), \text{ by def. of } \Theta,\\ & \to_{\beta i} & \Theta'((\lambda x.\Theta t)\Theta u, l'), \text{ by IH2},\\ & = & \Theta((\lambda x.t)(u\cdot l')), \text{ by def. of } \Theta.\end{array}$$

Case *Lft*1: Suppose $\Theta u \to_{\beta i} \Theta u'$ (IH1).

$$\begin{array}{rcl}\Theta'(A, u :: l) & = & \Theta'(A\Theta u, l), \text{ by def. of } \Theta,\\ & \to_{\beta i} & \Theta'(A\Theta u', l) \ (^*)\\ & = & \Theta'(A, u' :: l), \text{ by def. of } \Theta,\end{array}$$

where the reduction step (*) is by Lemma 43, IH1 and closure of $\to_{\beta i}$ in $\lambda\mathcal{N}$ under *AElim*2.

Case *Lft*2: Suppose $\Theta'(A, l) \to_{\beta i} \Theta'(A, l')$, all A (IH2).

$$\begin{array}{rcl}\Theta'(A, u :: l) & = & \Theta'(A\Theta u, l), \text{ by def. of } \Theta,\\ & \to_{\beta i} & \Theta'(A\Theta u, l'), \text{ by IH2},\\ & = & \Theta'(A, u :: l'), \text{ by def. of } \Theta.\end{array}$$

■

Corollary 20 (Isomorphism) *Let $i \in \{1, 2\}$.*

1. $M \to_{\beta i} M'$ *in* $\lambda\mathcal{N}$ *iff* $\Psi M \to_{\beta i} \Psi M'$ *in* $\lambda\mathcal{P}$.

2. $t \to_{\beta i} t'$ *in* $\lambda\mathcal{P}$ *iff* $\Theta t \to_{\beta i} \Theta t'$ *in* $\lambda\mathcal{N}$.

Corollary 21 $\lambda\mathcal{P}$, $\lambda\mathcal{P}h$, $\lambda\mathcal{P}h\mathtt{x}$ *and* $\overline{\lambda}\mathcal{P}h\mathtt{x}$ *are confluent.*

Proof: First, $\lambda\mathcal{P}$ is confluent from Corollary 20 and confluence of $\lambda\mathcal{N}$. Then, confluence of the other calculi follows from Corollaries 2, 6 and 10. ■

Chapter 5

Gentzen versus Prawitz

In this chapter we continue the analysis of the relationship between cut-elimination in the canonical fragment and normalisation. We recall two mappings of natural deduction into sequent calculus, one due to Gentzen and the other due to Prawitz. We show that they both are isomorphisms, and that the isomorphic image of λ by Prawitz's mapping $\mathcal{P}$ is $\lambda\mathcal{P}$. Then, a comparison of mappings Ψ and $\mathcal{P}$ as mappings for "turning the main branch upside down" suggests that the advantage of $\lambda\mathcal{N}$ over λ is that $\lambda\mathcal{N}$ includes a built-in distinction between *head* and *tail* applications. Finally, we study (an extension of) the inverse of $\mathcal{P}$, named $\mathcal{Q}$. This mapping is the restriction to the canonical fragment of the good old φ.

5.1 Gentzen's mapping

In the original paper where sequent calculus was introduced [Gentzen, 1935][1], Gentzen proposed the well-known mapping of NJ derivations into sequent calculus derivations that, essentially, translates assumptions as axioms, introduction rules as right rules and elimination rules as cuts plus left rules. For instance, elimination of $\supset$ becomes (ignoring contexts)

[1]The origin of natural deduction precedes Gentzen's paper [Prawitz, 1965].

$$\dfrac{... \vdash A \supset B \qquad \dfrac{... \vdash A \qquad \dfrac{}{..., B \vdash B}\,Ax}{..., A \supset B \vdash B}\,Left}{... \vdash B}\,Cut$$

Therefore, every elimination rule becomes a cut and, in general, normal proofs are not mapped to cut-free derivations.

Let us call this translation $\mathcal{G}$ and let us restrict ourselves to implication. An immediate observation is that, in a derivation in the range of $\mathcal{G}$: (1) every instance of the left rule is canonical. Actually, the active formula of the right premiss of each left inference is main in an axiom. (2) the right cut formula of every cut instance is main in such an instance of the left rule. These observations suggest that Gentzen's mapping may be written as the following mapping from λ into $\lambda\mathcal{P}h$:

$$\begin{aligned} \mathcal{G}x &= x \\ \mathcal{G}(\lambda x.M) &= \lambda x.\mathcal{G}M \\ \mathcal{G}(MN) &= \mathcal{G}M(\mathcal{G}N \cdot []) \ . \end{aligned}$$

Application may be seen as a very particular kind of head-cut, namely a head-cut in which the list of extra arguments is empty.

The origin of this mapping as a translation of logical systems guarantees its correctness.

Proposition 26 (Correctness of $\mathcal{G}$) *If λ derives $\Gamma \vdash M : A$, then $\lambda\mathcal{P}h$ derives $\Gamma; - \vdash \mathcal{G}(M) : A$.*

Proof: By induction on M. The only interesting case is $M = M_0 N_0$. Suppose λ derives $\Gamma \vdash M_0 N_0 : A$. Then λ derives $\Gamma \vdash M_0 : B \supset A$ and $\Gamma \vdash N_0 : B$, for some B. By induction hypothesis, there are derivations in $\lambda\mathcal{P}h$ of $\Gamma; - \vdash \mathcal{G}(M_0) : B \supset A$ and $\Gamma; - \vdash \mathcal{G}(N_0) : B$. These are combined with an application of the head-cut rule:

$$\dfrac{\dfrac{\vdots}{\Gamma; - \vdash \mathcal{G}(M_0) : B \supset A} \quad \dfrac{\vdots}{\Gamma; - \vdash \mathcal{G}(N_0) : B} \quad \dfrac{}{\Gamma; A \vdash [] : A}\,Ax}{\Gamma; - \vdash \mathcal{G}(M_0)(\mathcal{G}(N_0) \cdot []) : A}\,HeadCut$$

■

As we may observe, there are no instances of the left rule in derivations of $\Gamma; - \vdash \mathcal{G}M : A$ in $\lambda\mathcal{P}h$. They are "absorbed" in head-cuts. Actually, $\mathcal{G}$ maps into the ::-free fragment of $\lambda\mathcal{P}h$, that is

$$\begin{array}{rcl} t, u & ::= & x \,|\, \lambda x.t \,|\, t(u \cdot l) \\ l & ::= & [] \end{array}$$

In this fragment, lists are really residual. Let us write $t(u \cdot [])$ as $t[u]$ [2], and let us rewrite the previous grammar as

$$t, u ::= x \,|\, \lambda x.t \,|\, t[u] \ .$$

This is very much like λ-calculus. Such impression is fully confirmed.

As to typing rules, in this fragment sequents have the form $\Gamma; - \vdash t : A$, typing rules for variables and λ-abstraction are as usual, and $t[u]$ is typed as an application by the rule

$$\dfrac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A}{\Gamma; - \vdash t[u] : B}$$

which should be seen as an abbreviation of the head-cut

$$\dfrac{\Gamma; - \vdash t : A \supset B \quad \Gamma; - \vdash u : A \quad \dfrac{}{\Gamma; B \vdash [] : B}\,Ax}{\Gamma; - \vdash t(u \cdot []) : B}\,HeadCut$$

As to reduction, only rule $\beta 1$ makes sense in this fragment, as both $\beta 2$ and h require ::. In the fragment, $\beta 1$ reads

[2]Recall from the relation between $\overline{\lambda}\mathcal{P}h\mathbf{x}$ and $\overline{\lambda}_3$ that $t(u \cdot [])$ may be seen as $t(u :: [])$ anyway.

$$(\lambda x.t)[u] \quad \rightarrow \quad subst(u, x, t) \ ,$$

where *subst* is the operator *subst* of $\lambda\mathcal{P}h$. Now, the calculation

$$\begin{aligned}
subst(v, x, t[u]) &= subst(v, x, t(u \cdot [])) \\
&= subst(v, x, t)(subst(v, x, u) \cdot subst(v, x, [])) \\
&= subst(v, x, t)(subst(v, x, u) \cdot []) \\
&= subst(v, x, t)[subst(v, x, u)]
\end{aligned}$$

shows two things. First, that the ::-free fragment is indeed a fragment of $\lambda\mathcal{P}h$ because it is closed for *subst* and $\beta 1$. Second, that the restriction of *subst* to this fragment behaves exactly as λ-calculus' substitution (the calculation is enough because the other cases in the definition of *subst* did not raise any doubt).

Therefore, the ::-free fragment of $\lambda\mathcal{P}h$ is simply a rephrasing of λ, where application is written $t[u]$, substitution is written *subst* and sequents are written in the form $\Gamma; - \vdash t : A$. Furthermore, Gentzen's mapping $\mathcal{G}$ is trivially an isomorphism between λ and this fragment, because $\mathcal{G}$ is a mere rephrasing mapping. This justifies the following terminology.

Definition 10 *The* ::-*free fragment of* $\lambda\mathcal{P}h$ *is denoted* $\lambda\mathcal{G}$.

As a by-product, we get the following gentle addition to the theory of the relationship between cut-elimination and normalisation [Gentzen, 1935, Prawitz, 1965, Zucker, 1974, Pottinger, 1977, Ungar, 1992]:

Theorem 6 *Gentzen's mapping* $\mathcal{G}$ *is an isomorphism from normalisation in* λ *to cut-elimination in* $\lambda\mathcal{G}$.

5.2 Prawitz's mapping

With the purpose of showing cut-elimination as a corollary of normalisation in natural deduction, Prawitz proposed in [Prawitz, 1965] a mapping from *normal* NJ proofs to *cut-free* derivations in a sequent calculus. Hence, Prawitz's mapping is an improvement over Gentzen's translation w.r.t preservation of normality (see also §6.3 in [Troelstra and Schwitchtenberg, 2000]). This optimisation makes use of the structure of normal proofs, a structure which Prawitz had just uncovered.

The new mapping (call it $\mathcal{P}$) translates again assumptions as axioms and introductions as instances of the right rule. Now suppose our normal proof M is an elimination. If we go upwards through the *main branch* [Prawitz, 1965] of M, we visit the main premiss of successive elimination rules until we stop at an assumption. Hence, M has the form (ignoring contexts)

$$\begin{array}{c}
\dfrac{\dfrac{}{..., x \vdash x : A_1 \supset ... \supset A_k \supset B}\,Var \qquad \begin{array}{c}\vdots \\ ... \vdash N_1 : A_1\end{array}}{..., x \vdash xN_1 : A_2 \supset ... \supset A_k \supset B}\,Elim \\
\vdots \\
\dfrac{..., x \vdash xN_1...N_{k-1} : A_k \supset B \qquad \begin{array}{c}\vdots \\ ... \vdash N_k : A_k\end{array}}{..., x \vdash xN_1...N_k : B}\,Elim
\end{array}$$

for some $k \geq 1$ (if $k = 1$, $A_2 \supset ... \supset A_k \supset B$ is just B). We now extract from this proof two smaller proofs. The first is just

$$\begin{array}{c}\vdots \\ ... \vdash N_1 : A_1\end{array}$$

whereas the second is

$$
\dfrac{\dfrac{}{..., z_1 \vdash z_1 : A_2 \supset ... \supset A_k \supset B}\,Var \qquad \begin{array}{c}\vdots\\ ... \vdash N_2 : A_2\end{array}}{\dfrac{\begin{array}{c}..., z_1 \vdash z_1 N_2 : A_3 \supset ... \supset A_k \supset B\\ \vdots \\ ..., z_1 \vdash z_1 N_2 ... N_{k-1} : A_k \supset B\end{array} \qquad \begin{array}{c}\vdots\\ ... \vdash N_k : A_k\end{array}}{..., z_1 \vdash z_1 N_2 ... N_k : B}\,Elim}\,Elim
$$

where z_1 is free in no N_i. In the case $k = 1$, this proof consists solely of the assumption

$$
\dfrac{}{..., z_1 : B \vdash z_1 : B}\,Var
$$

Now apply $\mathcal{P}$ to these two smaller proofs. If $\mathcal{P}$ is correct, we get two cut-free derivations of sequents $... \vdash \mathcal{P}(N_1) : A_1$ and $..., z_1 : A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_1 N_2 ... N_k) : B$. Finally, conclude with an application of the left rule:

$$
\dfrac{\begin{array}{c}\vdots\\ ... \vdash \mathcal{P}(N_1) : A_1\end{array} \qquad \begin{array}{c}\vdots\\ ..., z_1 : A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_1 N_2 ... N_k) : B\end{array}}{..., x : A_1 \supset A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(x N_1 ... N_k) : B}\,Left
$$

If we borrow from Chapter 2 the notation for proofs in a generic sequent calculus, Prawitz's mapping is defined by

$$
\begin{array}{rcl}
\mathcal{P}(x) & = & \mathsf{Ax}(x)\\
\mathcal{P}(\lambda x.M) & = & \mathsf{R}((x)\mathcal{P}(M))\\
\mathcal{P}(x N_1 ... N_k) & = & \mathsf{L}(x, \mathcal{P}(N_1), (z_1)\mathcal{P}(z_1 N_2 ... N_k))
\end{array}
$$

As an algorithm for performing the translation, this recursive definition is rather inefficient because, each time we have to calculate $\mathcal{P}(MN)$, we need to match MN with $x N_1 ... N_k$, which means inspecting M without reusing previous inspections or saving information for subsequent calculations. Is there a definition of $\mathcal{P}$ with which we travel through a main branch $xN_1...N_k$ just once? Another issue, like in the case of $\mathcal{G}$, is whether the derivations in the range of this translation are of a particular form. It turns out that these two questions are related.

Let us go back to the derivation of

$$...,x : A_1 \supset A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(xN_1...N_k) : B$$

and let us unfold the derivation of

$$...,z_1 : A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_1N_2...N_k) : B \ .$$

We obtain

$$
\dfrac{\begin{array}{c}\vdots\\ ... \vdash \mathcal{P}(N_1) : A_1\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ ... \vdash \mathcal{P}(N_2) : A_2\end{array} \qquad \begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ ... \vdash \mathcal{P}(N_k) : A_k\end{array} \qquad \dfrac{}{...,z_k : B \vdash \mathcal{P}(z_k) : B}\,Ax}{...,z_{k-1} : A_k \supset B \vdash \mathcal{P}(z_{k-1}N_k) : B}\,Left\\ \vdots\\ ...,z_2 : A_3 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_2N_3...N_k) : B\end{array}}{...,z_1 : A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(z_1N_2...N_k) : B}\,Left}{...,x : A_1 \supset A_2 \supset ... \supset A_k \supset B \vdash \mathcal{P}(xN_1...N_k) : B}\,Left
$$

Again, we may observe that each displayed left inference is canonical because each z_i is fresh. Therefore, each $Left$ occurrence, except the lower one, is indeed a Lft-inference and corresponds to the :: constructor, and the right subderivation may be represented by $[\mathcal{P}(N_2), ..., \mathcal{P}(N_k)]$. As to the lower inference, it corresponds to the constructor $x(u \cdot l)$. Hence, $\mathcal{P}(xN_1N_2...N_k) = x(\mathcal{P}(N_1) \cdot [\mathcal{P}(N_2), ..., \mathcal{P}(N_k)])$.

We would like to consider the range of $\mathcal{P}$ to be in $\lambda\mathcal{P}$ instead of $\lambda\mathcal{P}h$ because in the latter $x(u \cdot l)$ is a cut and hence preservation of normality is lost. But the decisive argument concerns the shape of the inevitable cuts in the range of $\mathcal{P}$ when one translates non-normal proofs. In such *generalised* Prawitz's mapping, the translation of an application requires again a walk through the main premiss of successive instances of the elimination rule. However, such walk may now end in the conclusion of an introduction rule. The new case in the definition of $\mathcal{P}$ is then $\mathcal{P}((\lambda x.M)N_1N_2...N_k)$. It is rather natural to define this to be the key cut $(\lambda x.\mathcal{P}(M))(\mathcal{P}(N_1)\cdot[\mathcal{P}(N_2),...,\mathcal{P}(N_k)]$. We do not need any other kind of cut in the range of $\mathcal{P}$.

Summing up, $\mathcal{P}$ is a mapping from λ to $\lambda\mathcal{P}$ defined by the clauses

$$\begin{array}{rcll}
\mathcal{P}(x) & = & x & (5.1)\\
\mathcal{P}(\lambda x.M) & = & \lambda x.\mathcal{P}(M) & (5.2)\\
\mathcal{P}(xN_1N_2...N_k) & = & x(\mathcal{P}(N_1)\cdot[\mathcal{P}(N_2),...,\mathcal{P}(N_k)]) & (5.3)\\
\mathcal{P}((\lambda x.M)N_1N_2...N_k) & = & (\lambda x.\mathcal{P}(M))(\mathcal{P}(N_1)\cdot[\mathcal{P}(N_2),...,\mathcal{P}(N_k)]) & (5.4)
\end{array}$$

This definition is somewhat informal because of the implicit decomposition of an application MN. What recursion is being used? What is, after all, $\mathcal{P}(MN)$? We will try to shed some light at the above questions by studying other mappings from λ to $\lambda\mathcal{P}$ that we met before.

Theorem 7 (λ-square) *The following square commutes*[3]*:*

$$\begin{tikzcd}
\lambda\mathcal{G} \arrow[d, "(_)^-"'] & \lambda \arrow[l, "\mathcal{G}"'] \arrow[d, "\mathcal{N}"] \\
\lambda\mathcal{P} & \lambda\mathcal{N} \arrow[l, "\Psi"']
\end{tikzcd}$$

Proof: We prove $\Psi(\mathcal{N}M)=(\mathcal{G}M)^-$ by induction on M. Cases $M=x$ and $M=\lambda x.M_0$ are straightforward. Let $M=M_0N_0$. Then,

$$\begin{array}{rcl}
\Psi(\mathcal{N}M) & = & \Psi(\mathcal{N}(M_0N_0))\\
& = & \Psi(app(\mathcal{N}(M_0)@\mathcal{N}(N_0))),\text{ by def. of }\mathcal{N},
\end{array}$$

[3]An observation similar to this in spirit is due to Curien and Herbelin, and may be found if one reads intuitionistically the second half of Proposition 2.3 of [Curien and Herbelin, 2000].

$$
\begin{aligned}
&= \Psi'(\mathcal{N}(M_0)@\mathcal{N}(N_0), []), \text{ by def. of } \Psi,\\
&= \mathit{insert}(\Psi(\mathcal{N}(N_0)), [], \Psi(\mathcal{N}(M_0))), \text{ by Lemma 40},\\
&= \mathit{insert}((\mathcal{G}N_0)^-, [], (\mathcal{G}M_0)^-), \text{ by IH},\\
&= (\mathcal{G}M_0(\mathcal{G}N_0 \cdot []))^-, \text{ by def. of } (\_)^- \text{ and } []^- = [],\\
&= (\mathcal{G}(M_0N_0))^-, \text{ by def. of } \mathcal{G},\\
&= (\mathcal{G}M)^- \ .
\end{aligned}
$$

■

Let us clarify the situation after Theorem 7. Since $\mathcal{G}; (\_)^-$ is an isomorphism (because it is the composition of two isomorphisms), we have

$$M \to_\beta N \quad \text{iff} \quad (\mathcal{G}M)^- \to (\mathcal{G}N)^- \ . \tag{5.5}$$

When restricted to the codomain of $\mathcal{G}$, $(\_)^-$ is bijective (because $\mathcal{G}; (\_)^-$ also is). Moreover, by

$$\mathcal{G}M \to_{\beta 1} \mathcal{G}N \quad \text{iff} \quad M \to_\beta N$$

and (5.5), it is also an isomorphism of normalisation procedures. Therefore, in the above square (which we refer to as the λ-square), the four vertices are isomorphic systems.

From Theorem 7, the two compositions of arrows in the λ-square that lead from λ to $\lambda\mathcal{P}$ are one and the same mapping. Let us call this mapping the *diagonal* of the λ-square. Now it comes the official definition of $\mathcal{P}$.

Definition 11 (Prawitz's mapping) Prawitz's mapping $\mathcal{P}$ *is the diagonal of the λ-square.*

That is, $\mathcal{P}(M) = (\mathcal{G}M)^- = \Psi(\mathcal{N}(M))$, for all M in λ. Let us see informally that this definition agrees with definition given by clauses (5.1)-(5.4). As to variables and λ-abstractions, the situation is clear. As to applications, the calculations

$$
\begin{aligned}
\Psi(\mathcal{N}(xN_1N_2...N_k)) &= \Psi(app(x\mathcal{N}(N_1)\mathcal{N}(N_2)...\mathcal{N}(N_k))) \\
&= \Psi'(x\mathcal{N}(N_1)\mathcal{N}(N_2)...\mathcal{N}(N_k), []) \\
&= \Psi'(x\mathcal{N}(N_1), [\Psi(\mathcal{N}(N_2)), ..., \Psi(\mathcal{N}(N_k))]) \\
&= x(\Psi(\mathcal{N}(N_1)) \cdot [\Psi(\mathcal{N}(N_2)), ..., \Psi(\mathcal{N}(N_k))])
\end{aligned}
$$

and

$$
\begin{aligned}
& (\mathcal{G}((\lambda x.M)N_1N_2...N_k))^- \\
= \; & ((\lambda x.\mathcal{G}M)[\mathcal{G}N_1][\mathcal{G}N_2]...[\mathcal{G}N_k])^- \\
= \; & insert((\mathcal{G}N_k)^-, [], ...insert((\mathcal{G}N_2)^-, [], insert((\mathcal{G}N_1)^-, [], \lambda x.(\mathcal{G}M)^-))...) \\
= \; & insert((\mathcal{G}N_k)^-, [], ...insert((\mathcal{G}N_2)^-, [], (\lambda x.(\mathcal{G}M)^-)((\mathcal{G}N_1)^- \cdot []))...) \\
= \; & insert((\mathcal{G}N_k)^-, [], ...(\lambda x.(\mathcal{G}M)^-)((\mathcal{G}N_1)^- \cdot [(\mathcal{G}N_2)^-])...) \\
= \; & (\lambda x.(\mathcal{G}M)^-)((\mathcal{G}N_1)^- \cdot [(\mathcal{G}N_2)^-, ..., (\mathcal{G}N_k)^-])
\end{aligned}
$$

give enough evidence.

In the next result, one finds an answer to the question of what $\mathcal{P}(MN)$ is.

Proposition 27 *Prawitz's mapping is the unique mapping $\mathcal{P} : \lambda \to \lambda\mathcal{P}$ such that:*

$$
\begin{aligned}
\mathcal{P}x &= x \\
\mathcal{P}(\lambda x.M) &= \lambda x.\mathcal{P}M \\
\mathcal{P}(MN) &= insert(\mathcal{P}N, [], \mathcal{P}M)
\end{aligned}
$$

Proof: Because $(\_)^- \circ \mathcal{G}$ satisfies these equations. ∎

The clause for applications explains the difference between $\mathcal{G}$ and $\mathcal{P}$. $\mathcal{G}(MN)$ is simply the cut $\mathcal{G}M[\mathcal{G}N]$, whereas $\mathcal{P}$ requires, in addition, the complete left permutation of this cut, performed by *insert*.

Theorem 8 (Gentzen vs Prawitz) *Prawitz's translation of a proof is obtained from Gentzen's translation of the same proof by the complete left permutation of every cut occurring in the latter.*

Proof: Sum up the following facts: (1) $\mathcal{P} = (\_)^{-} \circ \mathcal{G}$. (2) Mapping $(\_)^{-} : \lambda\mathcal{G} \to \lambda\mathcal{P}$ is the restriction to $\lambda\mathcal{G}$ of $(\_)^{-} : \lambda\mathcal{P}h \to \lambda\mathcal{P}$. (3) The latter is the same as $\downarrow_h$. (4) In $\lambda\mathcal{P}h$, a term is a h-redex iff it is a left-permutable cut. ■

Finally, because $\mathcal{P}$ is a composition of isomorphisms, the following holds:

Theorem 9 *Prawitz's mapping $\mathcal{P}$ is an isomorphism from normalisation in λ to cut-elimination in $\lambda\mathcal{P}$.*

5.3 The nature of $\lambda\mathcal{N}$

In terms of derivations, one of the effects of Prawitz's mapping is to turn the main branch upside down, so to speak. Here "main branch" may have its usual sense in normal proofs, or a suitably generalised sense that even applies to non-normal proofs. Observe how the uppermost instance of the elimination rule in the main branch corresponds, in the translated derivation, to the lowest instance of the left rule; and how the instance of the elimination rule just below the former corresponds to the instance of the left rule just above the latter, and so on. This effect can be described in terms of bracketing. The term $(...((xN_1)N_2)...N_k)$, which is bracketed to the left, is translated as $x(PN_1 \cdot (PN_2 :: ...(PN_k :: [])...))$, which is bracketed to the right.

Mapping Ψ is another example of a translation into $\lambda\mathcal{P}$ which maps applications in a similar way, by turning main branches upside down. We might say that Ψ and $\mathcal{P}$ are based on the same idea, but that they differ because they translate two different formulations of the λ-calculus, namely usual λ and $\lambda\mathcal{N}$. Our goal is, by comparing the two mappings Ψ and $\mathcal{P}$, to understand the difference between the two formulations of the λ-calculus, and particularly what is the "nature" of $\lambda\mathcal{N}$. Along the way, we exploit the relation between Ψ and $\mathcal{P}$, as we did before with the relation between $\mathcal{P}$ and $\mathcal{G}$.

The first thing we want to do is to give another definition of $\mathcal{P}$, but one that is close to the spirit of Ψ. With this purpose, we introduce a new inductive definition of the λ-terms.

Definition 12 *The sets T and Ap are defined by the following simultaneous induction:*

$$\frac{}{x \in T} \quad \frac{M \in T}{\lambda x.M \in T} \quad \frac{(M,N) \in Ap}{MN \in T}$$

$$\frac{N \in T}{(x,N) \in Ap} \quad \frac{M \in T \quad N \in T}{(\lambda x.M, N) \in Ap} \quad \frac{(M_1, M_2) \in Ap \quad N \in T}{(M_1 M_2, N) \in Ap}$$

Lemma 44 *If $M \in T$ and $N \in T$, then $(M,N) \in Ap$ and hence $MN \in T$.*

Proof: By a case analysis of M. Case $M = x$. Since $N \in T$, $(x,N) \in Ap$. Case $M = \lambda x.M'$. Since $M \in T$, it follows $M' \in T$. From this and $N \in T$, it follows $(\lambda x.M', N) \in Ap$. Case $M = M_1 M_2$. Since $M \in T$, it follows $(M_1, M_2) \in Ap$. From this and $N \in T$, it follows $(M_1 M_2, N) \in Ap$. ■

Proposition 28 *$M \in T$ iff M is a λ-term.*

Proof: "If": By induction on a λ-term M. Case $M = x$. $x \in T$. Case $M = \lambda x.M'$. By IH, $M' \in T$ and, thus, $\lambda x.M' \in T$. Case $M = M'N$. By IH, $M' \in T$ and $N \in T$. By Lemma 44, $M'N \in T$.

"Only if": We prove that, for all $M \in T$, M is a λ-term, and that, for all $(M,N) \in Ap$, both M and N are λ-terms, by simultaneous induction on M and (M,N), with induction hypotheses IH1 and IH2, respectively.

Case $M = x$. x is a λ-term.

Case $M = \lambda x.M'$. By IH1, M' is a λ-term. Hence, $\lambda x.M'$ is a λ-term.

Case $M = M'N$. Then $(M', N) \in Ap$. By IH2, both M' and N are λ-terms. Hence, $M'N$ is a λ-term.

Case $(M,N) = (x, N')$. On the one hand, x is a λ-term. On the other hand, $N' \in T$. By IH1, N' is a λ-term.

Case $(M, N) = (\lambda x.M', N')$. On the one hand, $M' \in T$ and, by IH1, M' is a λ-term. Hence $\lambda x.M'$ is a λ-term. On the other hand, $N' \in T$. By IH1, N' is a λ-term.

Case $(M, N) = (M'M'', N')$. On the one hand, $(M', M'') \in Ap$ and, by IH2, both M' and M'' are λ-terms. Hence $M'M''$ is a λ-term. On the other hand, $N' \in T$. By IH1, N' is a λ-term. ■

Corollary 22 *$(M, N) \in Ap$ iff M and N are λ-terms.*

Proof: "Only if": see the the "only if" part of the proof of last Proposition. "If": If M and N are λ-terms, then M and N are in T (by last Proposition) and $(M, N) \in Ap$ (by Lemma 44). ■

The definition of $\mathcal{N} : \lambda \to \lambda\mathcal{N}$ needs an adjustment, if one takes the new definition of λ-terms. We define $(\_)^n$, where n is mnemonic for $\mathcal{N}$.

Definition 13 *The mapping $(\_)^n : \lambda \to \lambda\mathcal{N}$ is defined by:*

$$
\begin{array}{rcl}
x^n & = & x \\
(\lambda x.M)^n & = & \lambda x.M^n \\
(MN)^n & = & app((M, N)^n)
\end{array}
$$

$$
\begin{array}{rcl}
(x, N)^n & = & xN^n \\
(\lambda x.M, N)^n & = & (\lambda x.M^n)N^n \\
(M_1M_2, N)^n & = & (M_1, M_2)^n N^n
\end{array}
$$

Actually, this defines a mapping sending $M \in T$ to some $\lambda\mathcal{N}$-term and another mapping sending $(M, N) \in Ap$ to some application A in $\lambda\mathcal{N}$.

Proposition 29 *$M^n = \mathcal{N}(M)$ and $(M, N)^n = \mathcal{N}(M)@\mathcal{N}(N)$, for all M, N in λ.*

Proof: By simultaneous induction on M and (M, N), with induction hypotheses referred to by IH1 and IH2, respectively.

Cases $M = x$ and $M = \lambda x.M_0$: straightforward.

Case $M = M_0 N_0$:

$$\begin{aligned} M^n &= (M_0 N_0)^n \\ &= app((M_0, N_0)^n), \text{ by def. of } (\_)^n, \\ &= app(\mathcal{N}(M_0)@\mathcal{N}(N_0)), \text{ by IH2}, \\ &= \mathcal{N}(M_0 N_0), \text{ by def. of } \mathcal{N}, \\ &= \mathcal{N}(M) \ . \end{aligned}$$

Case $(M, N) = (x, N_0)$:

$$\begin{aligned} (M, N)^n &= (x, N_0)^n \\ &= x N_0^n, \text{ by def. of } (\_)^n, \\ &= x\mathcal{N}(N_0), \text{ by IH1}, \\ &= x@\mathcal{N}(N_0), \text{ by def. of } @, \\ &= \mathcal{N}(x)@\mathcal{N}(N_0), \text{ by def. of } \mathcal{N}, \\ &= \mathcal{N}(M)@\mathcal{N}(N) \ . \end{aligned}$$

Case $(M, N) = (\lambda x.M_0, N_0)$. Similar.

Case $(M, N) = (M_1 M_2, N_0)$:

$$\begin{aligned} (M, N)^n &= (M_1 M_2, N_0)^n \\ &= (M_1, M_2)^n N_0^n, \text{ by def. of } (\_)^n, \\ &= (\mathcal{N}(M_1)@\mathcal{N}(M_2))\mathcal{N}(N_0), \text{ by IH1,IH2}, \\ &= app(\mathcal{N}(M_1)@\mathcal{N}(M_2))@\mathcal{N}(N_0), \text{ by def. of } @, \\ &= \mathcal{N}(M_1 M_2)@\mathcal{N}(N_0), \text{ by def. of } \mathcal{N}, \\ &= \mathcal{N}(M)@\mathcal{N}(N) \ . \end{aligned}$$

■

Here is the promised new definition of $\mathcal{P}$, with recursion according to the new inductive definition of λ-terms.

Proposition 30 *Prawitz's mapping is the unique mapping* $\mathcal{P} : \lambda \to \lambda\mathcal{P}$ *such that:*

$$\begin{array}{rcl} \mathcal{P}x & = & x \\ \mathcal{P}(\lambda x.M) & = & \lambda x.\mathcal{P}M \\ \mathcal{P}(MN) & = & \mathcal{P}'(M, N, []) \end{array}$$

$$\begin{array}{rcl} \mathcal{P}'(x, N, l) & = & x(\mathcal{P}N \cdot l) \\ \mathcal{P}'(\lambda x.M, N, l) & = & (\lambda x.\mathcal{P}M)(\mathcal{P}N \cdot l) \\ \mathcal{P}'(M_1 M_2, N, l) & = & \mathcal{P}'(M_1, M_2, \mathcal{P}N :: l) \end{array}$$

Proof: $\mathcal{P}'(M, N, l)$ is to be understood as $\mathcal{P}'((M, N), l)$, with $(M, N) \in Ap$. Recall that, by definition, $\mathcal{P}M$ is $\Psi(\mathcal{N}(M))$, which is the same as $\Psi(M^n)$, by Proposition 29. In this proof we let $\mathcal{P}$ denote the mapping defined by the above recursive definition. We prove $\mathcal{P}M = \Psi(M^n)$ and $\mathcal{P}'(M, N, l) = \Psi'((M, N)^n, l)$ by simultaneous induction on M and (M, N), with induction hypotheses IH1 and IH2, respectively.

Cases $M = x$ and $M = \lambda x.M_0$: straightforward.

Case $M = M_0 N_0$:

$$\begin{array}{rcl} \mathcal{P}M & = & \mathcal{P}(M_0 N_0) \\ & = & \mathcal{P}'(M_0, N_0, []), \text{ by def. of } \mathcal{P}, \\ & = & \Psi'((M_0, N_0)^n, []), \text{ by IH2}, \\ & = & \Psi(app(M_0, N_0)^n), \text{ by def. of } \Psi, \\ & = & \Psi((M_0 N_0)^n), \text{ by def. of } (\_)^n, \\ & = & \Psi(M^n) \ . \end{array}$$

Case $(M, N) = (x, N_0)$:

$$\begin{aligned}
\mathcal{P}'(M, N, l) &= \mathcal{P}'(x, N_0, l) \\
&= x(\mathcal{P}N_0 \cdot l), \text{ by def. of } \mathcal{P}, \\
&= x(\Psi(N_0^n) \cdot l), \text{ by IH1}, \\
&= \Psi'(xN_0^n, l), \text{ by def. of } \Psi, \\
&= \Psi'((x, N_0)^n, l), \text{ by def. of } (_)^n, \\
&= \Psi'((M, N)^n, l) \ .
\end{aligned}$$

Case $(M, N) = (\lambda x.M_0, N_0)$. Similar.
Case $(M, N) = (M_1 M_2, N_0)$:

$$\begin{aligned}
\mathcal{P}'(M, N, l) &= \mathcal{P}'(M_1 M_2, N_0, l) \\
&= \mathcal{P}'(M_1, M_2, \mathcal{P}N_0 :: l), \text{ by def. of } \mathcal{P}, \\
&= \mathcal{P}'(M_1, M_2, \Psi(N_0^-) :: l), \text{ by IH1}, \\
&= \Psi'((M_1, M_2)^n, \Psi(N_0^-) :: l), \text{ by IH2}, \\
&= \Psi'((M_1, M_2)^n N_0^n, l), \text{ by def. of } \Psi, \\
&= \Psi'(((M_1 M_2), N_0)^n, l), \text{ by def. of } (_)^n, \\
&= \Psi'((M, N)^n, l) \ .
\end{aligned}$$

■

This proposition, which allows a comparison between $\mathcal{P}$ and Ψ, should be contrasted with Proposition 27, which allowed a comparison between $\mathcal{P}$ and $\mathcal{G}$. The proposition also contains a new way of calculating $\mathcal{P}(MN)$.

We also need to adjust mapping $|\_| : \lambda\mathcal{N} \to \lambda$ to the new inductive definition of the λ-terms. The new mapping is denoted $(\_)^a$, with a mnemonic for "absolute value".

Definition 14 *The mapping* $(\_)^a : \lambda\mathcal{N} \to \lambda$ *is defined by:*

$$\begin{array}{rcl} x^a & = & x \\ (\lambda x.M)^a & = & \lambda x.M^a \\ app(A)^a & = & let\ (M,N)\ be\ A^a\ in\ MN \end{array}$$

$$\begin{array}{rcl} (xN)^a & = & (x, N^a) \\ ((\lambda x.M)N)^a & = & (\lambda x.M^a, N^a) \\ (AN)^a & = & let\ (M_1, M_2)\ be\ A^a\ in\ (M_1M_2, N^a) \end{array}$$

Actually, this defines a mapping that sends a $\lambda\mathcal{N}$-term to some $M \in T$ and another mapping that sends each A in $\lambda\mathcal{N}$ to a $(M,N) \in Ap$. Notice the use of an informal "let" notation. We now see that this definition agrees with $|\_|$.

Proposition 31 *For all* M *in* $\lambda\mathcal{N}$, $M^a = |M|$. *For all* A *in* $\lambda\mathcal{N}$, *let* $A^a = (M,N)$ *and* $|A| = M'N'$. *Then* $M = M'$ *and* $N = N'$.

Proof: By simultaneous induction on M and A, with induction hypotheses referred to as IH1 and IH2, respectively.

Cases $M = x$ and $M = \lambda x.M_0$: straightforward.

Case $M = app(A)$. Let $A^a = (M,N)$ and $|A| = M'N'$. By IH2, $M = M'$ and $N = N'$. Hence, $MN = M'N'$. Then, $M^a = app(A)^a = MN = M'N' = |A| = |app(A)| = |M|$.

Case $A = xN_0$. Then, $A^a = (x, N_0^a)$ and $|A| = x|N_0|$. On the one hand, $x = x$. On the other hand, $N_0^a = |N_0|$, by IH1.

Case $A = (\lambda x.M_0)N_0$. Similar.

Case $A = A_0N_0$. Then, $|A| = |A_0||N_0|$. Let $A_0^a = (M_1, M_2)$. Then, $A^a = (M_1M_2, N_0^a)$. We want $M_1M_2 = |A_0|$ and $N_0^a = |N_0|$. The latter follows by IH1. As to the former, let $|A_0| = M_1'M_2'$. Then, by IH2, $M_1 = M_1'$ and $M_2 = M_2'$. Then, $M_1M_2 = M_1'M_2' = |A_0|$. ∎

We now check that $(\_)^n$ and $(\_)^a$ are indeed mutually inverse.

Proposition 32 *$M^{na} = M$ and $(M, N)^{na} = (M, N)$, all M and (M, N) in λ.*

Proof: Two proofs are possible. The first is a direct proof, by simultaneous induction on M and (M, N). The second, which we do next, uses the fact that $\mathcal{N}$ and $|\_|$ are mutually inverse.

M^{na} is $|\mathcal{N}(M)|$ by Propositions 29 and 31, and the latter is M by Proposition 21. As to the second assertion, let $A = (M, N)^n$. By Proposition 29, $A = \mathcal{N}(M)@\mathcal{N}(N)$. Now, by Proposition 31, there are M' and N' such that $A^a = (M', N')$ and $|A| = M'N'$. On the other hand,

$$\begin{aligned} |A| &= |\mathcal{N}(M)@\mathcal{N}(N)| \\ &= |\mathcal{N}(M)||\mathcal{N}(N)|, \text{ by Lemma 37,} \\ &= MN, \text{ by Proposition 21.} \end{aligned}$$

Therefore, $MN = |A| = M'N'$ and thus $M = M'$ and $N = N'$. Finally, $(M, N)^{na} = A^a = (M', N') = (M, N)$. ■

Proposition 33 *$M^{an} = M$ and $A^{an} = A$, all M and A in $\lambda\mathcal{N}$.*

Proof: Again, two proofs are possible. The first is a direct proof, by simultaneous induction on M and A. The second, which we do next, uses the fact that $\mathcal{N}$ and $|\_|$ are mutually inverse.

M^{an} is $\mathcal{N}|M|$ by Propositions 29 and 31, and the latter is M by Proposition 20. As to the second assertion, by Proposition 31, there are M, N such that $A^a = (M, N)$ and $|A| = MN$. Hence, $A^{an} = (M, N)^n = \mathcal{N}(M)@\mathcal{N}(N)$, by Proposition 29. On the other hand,

$$\begin{aligned} app(A) &= \mathcal{N}|A|, \text{ by Proposition 20} \\ &= \mathcal{N}(MN) \\ &= app(\mathcal{N}(M)@\mathcal{N}(N)), \text{ by def. of } \mathcal{N}. \end{aligned}$$

Hence, $app(A) = app(\mathcal{N}(M)@\mathcal{N}(N))$ and thus $A = \mathcal{N}(M)@\mathcal{N}(N)$. Therefore, both A^{an} and A are $\mathcal{N}(M)@\mathcal{N}(N)$. Thus $A^{an} = A$. ■

Definition 15 *Mapping $\mathcal{P}^{-1}$ is denoted $\mathcal{Q}$.*

Hence $\mathcal{Q} = (\_)^a \circ \Theta$. We will now prove an explicit definition of $\mathcal{Q}$.

Proposition 34 *The inverse of Prawitz's mapping is the mapping $\mathcal{Q} : \lambda\mathcal{P} \to \lambda$ defined by:*

$$\begin{array}{rcl}
\mathcal{Q}x & = & x \\
\mathcal{Q}(\lambda x.t) & = & \lambda x.\mathcal{Q}t \\
\mathcal{Q}(x(u \cdot l)) & = & \mathcal{Q}'(x, \mathcal{Q}u, l) \\
\mathcal{Q}((\lambda x.t)(u \cdot l)) & = & \mathcal{Q}'(\lambda x.\mathcal{Q}t, \mathcal{Q}u, l) \\
 & & \\
\mathcal{Q}'(M, N, []) & = & MN \\
\mathcal{Q}'(M, N, u :: l) & = & \mathcal{Q}'(MN, \mathcal{Q}u, l).
\end{array}$$

Proof: Again $\mathcal{Q}'(M, N, l)$ stands for $\mathcal{Q}'((M, N), l)$. In this proof we let $\mathcal{Q}$ denote the mapping defined by the above recursive definition. We prove $(\Theta t)^a = \mathcal{Q}t$ and

$$\Theta'(A, l)^a = let\ (M, N)\ be\ A^a\ in\ \mathcal{Q}'(M, N, l) \tag{5.6}$$

by simultaneous induction on t and l, with induction hypotheses referred to by IH1 and IH2, respectively .

Cases $t = x$ and $t = \lambda x.t_0$: straightforward.

Case $t = x(u \cdot l)$:

$$\begin{array}{rcl}
(\Theta t)^a & = & \Theta(x(u \cdot l))^a \\
 & = & \Theta'(x\Theta u, l)^a, \text{ by def. of } \Theta, \\
 & = & let\ (M, N)\ be\ (x\Theta u)^a\ in\ \mathcal{Q}'(M, N, l), \text{ by IH2,}
\end{array}$$

$$\begin{aligned}
&= \mathit{let}\ (M,N)\ \mathit{be}\ (x,(\Theta u)^a)\ \mathit{in}\ \mathcal{Q}'(M,N,l), \text{ by def. of } (_)^a,\\
&= \mathit{let}\ (M,N)\ \mathit{be}\ (x,\mathcal{Q}u)\ \mathit{in}\ \mathcal{Q}'(M,N,l), \text{ by IH1},\\
&= \mathcal{Q}'(x,\mathcal{Q}u,l)\\
&= \mathcal{Q}(x(u\cdot l)), \text{ by def. of } \mathcal{Q},\\
&= \mathcal{Q}(t)\ .
\end{aligned}$$

Case $t = (\lambda x.t_0)(u \cdot l)$: similar.
Case $l = []$:

$$\begin{aligned}
\Theta'(A,l)^a &= \Theta'(A,[])^a\\
&= \mathit{app}(A)^a, \text{ by def. of } \Theta,\\
&= \mathit{let}\,(M,N)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M,N,[]), \text{ by def. of } (_)^a,\\
&= \mathit{let}\,(M,N)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M,N,l)\ .
\end{aligned}$$

Case $l = u_0 :: l_0$:

$$\begin{aligned}
&\Theta'(A,l)^a\\
=\ &\Theta'(A,u_0 :: l_0)^a\\
=\ &\Theta'(A\Theta u_0,l_0)^a, \text{ by def. of } \Theta,\\
=\ &\mathit{let}\ (M,N)\ \mathit{be}\ (A\Theta u_0)^a\ \mathit{in}\ \mathcal{Q}'(M,N,l_0), \text{ by IH2},\\
=\ &\mathit{let}\ (M,N)\ \mathit{be}\ (\mathit{let}\ (M_1,M_2)\ \mathit{be}\ A^a\ \mathit{in}\ (M_1M_2,(\Theta u_0)^a)\ \mathit{in}\ \mathcal{Q}'(M,N,l_0),\\
&\text{by def. of } (_)^a,\\
=\ &\mathit{let}\ (M_1,M_2)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M_1M_2,(\Theta u_0)^a,l_0)\\
=\ &\mathit{let}\ (M_1,M_2)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M_1M_2,Qu_0,l_0), \text{ by IH1},\\
=\ &\mathit{let}\ (M_1,M_2)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M_1,M_2,u_0 :: l_0), \text{ by def. of } \mathcal{Q},\\
=\ &\mathit{let}\ (M_1,M_2)\ \mathit{be}\ A^a\ \mathit{in}\ \mathcal{Q}'(M_1,M_2,l)\ .
\end{aligned}$$

■

It is revealing that mapping $\mathcal{Q}$, as defined in Proposition 34, could have been shown to be the inverse of $\mathcal{P}$ in the same way as we proved that Θ is the inverse of Ψ. Indeed, one proves $\mathcal{Q} \circ \mathcal{P} = id$ and $\mathcal{Q} \circ \mathcal{P}' = \mathcal{Q}'$ by a simultaneous induction similar to the one we find in the proof of Proposition 24; and one proves $\mathcal{P} \circ \mathcal{Q} = id$ and $\mathcal{P} \circ \mathcal{Q}' = \mathcal{P}'$ by a simultaneous induction similar to the one we find in the proof of Proposition 25. The parallel between $\mathcal{P}$ and $\mathcal{Q}$, on the one hand, and Ψ and Θ, on the other hand, is quite tight. We recapitulate the situation in the following diagram:

$$\begin{tikzcd}
 & \lambda \arrow[dl, shift right, "\mathcal{P}"'] \arrow[dd, shift left, "(_)^n"] \\
\lambda\mathcal{P} \arrow[ur, shift right, "\mathcal{Q}"'] \arrow[dr, shift left, "\Theta"] & \\
 & \lambda\mathcal{N} \arrow[ul, shift left, "\Psi"] \arrow[uu, shift left, "(_)^a"]
\end{tikzcd}$$

Let us look again at Definition 12, which allowed this parallel between $\mathcal{P}$ and Ψ. Another presentation of the same inductive definition of the set of λ-terms is

$$\begin{array}{rcl} T & ::= & x \,|\, \lambda x.t \,|\, app(A) \\ A & ::= & (x, T) \,|\, (\lambda x.t, T) \,|\, (app(A), N) \end{array} \tag{5.7}$$

This may be obtained by unfolding T_1 in

$$\begin{array}{rcl} T & ::= & x \,|\, \lambda x.t \,|\, app(A) \\ A & ::= & (T_1, T_2) \end{array}$$

and this, in turn, is just the usual syntax of the λ-calculus

$$T ::= x \,|\, \lambda x.t \,|\, app(T_1, T_2) \ .$$

Hence, every A in (5.7) and every Ap in Definition 12 may be seen as an application with *look ahead*.

As promised, the difference between (5.7) and $\lambda\mathcal{N}$

$$\begin{array}{rcl} M, N & ::= & x \,|\, \lambda x.t \,|\, app(A) \\ A & ::= & xN \,|\, (\lambda x.M)N \,|\, AN \end{array}$$

is best seen by comparing how $\mathcal{P}$ and Ψ translate $VN_1...N_k$, where V is a variable or a λ-abstraction and $k \geq 1$. That is, we compare how they turn a main branch "upside down".

$\mathcal{P}$ calls $\mathcal{P}'((M, N_k), [])$, for some M, and now a decision has to be taken as to what to do with N_k. If $k > 1$, then $M = M'N_{k-1}$, for some M', and N_k is, so to speak, pushed on top of the second argument. The computation continues with $\mathcal{P}'((M', N_{k-1}), \mathcal{P}N_k :: [])$. If $k = 1$, then $M = V$ and $\mathcal{P}'$ does not produce another occurrence of constructor ::, instead it either returns $x(u{\cdot}l)$ or $(\lambda x.t)(u{\cdot}l)$, according to whether V is a variable or a λ-abstraction. Now, how does $\mathcal{P}$ make up its mind? The constructor (M, N_k) *per se* does not tell anything. $\mathcal{P}'$ has to check whether M is an application or some V, that is $\mathcal{P}'$ has to look ahead.

As to Ψ, the situation is different. $\Psi'(A, [])$ is called, for some A. But now the topmost constructor of A tells everything Ψ' needs to decide the dilemma above. If $k > 1$, A is of the form $A'N_k$ and Ψ' immediately knows (without checking what A' is) that this application is not a value application. The computation resumes with $\Psi'(A', \Psi N_k :: [])$. If $k = 1$, then A is of the form xN_k or $(\lambda x.M)N_k$ and some $x(u \cdot l)$ or $(\lambda x.t)(u \cdot l)$ is returned.

It seems that the dilemma $\mathcal{P}'$ and Ψ' are faced with is whether the application they have to translate is value or not. The true dilemma is slightly more general: it is whether the application they have to translate is a *head* application or not and the distinction between $\mathcal{P}$ and Ψ is that, while (M, N) does not tell $\mathcal{P}'$ this information, A does tell Ψ'.

In order to see this, recall again the process of turning the main branch upside down. The main branch contains k instances of the elimination rule. Each of these instances, except the topmost one, corresponds, in the resulting derivation, to an instance of Herbelin's left rule (the Lft rule, or constructor ::). The topmost one either corresponds to a $Left$ inference (the constructor $x(u \cdot l)$), when the topmost formula of the main branch is an assumption, or corresponds to a cut (the constructor $(\lambda x.t)(u \cdot l)$), when the topmost formula is the conclusion of an introduction (recall that in $\lambda\mathcal{P}h$, $x(u \cdot l)$ and $(\lambda x.t)(u \cdot l)$ are particular cases of *head*-cut $t(u \cdot l)$). Therefore, *the k instances of the elimination rule are not of the same kind, from the point of view of sequent calculus.* There is a distinction between the topmost one, which we call the *head* instance (hence the terminology head application), and the remaining instances, which we call *tail* instances (and which correspond to *tail* applications).

The difference between λ and $\lambda\mathcal{N}$ becomes conspicuous. In $\lambda\mathcal{N}$, the distinction between a head and a tail application is built-in and reflects the distinction between the two kinds of constructors they correspond to in $\lambda\mathcal{P}$. This is the nature of $\lambda\mathcal{N}$.

With this understanding of $\lambda\mathcal{N}$, we can re-interpret $(\_)^n$ and $(\_)^a$. Observe how $(M, N)^n$ is defined as a different kind of application (head or tail), according to what kind of term M is. Conversely, A^a forgets the kind of application A is and always returns a (M, N). Moreover, since $\mathcal{P} = \Psi \circ (\_)^n$, Prawitz's mapping can be implemented as a two-pass translation. The first goes through the proof and classifies each instance of elimination as head or tail. The second turns main branches upside down without looking ahead.

5.4 Mapping $\mathcal{Q}$

In this section we define and establish the properties of a mapping $\mathcal{Q}$ from $\lambda\mathcal{P}h$ to λ that extends the inverse of $\mathcal{P}$. This mapping $\mathcal{Q}$ is important for two main reasons: (1) it embodies part of the computational interpretation of $\lambda\mathcal{P}h$ (and, in particular $\lambda\mathcal{P}$) in the style of Curien and Herbelin [Curien and Herbelin, 2000]. Indeed, when one reads $\lambda\mathcal{P}h$-terms as λ-terms, it is mapping $\mathcal{Q}$ that is being applied. This will be seen in Chapter 7. (2) $\mathcal{Q}$ is nothing else but the traditional assignment φ [Prawitz, 1965, Zucker, 1974] of natural deduction proofs (or λ-terms) to sequent calculus, when sequent calculus is restricted to the canonical fragment. This will be seen at the end of this section.

We also study the difference between $\lambda\mathcal{G}$ and $\lambda\mathcal{P}$ as subsystems of $\lambda\mathcal{P}h$, and, in particular, the difference between $(\_)^-$ and f, the projections from $\lambda\mathcal{P}h$ to $\lambda\mathcal{P}$ and $\lambda\mathcal{G}$, respectively. Of course, f is basically the same as $\mathcal{Q}$.

Properties of $\mathcal{Q} : \lambda\mathcal{P} \to \lambda$

We start by proving some properties of $\mathcal{Q}$ (the inverse of $\mathcal{P}$), namely how it maps *subst*, *insert* and *append* of $\lambda\mathcal{P}$. Direct proofs could be provided from the definition of $\mathcal{Q}$ contained in Proposition 34. Nevertheless, we will show how to reuse similar properties of Θ proved before (Corollaries 18 and 19), having in mind that $\mathcal{Q}$ is the composition of Θ with $(\_)^a$. In view of Propositions 29 and 31, we will freely shift between $\mathcal{N}$ and $(\_)^n$ and between $|\_|$ and $(\_)^a$.

We start by giving a more manageable characterisation of $\mathcal{Q}'(M, N, l)$ than that of (5.6).

Lemma 45 *$\mathcal{Q}'(M, N, l) = \Theta'(M^n@N^n, l)^a$, for all M, N in λ, all l in $\lambda\mathcal{P}$.*

Proof: Observe that

$$\begin{aligned} |M^n@N^n| &= |M^n||N^n|, \text{ by Lemma 37,} \\ &= M^{na}N^{na} \\ &= MN \ . \end{aligned}$$

Therefore, by Proposition 31,

$$(M^n@N^n)^a = (M, N) \ , \tag{5.8}$$

and

$$\begin{aligned} \Theta'(M^n@N^n, l) &= \mathit{let}\ (M_0, N_0)\ \mathit{be}\ (M^n@N^n)^a\ \mathit{in}\ \mathcal{Q}'(M_0, N_0, l), \text{ by (5.6),} \\ &= \mathit{let}\ (M_0, N_0)\ \mathit{be}\ (M, N)\ \mathit{in}\ \mathcal{Q}'(M_0, N_0, l), \text{ by (5.8),} \\ &= \mathcal{Q}'(M, N, l) \ . \end{aligned}$$

■

Lemma 46

1. $\mathcal{Q}(insert(u,l,t)) = \mathcal{Q}'(\mathcal{Q}t, \mathcal{Q}u, l)$, *for all* t, u, l *in* $\lambda\mathcal{P}$.
2. $\mathcal{Q}'(M, N, append(l, u' :: l')) = \mathcal{Q}'(\mathcal{Q}'(M, N, l), u', l')$, *for all* M, N *in* λ, *all* u', l, l' *in* $\lambda\mathcal{P}$.

Proof: 1.

$$\begin{array}{rcl}
\mathcal{Q}(insert(u,l,t)) & = & \Theta(insert(u,l,t))^a \\
 & = & \Theta'(\Theta t @ \Theta u, l)^a, \text{ by Corollary 18,} \\
 & = & \Theta'((\Theta t)^{an} @ (\Theta u)^{an}, l)^a \\
 & = & \Theta'((\mathcal{Q}t)^n @ (\mathcal{Q}u)^n, l)^a \\
 & = & \mathcal{Q}'(\mathcal{Q}t, \mathcal{Q}u, l), \text{ by Lemma 45.}
\end{array}$$

2.

$$\begin{array}{rl}
 & \mathcal{Q}'(M, N, append(l, u' :: l')) \\
= & \Theta'(M^n @ N^n, append(l, u' :: l'))^a, \text{ by Lemma 45,} \\
= & \Theta'(\Theta'(M^n @ N^n, l) @ \Theta u', l')^a, \text{ by Corollary 18,} \\
= & \Theta'(\Theta'(M^n @ N^n, l)^{an} @ (\Theta u')^{an}, l')^a \\
= & \Theta'(\mathcal{Q}'(M, N, l)^n @ (\mathcal{Q}u')^n, l')^a, \text{ by Lemma 45,} \\
= & \mathcal{Q}'(\mathcal{Q}'(M, N, l), \mathcal{Q}u', l'), \text{ by Lemma 45.}
\end{array}$$

■

Lemma 47

1. $\mathcal{Q}(subst(u,x,t)) = \mathcal{Q}(t)[\mathcal{Q}(u)/x]$, *for all* u, t *in* $\lambda\mathcal{P}$.

2. *$\mathcal{Q}'(M[\mathcal{Q}u/x], N[\mathcal{Q}u/x], subst(u,x,l)) = \mathcal{Q}'(M,N,l)[\mathcal{Q}u/x]$, for all M, N in λ, all u, l in $\lambda\mathcal{P}$.*

Proof: 1. Since $\mathcal{Q}$ is the composition of Θ and $|\_|$, it follows from Corollary 19 and Lemma 38.

2.

$$
\begin{aligned}
& \mathcal{Q}'(M[\mathcal{Q}u/x], N[\mathcal{Q}u/x], subst(u,x,l)) \\
= \; & \Theta'(M[\mathcal{Q}u/x]^n @ N[\mathcal{Q}u/x]^n, subst(u,x,l))^a, \text{ by Lemma 45} \\
= \; & \Theta'(M^n[\Theta u/x] @ N^n[\Theta u/x], subst(u,x,l))^a, \text{ by (*) below,} \\
= \; & \Theta'((M^n @ N^n)[\Theta u/x], subst(u,x,l))^a, \text{ by (**) below,} \\
= \; & (\Theta'(M^n @ N^n, l)[\Theta u/x])^a, \text{ by Corollary 19,} \\
= \; & \Theta'(M^n @ N^n, l)^a[(\Theta u)^a/x], \text{ by Lemma 38,} \\
= \; & \mathcal{Q}'(M,N,l)[(\Theta u)^a/x], \text{ by Lemma 45.}
\end{aligned}
$$

(*) For all M_0 in λ,

$$
\begin{aligned}
(M_0[\mathcal{Q}u/x])^n &= (M_0[(\Theta u)^a/x])^n \\
&= M_0^n[(\Theta u)^{an}/x], \text{ by Corollary 14,} \\
&= M_0^n[\Theta u/x] \ .
\end{aligned}
$$

(**) For all M_1, M_2, N in $\lambda\mathcal{N}$, $(M_1 @ M_2)[N/x] = M_1[N/x] @ M_2[N/x]$. This follows by a straightforward case analysis of M_1. ■

Two subsystems of $\lambda\mathcal{P}h$

Observe the situation

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[dd, "(_)^- = \downarrow_h"'] & & \\
& \lambda\mathcal{G} \arrow[ul, "i"'] \arrow[dl, "(_)^-"'] & \lambda \arrow[l, "\mathcal{G}"'] \arrow[dll, "\mathcal{P}"] \\
\lambda\mathcal{P} & &
\end{tikzcd}$$

Calculi $\lambda\mathcal{G}$ and $\lambda\mathcal{P}$ are, respectively, Gentzen's and Prawitz's isomorphic copies of λ as a sequent calculus. They are also subsystems of $\lambda\mathcal{P}h$. In the following we explain the differences between the two copies of λ by explaining the differences between them while subsystems of $\lambda\mathcal{P}h$.

Recall that terms in $\lambda\mathcal{P}h$ are defined by

$$\begin{array}{rcl} t, u, v & ::= & x \,|\, \lambda x.t \,|\, t(u \cdot l) \\ l & ::= & [] \,|u :: l \end{array}$$

and that: (1) terms in $\lambda\mathcal{G}$ are those of $\lambda\mathcal{P}h$ where :: does not occur; (2) terms in $\lambda\mathcal{P}$ are those of $\lambda\mathcal{P}h$ such that t in $t(u \cdot l)$ is always some x or some $\lambda x.t_0$. The difference between the two syntaxes is best seen when one tries to write down an applicative term. In $\lambda\mathcal{G}$ this is done by means of iterated cuts: $t[u_1][u_2]...[u_k]$ (recall that $t[u]$ abbreviates $t(u \cdot []))$. In $\lambda\mathcal{P}$, provided t itself is not an applicative term and, therefore, is some value, the applicative term is written $t(u_1 \cdot [u_2, ..., u_k])$. In terms of $\lambda\mathcal{P}h$, the application to the first argument is always a cut, but then there are two ways of expressing application to further arguments: either by further cuts or by ::.

Mapping $(\_)^-$ from $\lambda\mathcal{G}$ to $\lambda\mathcal{P}$ amounts to the unfolding of $t[u_1][u_2]...[u_k]$ as $t(u_1 \cdot [u_2, ..., u_k])$. It is generalised by mapping $(\_)^-$ from $\lambda\mathcal{P}h$ to $\lambda\mathcal{P}$, which is determined by the clause $(t(u \cdot l))^- = insert(u^-, l^-, t^-)$, and is the same as the normal form mapping $\downarrow_h$.

Conversely, the inverse of $(\_)^-$, that goes from $\lambda\mathcal{P}$ to $\lambda\mathcal{G}$ and which we will denote by $(\_)^+$, folds $t(u_1 \cdot [u_2, ..., u_k])$ as $t[u_1][u_2]...[u_k]$. From Theorem 7 and $\mathcal{Q} = \mathcal{P}^{-1}$, the following is immediate.

Lemma 48 *The inverse of* $(\_)^- : \lambda\mathcal{G} \to \lambda\mathcal{P}$ *is* $\mathcal{G} \circ \mathcal{Q}$.

Therefore, the two mappings from $\lambda\mathcal{P}$

$$\begin{tikzcd}
 & \lambda\mathcal{G} & \lambda \arrow[l, "\mathcal{G}"'] \\
\lambda\mathcal{P} \arrow[ur, "(_)^+"] \arrow[urr, "\mathcal{Q}"'] & &
\end{tikzcd}$$

are the same up to $\mathcal{G}$, which, in turn, is simply a rephrasing of λ-terms with syntax $t ::= x | \lambda x.t | t[t']$. Hence, having in mind the definition of $\mathcal{Q}$, it is clear that $(\_)^+$ is defined by

$$\begin{array}{rcl}
x^+ & = & x \\
(\lambda x.t)^+ & = & \lambda x.t^+ \\
(x(u \cdot l))^+ & = & (x, u^+, l)^+ \\
(\lambda x.t)(u \cdot l))^+ & = & (\lambda x.t^+, u^+, l)^+ \\
 & & \\
(t_1, t_2, [])^+ & = & t_1[t_2] \\
(t_1, t_2, u :: l)^+ & = & (t_1[t_2], u^+, l)^+
\end{array}$$

where the ternary operator $(t_1, t_2, l)^+$ is defined for all t_1, t_2 in $\lambda\mathcal{G}$ and l in $\lambda\mathcal{P}$. Of course, a direct proof that this $(\_)^+$ is really $\mathcal{G} \circ \mathcal{Q}$ is possible. One proves

$$\mathcal{G}^{-1}(t^+) = \mathcal{Q}t \ , \tag{5.9}$$

for all t in $\lambda\mathcal{P}$, and

$$\mathcal{G}^{-1}((\mathcal{G}M, \mathcal{G}N, l)^+) = \mathcal{Q}'(M, N, l) \ , \tag{5.10}$$

for all M, N in λ and l in $\lambda\mathcal{P}$, by simultaneous induction on t and l.

We propose next a mapping for folding cuts in $\lambda\mathcal{P}h$.

Definition 16 *The mapping $f : \lambda\mathcal{P}h \to \lambda\mathcal{G}$ is defined by:*

$$\begin{array}{rcl} fx & = & x \\ f(\lambda x.t) & = & \lambda x.ft \\ f(t(u \cdot l)) & = & f'(ft, fu, l) \end{array}$$

$$\begin{array}{rcl} f'(t_1, t_2, []) & = & t_1[t_2] \\ f'(t_1, t_2, u :: l) & = & f'(t_1[t_2], fu, l) \end{array}$$

Observe that $f'(t_1, t_2, l)$ is defined for all t_1, t_2 in $\lambda\mathcal{G}$ and l in $\lambda\mathcal{P}h$. Our goal now is to show that f generalises $(\_)^+$, that is, $ft = t^+$, for all t in $\lambda\mathcal{P}$. This will follow from $ft = (\downarrow_h (t))^+$, which we prove next. We start with a rephrasing of part 1. of Lemma 46.

Corollary 23 *$(insert(u, l, t))^+ = (t^+, u^+, l)^+$, all t, u, l in $\lambda\mathcal{P}$.*

Proposition 35

1. *$ft = (\downarrow_h (t))^+$, for all t in $\lambda\mathcal{P}h$.*
2. *$f'(t, u, l) = (t, u, \downarrow_h (l))^+$, for all t, u in $\lambda\mathcal{G}$ and l in $\lambda\mathcal{P}h$.*

Proof: By simultaneous induction on t and l, with induction hypotheses IH1 and IH2, respectively . In this proof, we write ht and hl instead of $\downarrow_h (t)$ and $\downarrow_h (l)$. Below, when we justify an equality with "$h = (\_)^-$", we mean that we are using the definition of $(\_)^- : \lambda\mathcal{P}h \to \lambda\mathcal{P}$ and the fact that the latter is the same as $\downarrow_h$.

Cases $t = x$ and $t = \lambda x.t_0$: straightforward.

Case $t = t_0(u_0 \cdot l_0)$:

$$\begin{aligned}
ft &= f(t_0(u_0 \cdot l_0)) \\
&= f'(ft_0, fu_0, l_0), \text{ by def. of } f, \\
&= f'((ht_0)^+, (hu_0)^+, l_0), \text{ by IH1}, \\
&= ((ht_0)^+, (hu_0)^+, hl_0)^+, \text{ by IH2}, \\
&= (insert(hu_0, hl_0, ht_0))^+, \text{ by Corollary 23}, \\
&= (h(t_0(u_0 \cdot l_0)))^+, \text{ as } h = (_)^-.
\end{aligned}$$

Case $l = []$:

$$\begin{aligned}
f'(t, u, l) &= f'(t, u, []) \\
&= t[u], \text{ by def. of } f, \\
&= (t, u, [])^+, \text{ by def. of } (_)^+, \\
&= (t, u, h[])^+, \text{ as } h = (_)^-, \\
&= (t, u, hl)^+ \quad .
\end{aligned}$$

Case $l = u_0 :: l_0$:

$$\begin{aligned}
f'(t, u, l) &= f'(t, u, u_0 :: l_0) \\
&= f'(t[u], fu_0, l_0), \text{ by def. of } f, \\
&= (t[u], fu_0, h(l_0))^+, \text{ by IH2}, \\
&= (t[u], (hu_0)^+, h(l_0))^+, \text{ by IH1}, \\
&= (t, u, hu_0 :: hl_0)^+, \text{ by def. of } (_)^+, \\
&= (t, u, h(u_0 :: l_0))^+, \text{ as } h = (_)^-, \\
&= (t, u, hl)^+ \quad .
\end{aligned}$$

■

Therefore, f is a projection, because it is the composition of a projection with an isomorphism.

Corollary 24 *$ft = t^+$, all t in $\lambda\mathcal{P}$.*

Proof: Let $t \in \lambda\mathcal{P}$. Then $ft = (\downarrow_h (t))^+$, by Proposition 35. But $\downarrow_h (t) = t$, because $t \in \lambda\mathcal{P}$ and by Lemma 16. Hence $ft = t^+$. ■

Thus, f is an extension of $(\_)^+$ to $\lambda\mathcal{P}h$.

Corollary 25 *$ft = t$, all t in $\lambda\mathcal{G}$.*

Proof: Let $t \in \lambda\mathcal{G}$. For emphasis, let $i(t)$ be t seen as a $\lambda\mathcal{P}h$ term. Then $ft = (\downarrow_h (i(t)))^+$, by Proposition 35. But $\downarrow_h (i(t)) = t^-$ (here $(\_)^-$ is the mapping with domain in $\lambda\mathcal{G}$) and, thus, $ft = t^{-+} = t$. ■

It is also immediate that $\downarrow_h (t) = (ft)^-$, for all t in $\lambda\mathcal{P}h$.

Let us sum up in a diagram the situation regarding Gentzen's and Prawitz's subsystems of $\lambda\mathcal{P}h$:

$$\begin{tikzcd}
 & \lambda\mathcal{P}h \arrow[dl, "\downarrow_h = (_)^-"'] \arrow[dr, "(_)^+ = f"] & \\
\lambda\mathcal{P} \arrow[rr, shift left, "(_)^+"] & & \lambda\mathcal{G} \arrow[ll, shift left, "(_)^-"]
\end{tikzcd}$$

The bridge between $\lambda\mathcal{P}$ and $\lambda\mathcal{G}$ is a pair of mutually inverse mappings $(\_)^+$ and $(\_)^-$. So to speak, the former performs folding whereas the latter performs unfolding of cuts. The projection $\downarrow_h$ is an extension of the unfolding map, whereas the projection f is an extension of the folding one. The fact that $ft = t$, for t in $\lambda\mathcal{G}$, can be seen as saying that terms in $\lambda\mathcal{G}$ are fully folded and, similarly, terms in $\lambda\mathcal{G}$ can be seen as fully unfolded.

However, the situation is absolutely asymmetric w.r.t cut elimination. Projection $(\_)^-$ is the normal form mapping w.r.t. reduction rule h of $\lambda\mathcal{P}h$, that is, $\lambda\mathcal{P}h$ is internally conservative over $\lambda\mathcal{P}$. Moreover, h is necessary for the simulation in $\lambda\mathcal{P}h$ of cut elimination of $\lambda\mathcal{P}$. Therefore, unfolding (which is, of course, another name for left permutation) is a part of cut elimination in both $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$. All this fails for folding. $\lambda\mathcal{P}h$ is conservative, but not internally conservative, over $\lambda\mathcal{G}$, and β-reduction in the latter is just $\beta1$-reduction in the former. Hence, even if we added to $\lambda\mathcal{P}h$ a reduction rule for folding (a suggestion is $t(u \cdot (v :: l)) \to t[u](v \cdot l))$, this rule would remain unnecessary for the simulation of $\lambda\mathcal{G}$.

In the following chapter we will show that a similar asymmetry exists, in the "natural deduction side", between λ and $\lambda\mathcal{N}$, this time w.r.t. a suitably generalised notion of normalisation.

Mapping $\mathcal{Q} : \lambda\mathcal{P}h \to \lambda$

Consider again the diagram

$$
\begin{tikzcd}
\lambda\mathcal{P}h \arrow[dr, "f"] \arrow[dd, "\downarrow_h = (\_)^-"'] & & \\
& \lambda\mathcal{G} \arrow[r, "\mathcal{G}^{-1}"] & \lambda \\
\lambda\mathcal{P} \arrow[ur, "(\_)^+"] \arrow[urr, "\mathcal{Q}_0"'] & &
\end{tikzcd}
$$

For the moment, we denote by $\mathcal{Q}_0$ mapping $\mathcal{Q} : \lambda\mathcal{P} \to \lambda$. This is so because we want to define a mapping $\mathcal{Q} : \lambda\mathcal{P}h \to \lambda$ that will turn out to be an extension of $\mathcal{Q}_0$. $\mathcal{Q}$ is just the following rephrasing of f

$$
\begin{array}{rcl}
\mathcal{Q}x & = & x \\
\mathcal{Q}(\lambda x.t) & = & \lambda x.\mathcal{Q}t \\
\mathcal{Q}(t(u \cdot l)) & = & \mathcal{Q}'(\mathcal{Q}t, \mathcal{Q}u, l)
\end{array}
$$

$$
\mathcal{Q}'(M_1, M_2, []) = M_1 M_2
$$

$$\mathcal{Q}'(M_1, M_2, u :: l) \;=\; \mathcal{Q}'(M_1 M_2, \mathcal{Q}u, l)$$

That is, we have

$$\begin{aligned} \mathcal{Q}t &= \mathcal{G}^{-1}(f(t)) & (5.11)\\ \mathcal{Q}'(M,N,l) &= \mathcal{G}^{-1}(f'(\mathcal{G}M,\mathcal{G}N,l))\ , & (5.12) \end{aligned}$$

for all M, N in λ, all t, l in $\lambda\mathcal{P}h$. $\mathcal{Q}'(M_1, M_2, l)$ is defined for all M_1, M_2 in λ (or, equivalently, for all $(M_1, M_2) \in Ap$ - recall Definition 12) and all l in $\lambda\mathcal{P}h$.

Lemma 49

1. *$\mathcal{Q}t = \mathcal{Q}_0(\downarrow_h (t))$, all t in $\lambda\mathcal{P}h$.*

2. *$\mathcal{Q}'(M,N,l) = \mathcal{Q}'_0(M,N,\downarrow_h (l))$, all M, N in λ, all l in $\lambda\mathcal{P}h$.*

Proof: 1.

$$\begin{aligned} \mathcal{Q}t &= \mathcal{G}^{-1}(f(t)), \text{ by (5.11)},\\ &= \mathcal{G}^{-1}(\downarrow_h (t)^+), \text{ by Proposition 35},\\ &= \mathcal{Q}_0(\downarrow_h (t)), \text{ by (5.9)}. \end{aligned}$$

2.

$$\begin{aligned} \mathcal{Q}'(M,N,l) &= \mathcal{G}^{-1}(f'(GM,GN,l)), \text{ by (5.12)},\\ &= \mathcal{G}^{-1}(GM,GN,\downarrow_h (l))^+, \text{ by Proposition 35},\\ &= \mathcal{Q}'_0(M,N,\downarrow_h (l)), \text{ by (5.10)}. \end{aligned}$$

■

Since $\downarrow_h$ collapses $\to_h$ steps ($t \to_h t'$ implies $\downarrow_h (t) =\downarrow_h (t')$), so does $\mathcal{Q}$. Moreover, $\mathcal{Q}$ is indeed an extension of $\mathcal{Q}_0$ because $\downarrow_h (t) = t$ and $\downarrow_h (l) = l$ when t and l are in $\lambda\mathcal{P}$.

Let us see how $\mathcal{Q}$ interprets operators *subst* and *append* of $\lambda\mathcal{P}h$. We want to "lift" Lemmas 46 and 47.

Lemma 50 *$\mathcal{Q}'(M, N, append(l, u' :: l')) = \mathcal{Q}'(\mathcal{Q}'(M, N, l), \mathcal{Q}u', l')$, for all M, N in λ, all u', l' in $\lambda\mathcal{P}h$.*

Proof: From Lemmas 46, 17 and 49. ■

Lemma 51

1. *$\mathcal{Q}(subst(u, x, t)) = \mathcal{Q}(t)[\mathcal{Q}(u)/x]$, all u, t in $\lambda\mathcal{P}h$.*
2. *$\mathcal{Q}'(M[\mathcal{Q}u/x], N[\mathcal{Q}u/x], subst(u, x, l)) = \mathcal{Q}'(M, N, l)[\mathcal{Q}u/x]$, all M, N in λ, all u, l in $\lambda\mathcal{P}h$.*

Proof: From Lemmas 47, 18 and 49. ■

The return of φ

Consider a generic term notation for sequent calculus, as the one employed in Chapter 2. The traditional assignment of λ-terms to sequent calculus proofs is defined by

$$\varphi(\mathsf{Ax}(x)) = x \tag{5.13}$$

$$\varphi(\mathsf{R}((x)L)) = \lambda x.\varphi(L) \tag{5.14}$$

$$\varphi(\mathsf{L}(x, L_1, (y)L_2) = \varphi(L_2)[x\varphi(L_1)/y] \tag{5.15}$$

$$\varphi(\mathsf{Cut}(L_1, (x)L_2) = \varphi(L_2)[\varphi(L_1)/x] \tag{5.16}$$

Consider a right-permuted cut $\mathsf{Cut}(L_1, (x)\mathsf{L}(x, L_{21}, (y)L_{22}))$. The right cut formula is main and linear. Hence, $x \notin L_{21}, L_{22}$. This cut is mapped as follows:

$$\begin{array}{rl} & \varphi(\mathsf{Cut}(L_1, (x)\mathsf{L}(x, L_{21}, (y)L_{22})) \\ = & \varphi(\mathsf{L}(x, L_{21}, (y)L_{22}))[\varphi(L_1)/x] \\ = & \varphi(L_{22})[x\varphi(L_{21})/y][\varphi(L_1)/x] \\ = & \varphi(L_{22})[\varphi(L_1)\varphi(L_{21})/y], \text{ as } x \notin L_{21}, L_{22}. \end{array} \tag{5.17}$$

Now we are ready to restrict φ to $\lambda\mathcal{P}h$.

$$\begin{array}{rcl}
\varphi(x) & = & x \\
\varphi(\lambda x.t) & = & \lambda x.\varphi(t) \\
\varphi(t(u \cdot l)) & = & \varphi(l,w)[\varphi(t)\varphi(u)/w],\ w \text{ fresh}
\end{array}$$

$$\begin{array}{rcl}
\varphi([],w) & = & w \\
\varphi(u :: l,w) & = & \varphi(l,z)[w\varphi(u)/z],\ z \text{ fresh}
\end{array}$$

The clause for $t(u \cdot l)$ is in accordance with (5.17). Since formulas in the stoup do not have a variable, we also have to pass a fresh variable when mapping a list l, as in $\varphi(l,w)$. The clauses for x and $[]$ are as (5.13), the clause for $::$ as (5.15).

Proposition 36

1. *$\mathcal{Q}t = \varphi t$, all t in $\lambda\mathcal{P}h$.*

2. *$\mathcal{Q}'(M,N,l) = \varphi(l,w)[MN/w]$, w fresh, all M,N in λ, all l in $\lambda\mathcal{P}h$.*

Proof: By simultaneous induction on t and l, with induction hypotheses IH1 and IH2 . Cases $t = x$ and $t = \lambda x.t_0$ are straightforward.

Case $t = t_0(u_0 \cdot l_0)$.

$$\begin{array}{rcl}
\mathcal{Q}t & = & \mathcal{Q}(t_0(u_0 \cdot l_0)) \\
& = & \mathcal{Q}'(\mathcal{Q}(t_0), \mathcal{Q}(u_0), l_0) \\
& = & \varphi(l,w)[\mathcal{Q}(t_0)\mathcal{Q}(u_0)/w], \text{ by IH2,} \\
& = & \varphi(l,w)[\varphi(t_0)\varphi(u_0)/w], \text{ by IH1,} \\
& = & \varphi(t_0(u_0 \cdot l_0)) \\
& = & \varphi t\ .
\end{array}$$

Case $l = []$.

$$\begin{aligned}
\mathcal{Q}'(M, N, l) &= \mathcal{Q}'(M, N, []) \\
&= MN \\
&= w[MN/w] \\
&= \varphi([], w)[MN/w] \\
&= \varphi(l, w)[MN/w]
\end{aligned}$$

Case $l = u_0 :: l_0$.

$$\begin{aligned}
\mathcal{Q}'(M, N, l) &= \mathcal{Q}'(M, N, u_0 :: l_0) \\
&= \mathcal{Q}'(MN, \mathcal{Q}(u_0), l_0) \\
&= \varphi(l_0, z)[(MN)\mathcal{Q}(u_0)/z], \text{ by IH2,} \\
&= \varphi(l_0, z)[(MN)\varphi(u_0)/z], \text{ by IH1} \\
&= \varphi(l_0, z)[w\varphi(u_0)/z][MN/w], \text{ as } w \text{ is fresh,} \\
&= \varphi(u_0 :: l_0, w)[MN/w] \\
&= \varphi(l, w)[MN/w] \ .
\end{aligned}$$

■

Chapter 6

Extensions of natural deduction

In this chapter, we construct systems of natural deduction that stand for $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}h\mathtt{x}$ as $\lambda\mathcal{N}$ stands for $\lambda\mathcal{P}$. The idea of built-in distinction between head and tail elimination is the key ingredient to obtain the counterpart of $\lambda\mathcal{P}h$. Then, we obtain the counterpart of $\lambda\mathcal{P}h\mathtt{x}$ by making substitution explicit.

We also discuss at length the logical status of explicit substitutions, having in mind that they serve as counterpart to explicit right permutation of cuts.

Finally, we extract some conceptual and taxonomical consequences of the fact that usual λ-calculus is in the intersection of two degenerate fragments, the ::-free fragment of sequent calculus, and the tail-application-free fragment of extended natural deduction.

6.1 Head and tail eliminations

In Section 5.3, by an analysis of Prawitz's mapping, we observed that, from the point of view of sequent calculus, not all instances of the elimination rule have the same nature, and that indeed the distinction between head and tail applications built in the syntax of $\lambda\mathcal{N}$ (and which does not exist in λ) matched the distinction in $\lambda\mathcal{P}$ between a cut $t(u \cdot l)$ (more precisely, a left inference $x(u \cdot l)$ or a key-cut $(\lambda x.t)(u \cdot l)$) and a Herbelin left inference :: . Moreover, recall that Ψ is an isomorphism between βi in $\lambda\mathcal{N}$ and $\lambda\mathcal{P}$, for $i = 1, 2$, whereas β-reduction in λ, when rephrased by $\mathcal{G}$, corresponds solely to $\beta 1$-reduction in $\lambda\mathcal{P}h$. Therefore, $\lambda\mathcal{N}$

also improves over λ in matching certain aspects of cut-elimination in $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$, namely left permutation of cuts. The natural challenge is then to define an extension of $\lambda\mathcal{N}$ that stands for this calculus as $\lambda\mathcal{P}h$ stands for $\lambda\mathcal{P}$ and, in particular, that captures in the natural deduction "space" general head-cuts $t(u \cdot l)$ and their complete left permutation.

Such calculus, named $\lambda\mathcal{N}h$, exists and produces in the natural deduction "space of calculi" a perfect counterpart to the situation involving the calculi $\lambda\mathcal{P}h$, $\lambda\mathcal{P}$ and $\lambda\mathcal{G}$ (recall Section 5.4), as illustrated in the following diagram.

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[dd, "\downarrow_h"'] & & \lambda\mathcal{N}h \arrow[ll, "\Psi"'] \arrow[dd, "\downarrow_h"'] & \\
& \lambda\mathcal{G} \arrow[ul, "i"'] \arrow[dl, "(_)^-"] & & \lambda \arrow[ll, "\mathcal{G}"'] \arrow[ul, "\iota"'] \arrow[dl, "\mathcal{N}"] \\
\lambda\mathcal{P} & & \lambda\mathcal{N} \arrow[ll, "\Psi"'] &
\end{tikzcd} \tag{6.1}$$

The $\lambda\mathcal{N}h$-calculus

The $\lambda\mathcal{N}h$-calculus is defined in Table 6.1. Typing rules are in Table 6.2. Notice again the separation between *applicative terms* $app(A)$ and *applications* $A \in Apps$.

Corresponding to full head-cuts $t(u \cdot l)$ (in which t is not necessarily a variable or λ-abstraction), we generalise applications of $\lambda\mathcal{N}$ by

$$A ::= xN \,|\, (\lambda x.M)N \,|\, app(A)N \,|\, AN \ \ ,$$

which will simply be defined as

$$A ::= MN \,|\, AN \ \ .$$

Table 6.1: The $\lambda\mathcal{N}h$-calculus

$$
\begin{array}{rrcl}
(Terms) & M, N & ::= & x \mid \lambda x.M \mid app(A) \\
(Apps) & A & ::= & MN \mid AN
\end{array}
$$

$$
\begin{array}{rrcl}
(\beta 1) & app((\lambda x.M)N) & \to & M[N/x] \\
(\beta 2) & ((\lambda x.M)N)N' & \to & M[N/x]N' \\
(h) & app(A)N & \to & AN
\end{array}
$$

where

$$
\begin{array}{rcl}
x[N/x] & = & N \\
y[N/x] & = & y,\ y \neq x \\
(\lambda y.M)[N/x] & = & \lambda y.M[N/x] \\
(app(A))[N/x] & = & app(A[N/x])
\end{array}
$$

$$
\begin{array}{rcl}
(M_1 M_2)[N/x] & = & M_1[N/x]M_2[N/x] \\
(AM)[N/x] & = & A[N/x]M[N/x]
\end{array}
$$

Table 6.2: Typing rules for $\lambda\mathcal{N}h$

$$Var\,\frac{}{\Gamma, x : B \vdash x : B} \qquad Intro\,\frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x.M : B \supset C}\, x \notin \Gamma$$

$$App\,\frac{\Gamma \vdash A : B}{\Gamma \vdash app(A) : B}$$

$$HdElim\,\frac{\Gamma \vdash M : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash MN : C}$$

$$TailElim\,\frac{\Gamma \vdash A : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash AN : C}$$

An application of the form MN (resp. AN) is called a *head* (resp. *tail*) application. The corresponding typing rules are named head and tail elimination, respectively. It is clear that tail application corresponds to the constructor :: or, in other terms, tail elimination correspond to Herbelin's left rule. As to head application, it subsumes the constructors xN and $(\lambda x.M)N$ of $\lambda\mathcal{N}$ in the same way as head-cut in $\lambda\mathcal{P}h$ subsumes the constructors $x(u \cdot l)$ and $(\lambda x.t)(u \cdot l)$ of $\lambda\mathcal{P}$.

In $\lambda\mathcal{N}h$ we still have to split β in two cases but we no longer need @ in $(\beta 2)$ (as we did in $\lambda\mathcal{N}$). In addition to $(\beta 1)$ and $(\beta 2)$, we only require the simple

$$(h) \quad app(A)N \quad \rightarrow \quad AN \ ,$$

which will play the role of counterpart to complete left permutation of cuts. A h-redex is an head application that is not a value application.

Definition 17 (Compatible closure) *Given a pair R of binary relations, the first on $Terms$ and the second on $Apps$, the compatible closure $\rightarrow_R$ is the least pair of relations $\rightarrow$, the first on $Terms$ and containing the first relation of R, the second on $Apps$ and containing the second relation of R, closed under:*

$$Intro\,\frac{M \to M'}{\lambda x.M \to \lambda x.M'} \qquad App\,\frac{A \to A'}{app(A) \to app(A')}$$

$$HdElim1\,\frac{M \to M'}{MN \to M'N} \qquad HdElim2\,\frac{N \to N'}{MN \to MN'}$$

$$TailElim1\,\frac{A \to A'}{AN \to A'N} \qquad TailElim2\,\frac{N \to N'}{AN \to AN'}$$

For instance, for defining $\to_\beta$, take $R = (\beta 1, \beta 2)$ in Definition 17. That is, in $\lambda\mathcal{N}h$ we also set (as we did in $\lambda\mathcal{N}$)

$$\beta = (\beta 1, \beta 2) \ .$$

One can again define $\to_{\beta 1}$ (resp. $\to_{\beta 2}$) by taking $R = (\beta 1, \emptyset)$ (resp. $R = (\emptyset, \beta 2)$), or define $\to_h$ by taking $R = (\emptyset, h)$.

There is an injection ι between λ and $\lambda\mathcal{N}h$ that simply sends MN in λ to $app(MN)$ in $\lambda\mathcal{N}h$.

$$\begin{array}{rcl} \iota x & = & x \\ \iota(\lambda x.M) & = & \lambda x.\iota M \\ \iota(MN) & = & app(\iota(M)\iota(N)) \end{array}$$

It is immediately seen to be correct.

Proposition 37 *If* $\Gamma \vdash M : B$ *in* λ*, then* $\Gamma \vdash \iota(M) : B$*.*

Proof: By induction on M. Only case $M = M_0N_0$ matters. Suppose λ derives $\Gamma \vdash M_0N_0 : B$. Then λ derives $\Gamma \vdash M_0 : C \supset B$ and $\Gamma \vdash N_0 : C$, for some C. By induction hypothesis, there are derivations in $\lambda\mathcal{N}h$ of $\Gamma \vdash \iota(M_0) : C \supset B$ and $\Gamma \vdash \iota(N_0) : C$. Conclude with

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash \iota(M_0) : C \supset B\end{array} \qquad \begin{array}{c}\vdots\\ \Gamma \vdash \iota(N_0) : C\end{array}}{\Gamma \vdash \iota(M_0)\iota(N_0) : B}\,HdElim}{\Gamma \vdash app(\iota(M_0)\iota(N_0)) : B}\,App$$

■

Now a situation very similar to that of mapping $\mathcal{G} : \lambda \rightarrow \lambda\mathcal{P}h$ is now observed in mapping $\iota : \lambda \rightarrow \lambda\mathcal{N}h$. The range of ι is the tail-application-free fragment of $\lambda\mathcal{N}h$, which is

$$\begin{array}{rcl} M, N & ::= & x \mid \lambda x.M \mid app(A) \\ A & ::= & MN \end{array}$$

or, equivalently,

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

This is very much like λ-calculus, but with application written $app(MN)$. Actually, this constructor is typed by

$$\frac{\Gamma \vdash M : C \supset B \quad \Gamma \vdash N : C}{\Gamma \vdash app(MN) : B}$$

which should be seen as an abbreviation of

$$\cfrac{\cfrac{\Gamma \vdash M : C \supset B \quad \Gamma \vdash N : C}{\Gamma \vdash MN : B}\ HdElim}{\Gamma \vdash app(MN) : B}\ App$$

As to reduction, only rule $\beta 1$

$$app((\lambda x.M)N) \quad \rightarrow \quad M[N/x]$$

makes sense in this fragment, as both $\beta 2$ and h require tail elimination. Now, the calculation

$$\begin{array}{rcl} (app(M'N'))[N/x] & = & app((M'N')[N/x]) \\ & = & app(M'[N/x]N'[N/x]) \end{array}$$

shows two things. First, that the tail-application-free fragment is indeed a fragment of of $\lambda\mathcal{N}h$ because it is closed for substitution and $\beta 1$. Second, that the restriction of substitution of $\lambda\mathcal{N}h$ to this fragment behaves exactly as λ-calculus' substitution.

Therefore, the tail-application-free fragment of $\lambda\mathcal{N}h$ is simply a rephrasing of λ, where application is written $app(MN)$. Furthermore, mapping ι is trivially an isomorphism between λ and this fragment. This justifies the following terminology.

Definition 18 *The tail-application-free fragment of $\lambda\mathcal{N}h$ is denoted $\lambda\iota$.*

For simplicity, in the remainder of this section we will not separate λ and $\lambda\iota$ and, therefore, we will regard λ as being the tail-application-free fragment of $\lambda\mathcal{N}h$. We will come back to $\lambda\iota$ in the next section.

There is a simple mapping $(\_)^-$ from $\lambda\mathcal{N}h$ to $\lambda\mathcal{N}$, defined in Table 6.3. The idea is an adaptation of mapping $\mathcal{N}$ from λ to $\lambda\mathcal{N}$. When mapping a head application MN (where M may be some $app(A'))$ down to some A in $\lambda\mathcal{N}$, we make use of operator @ for assuring that the head application of A is a value application. The following is simple.

Proposition 38 *$\mathcal{N}(M) = (\iota(M))^-$, for all M in λ.*

Indeed, mapping from λ to $\lambda\mathcal{N}$, being inductively determined by the clause $\mathcal{N}(MN) = app(\mathcal{N}(M)@\mathcal{N}(N))$, is the composition of ι, that sends each MN to $app(\iota(M)\iota(N))$, with the mapping from $\lambda\mathcal{N}h$ to $\lambda\mathcal{N}$ inductively determined by $(MN)^- = M^-@N^-$.

One sees at once that, in $\lambda\mathcal{N}h$, $\rightarrow_h$ is terminating and weakly confluent. Therefore, $\rightarrow_h$ is confluent. It is also easy to see that $(\_)^-$ from $\lambda\mathcal{N}h$ to $\lambda\mathcal{N}$ is nothing but the normal form mapping $\downarrow_h$ w.r.t. h in $\lambda\mathcal{N}h$.

Proposition 39 *$M^- = \downarrow_h (M)$, all M in $\lambda\mathcal{N}h$.*

Proof: One proves by simultaneous induction on M and A that $M \rightarrow_h^* M^-$ and M^- is h-normal, and that $A \rightarrow_h^* A^-$ and A^- is h-normal, for all M and A in $\lambda\mathcal{N}h$.

$$\begin{array}{rcl} x^- & = & x \\ (\lambda x.M)^- & = & \lambda x.M^- \\ app(A)^- & = & app(A^-) \\ & & \\ (MN)^- & = & M^- @ N^- \\ (AN)^- & = & A^- N^- \end{array}$$

Table 6.3: From $\lambda\mathcal{N}h$ to $\lambda\mathcal{N}$

All cases are straightforward, the only interesting one being $A = app(A_0)M_0$. Then, $A^- = app(A_0^-)@M_0^- = A_0^- M_0^-$. By the induction hypotheses, both A_0^- and M_0^- are h-normal and, since a tail application cannot be a h-redex, A^- itself cannot be a h-redex. Therefore A^- is h-normal. Moreover $A = app(A_0)M_0 \rightarrow_h A_0 M_0$ and now, by the induction hypotheses, $A_0 M_0 \rightarrow_h^* A_0^- M_0^- = A^-$. ■

Mappings Ψ and Θ are naturally extended to $\lambda\mathcal{N}h$ and $\lambda\mathcal{P}h$ by

$$\Psi'(MN, l) = \Psi M(\Psi N \cdot l)$$

and

$$\Theta(t(u \cdot l)) = \Theta'(\Theta t \Theta u, l) \ .$$

This definition is coherent with the former definition of $\Psi'(xN, l)$ and $\Psi'((\lambda x.M)N, l)$, on the one hand, and with the former definition of $\Theta'(x(u \cdot l))$ and $\Theta'((\lambda x.t))(u \cdot l))$, on the other hand. Therefore, the following is immediate.

Proposition 40 *If i denotes both the inclusion of $\lambda\mathcal{N}$ in $\lambda\mathcal{N}h$ and of $\lambda\mathcal{P}$ in $\lambda\mathcal{P}h$, then $\Psi(i(M)) = i(\Psi M)$ and $\Theta(i(t)) = i(\Theta t)$, for all M in $\lambda\mathcal{N}$ and t in $\lambda\mathcal{P}$.*

Proposition 41 (Correctness of Ψ)

1. *If* $\lambda\mathcal{N}h$ *derives* $\Gamma \vdash M : B$ *then* $\lambda\mathcal{P}h$ *derives* $\Gamma; - \vdash \Psi(M) : B$.

2. *If* $\lambda\mathcal{N}h$ *derives* $\Gamma \vdash A : C$ *and* $\lambda\mathcal{P}h$ *derives* $\Gamma; C \vdash l : B$ *then* $\lambda\mathcal{P}h$ *derives* $\Gamma; - \vdash \Psi'(A, l) : B$.

Proof: By the same simultaneous induction as in the proof of Proposition 22. Instead of cases $A = xN$ and $A = (\lambda x.M)N$, one has

Case $A = MN$. Then there are π_1', π_1'', D such that π_2 has the form

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots \\ \Gamma \vdash M : D \supset C\end{array} \qquad \begin{array}{c}\pi_1''\\ \vdots \\ \Gamma \vdash N : D\end{array}}{\Gamma \vdash MN : C}\ HdElim$$

Let π_3 be a derivation in $\lambda\mathcal{P}h$ of $\Gamma; C \vdash l : B$. Since $\Psi'(A, l) = \Psi'(MN, l) = \Psi M(\Psi N \cdot l)$, we want a derivation π_2^* of $\Gamma; - \vdash \Psi M(\Psi N \cdot l) : B$. Take π_2^* as

$$\dfrac{\begin{array}{c}\pi_1^{+}\\ \vdots \\ \Gamma; - \vdash \Psi(M) : D \supset C\end{array} \qquad \begin{array}{c}\pi_1^{++}\\ \vdots \\ \Gamma; - \vdash \Psi(N) : D\end{array} \qquad \begin{array}{c}\pi_3\\ \vdots \\ \Gamma; C \vdash l : B\end{array}}{\Gamma; - \vdash \Psi M(\Psi N \cdot l) : B}\ HeadCut$$

where π_1^+ and π_1^{++} are given by IH1. ■

Proposition 42 (Correctness of Θ)

1. *If* $\lambda\mathcal{P}h$ *derives* $\Gamma; - \vdash t : B$ *then* $\lambda\mathcal{N}h$ *derives* $\Gamma \vdash \Theta t : B$.

2. *If* $\lambda\mathcal{N}h$ *derives* $\Gamma \vdash A : C$ *and* $\lambda\mathcal{P}h$ *derives* $\Gamma; C \vdash l : B$ *then* $\lambda\mathcal{N}h$ *derives* $\Gamma \vdash \Theta'(A, l) : B$.

Proof: By the same simultaneous induction as in the proof of Proposition 23. Instead of case $t = x(u \cdot l)$ and $t = (\lambda x.t')(u' \cdot l')$, one has

Case $t = t'(u' \cdot l')$. Then there are $\pi_1', \pi_1'', \pi_2', D, E$ such that π_1 has the form

$$
HeadCut \, \frac{\begin{array}{c}\pi_1' \\ \vdots \\ \Gamma; - \vdash t' : D \supset E\end{array} \qquad \begin{array}{c}\pi_1'' \\ \vdots \\ \Gamma; - \vdash u' : D\end{array} \qquad \begin{array}{c}\pi_2' \\ \vdots \\ \Gamma; E \vdash l' : B\end{array}}{\Gamma; - \vdash t'(u' \cdot l') : B}
$$

Since $\Theta t = \Theta(t'(u' \cdot l')) = \Theta'(\Theta t' \Theta u', l')$, we want a derivation π_1^* of $\Gamma \vdash \Theta'(\Theta t' \Theta u', l') : B$. Observe that

$$
HdElim \, \frac{\begin{array}{c}\pi_1^+ \\ \vdots \\ \Gamma \vdash \Theta t' : D \supset E\end{array} \qquad \begin{array}{c}\pi_1^{++} \\ \vdots \\ \Gamma \vdash \Theta u' : D\end{array}}{\Gamma \vdash \Theta t' \Theta u' : E}
$$

is a derivation in $\lambda\mathcal{N}h$ of $\Gamma \vdash \Theta t' \Theta u' : E$, where π_1^+ and π_1^{++} are given by IH1. Hence, by IH2, there is a derivation π_2^+ of $\Gamma \vdash \Theta'(\Theta t' \Theta u', l') : B$. Take $\pi_1^* = \pi_2^+$. ∎

Now, Ψ keeps transforming applications into head-cuts by turning them "upside down". In particular, if the application consists of a single head application, like $app(MN)$, the result is a head-cut with a single argument, namely $\Psi M[\Psi N]$, as witnessed by the calculation $\Psi(app(MN)) = \Psi'(MN, []) = \Psi M(\Psi N :: []) = \Psi M[\Psi N]$. But this is how $\mathcal{G}$ would have translated the application MN in λ. Having in mind that λ may be embedded into $\lambda\mathcal{N}h$, this means that Ψ is coherent with $\mathcal{G}$. This is expressed in the following proposition.

Proposition 43 $\Psi(\iota(M)) = \mathcal{G}(M)$, *for all* M *in* λ.

Proof: By induction on M. The only interesting case is $M = M_0 N_0$. On the one hand, $\mathcal{G}M = \mathcal{G}M_0[\mathcal{G}N_0]$. On the other hand,

$$
\begin{array}{rcl}
\Psi(\iota(M)) & = & \Psi(app(\iota(M_0)\iota(N_0))), \text{ by def. of } \iota, \\
 & = & \Psi(\iota(M_0))[\Psi(\iota(N_0))], \text{ by the discussion above,} \\
 & = & \mathcal{G}M_0[\mathcal{G}N_0], \text{ by IH.}
\end{array}
$$

■

Thus mapping $\Psi : \lambda\mathcal{N}h \to \lambda\mathcal{P}h$ generalises both $\mathcal{G} : \lambda \to \lambda\mathcal{G}$ and $\Psi : \lambda\mathcal{N} \to \lambda\mathcal{P}$. Having in mind that the latter may be seen as Prawitz's mapping, when this is defined in $\lambda\mathcal{N}$ and not in λ, mapping Ψ from $\lambda\mathcal{N}h$ to $\lambda\mathcal{P}h$ is coherent with both Gentzen's and Prawitz's way of mapping natural deduction to sequent calculus: (1) with the former because a head elimination (all eliminations in λ are head) is mapped to a head-cut (and head-cuts is what left inferences and key cuts of $\lambda\mathcal{P}$ are in $\lambda\mathcal{P}h$). (2) with the latter because tail eliminations are mapped to Herbelin left inferences (in accordance to Prawitz) and this necessarily agrees with Gentzen because Gentzen does not map tail eliminations.

Given that Ψ generalises $\mathcal{G}$, the following theorem generalises Theorem 7 and gives another commutative square in diagram (6.1), if one bears in mind that $(\_)^- =\downarrow_h$.

Theorem 10 $\Psi(M^-) = (\Psi M)^-$*, for all* M *in* $\lambda\mathcal{N}h$*.*

Proof: We prove the claim together with the claim that $\Psi'(A^-, l^-) = (\Psi'(A, l))^-$, for all A in $\lambda\mathcal{N}h$, by simultaneous induction on M and A, with induction hypotheses IH1 and IH2 respectively. The cases $M = x$ and $M = \lambda x.M_0$ are straightforward.

Case $M = app(A)$.

$$\begin{array}{rcl}
\Psi(M^-) & = & \Psi((app(A))^-) \\
& = & \Psi(app(A^-)), \text{ by def. of } (_)^-, \\
& = & \Psi'(A^-, []), \text{ by def. of } \Psi, \\
& = & (\Psi'(A, []))^-, \text{ by IH2 and } []^- = [], \\
& = & (\Psi(app(A)))^-, \text{ by def. of } \Psi, \\
& = & (\Psi M)^- \ .
\end{array}$$

Case $A = MN$.

$$
\begin{array}{rcl}
\Psi'(A^-, l^-) & = & \Psi'((MN)^-, l^-) \\
& = & \Psi'(M^- @ N^-, l^-), \text{ by def. of } (\_)^-, \\
& = & insert(\Psi(N^-), l^-, \Psi(M^-)), \text{ by Lemma 40}, \\
& = & insert((\Psi N)^-, l^-, (\Psi M)^-), \text{ by IH1}, \\
& = & (\Psi M(\Psi N :: l))^-, \text{ by def. of } (\_)^-, \\
& = & (\Psi'(MN, l))^-, \text{ by def. of } \Psi, \\
& = & (\Psi'(A, l))^- \ .
\end{array}
$$

Case $A = A_0 N_0$.

$$
\begin{array}{rcl}
\Psi'(A^-, l^-) & = & \Psi'((A_0 N_0)^-, l^-) \\
& = & \Psi'(A_0^- N_0^-, l^-), \text{ by def. of } (\_)^-, \\
& = & \Psi'(A_0^-, \Psi N_0^- :: l^-), \text{ by def. of } \Psi, \\
& = & \Psi'(A_0^-, (\Psi N_0)^- :: l^-), \text{ by IH1}, \\
& = & \Psi'(A_0^-, (\Psi N_0 :: l)^-), \text{ by def. of } (\_)^-, \\
& = & (\Psi'(A_0, \Psi N_0 :: l))^-, \text{ by IH2}, \\
& = & (\Psi'(A_0 N_0, l))^-, \text{ by def. of } \Psi, \\
& = & (\Psi'(A, l))^- \ .
\end{array}
$$

■

Now we get correctness of $(\_)^- : \lambda\mathcal{N}h \to \lambda\mathcal{N}$ as a corollary of correctness of $(\_)^- : \lambda\mathcal{P}h \to \lambda\mathcal{P}$

Corollary 26 *If $\lambda\mathcal{N}h$ derives $\Gamma \vdash M : A$, then $\lambda\mathcal{N}$ derives $\Gamma \vdash M^- : A$.*

Proof:

$$\lambda\mathcal{N}h \text{ derives } \Gamma \vdash M : A$$

$$\begin{array}{ll}
\text{implies that} & \lambda\mathcal{P}h \text{ derives } \Gamma; - \vdash \Psi M : A, \text{ by Proposition 41,} \\
\text{implies that} & \lambda\mathcal{P} \text{ derives } \Gamma; - \vdash (\Psi M)^- : A, \text{ by Proposition 4,} \\
\text{implies that} & \lambda\mathcal{P} \text{ derives } \Gamma; - \vdash \Psi(M^-) : A, \text{ by Theorem 10,} \\
\text{implies that} & \lambda\mathcal{N} \text{ derives } \Gamma \vdash \Theta\Psi(M^-) : A, \text{ by Proposition 23,} \\
\text{implies that} & \lambda\mathcal{N} \text{ derives } \Gamma \vdash M^- : A, \text{ by Proposition 24.}
\end{array}$$

■

Another isomorphism

We move to the proof of $\lambda\mathcal{P}h \cong \lambda\mathcal{N}h$, a generalisation of both the trivial $\lambda\mathcal{G} \cong \lambda$ and of $\lambda\mathcal{P} \cong \lambda\mathcal{N}$.

Proposition 44 *$\Theta \circ \Psi = id$ and $\Theta \circ \Psi' = \Theta'$.*

Proof: The proof is exactly as the proof of Proposition 24. We just do the new case.

Case $A = MN$:

$$\begin{array}{rcl}
\Theta\Psi'(A,l) & = & \Theta\Psi'(MN,l) \\
& = & \Theta(\Psi M(\Psi N \cdot l)) \\
& = & \Theta'((\Theta\Psi M)(\Theta\Psi N),l) \\
& = & \Theta'(MN,l), \text{ by IH1,} \\
& = & \Theta'(A,l) \ .
\end{array}$$

■

Proposition 45 *$\Psi \circ \Theta = id$ and $\Psi \circ \Theta' = \Psi'$.*

Proof: The proof is exactly as the proof of Proposition 25. We just do the new case.

Case $t = t_0(u_0 \cdot l_0)$:

$$
\begin{aligned}
\Psi\Theta t &= \Psi\Theta(t_0(u_0 \cdot l_0)) \\
&= \Psi\Theta'((\Theta t_0 \Theta u_0), l_0) \\
&= \Psi'((\Theta t_0 \Theta u_0), l_0), \text{ by IH2,} \\
&= \Psi\Theta t_0(\Psi\Theta u_0 \cdot l_0)) \\
&= t_0(u_0 \cdot l_0), \text{ by IH1} \\
&= t \ .
\end{aligned}
$$

■

Lemma 52

1. *$\Psi(M[N/x]) = subst(\Psi N, x, \Psi M)$, all M, N in $\lambda\mathcal{N}h$.*
2. *$\Psi'(A[N/x], subst(\Psi N, x, l)) = subst(\Psi N, x, \Psi'(A, l))$, all A, N in $\lambda\mathcal{N}h$ and all l in $\lambda\mathcal{P}h$.*

Proof: By the same simultaneous induction as in the proof of Lemma 41. All the cases stay unchanged, except that instead of cases $A = xM$ and $A = (\lambda y.M)M'$, we have case $A = M_0N_0$. Then, writing s for *subst*,

$$
\begin{aligned}
& s(\Psi N, x, \Psi'(A, l)) \\
= \ & s(\Psi N, x, \Psi'(M_0N_0, l)) \\
= \ & s(\Psi N, x, \Psi M_0(\Psi N_0 \cdot l)), \text{ by def. of } \Psi, \\
= \ & s(\Psi N, x, \Psi M_0)(s(\Psi N, x, \Psi N_0) \cdot s(\Psi N, x, l)), \text{ by def. of } subst, \\
= \ & \Psi(M_0[N/x])(\Psi(N_0[N/x] \cdot s(\Psi N, x, l))), \text{ by IH1,} \\
= \ & \Psi'(M_0[N/x]N_0[N/x], s(\Psi N, x, l)), \text{ by def. of } \Psi, \\
= \ & \Psi'((M_0N_0)[N/x], s(\Psi N, x, l)), \text{ by def. of } \_[N/x], \\
= \ & \Psi'(A[N/x], s(\Psi N, x, l)) \ .
\end{aligned}
$$

■

It turns out that the proof of this lemma becomes considerably simpler than the proof of Lemma 41 because substitution in $\lambda\mathcal{N}h$ does not call the operator @ and similarly *subst* in $\lambda\mathcal{P}h$ does not call the operator *insert*.

Corollary 27

1. $\Theta(subst(v, x, t)) = \Theta t[\Theta v/x]$, *for all* v, t *in* $\lambda\mathcal{P}h$.

2. $\Theta'(A[\Theta u/x], subst(u, x, t)) = \Theta'(A, l)[\Theta u/x]$, *for all* A *in* $\lambda\mathcal{N}h$, *all* u, t *in* $\lambda\mathcal{P}h$.

Proof: It follows from Lemma 52 and Propositions 45 and 44 in the same way as Corollary 19 follows from Lemma 41 and Propositions 25 and 24. ■

The following lemma and corollary may be seen as an adaptation to $\lambda\mathcal{P}h$ and $\lambda\mathcal{N}h$ of part 2. of Lemma 40 and of Corollary 17, respectively.

Lemma 53 $\Psi'(A, l)(u' \cdot l') \to_h \Psi'(A, append(l, u' :: l'))$, *for all* A *in* $\lambda\mathcal{N}h$, *all* u', l, l' *in* $\lambda\mathcal{P}h$.

Proof: By induction on A.

Case $A = MN$.

$$
\begin{array}{rcl}
\Psi'(A, l)(u' :: l') & = & \Psi'(MN, l)(u' \cdot l') \\
& = & (\Psi M(\Psi N \cdot l))(u' \cdot l'), \text{ by def. of } \Psi, \\
& \to_h & \Psi M(\Psi N \cdot append(l, u' :: l')) \\
& = & \Psi'(MN, append(l, u' :: l'), \text{ by def. of } \Psi, \\
& = & \Psi'(A, append(l, u' :: l') \ .
\end{array}
$$

Case $A = A'N'$.

$$
\begin{array}{rcl}
\Psi'(A,l)(u'::l') & = & \Psi'(A'N',l)(u'::l') \\
& = & \Psi'(A',\Psi N'::l)(u'::l'), \text{ by def. of } \Psi, \\
& \to_h & \Psi'(A',\mathit{append}(\Psi N'::l,u'::l')), \text{ by IH}, \\
& = & \Psi'(A',\Psi N'::\mathit{append}(l,u'::l')), \text{ by def. of } \mathit{append}, \\
& = & \Psi'(A'N',\mathit{append}(l,u'::l'), \text{ by def. of } \Psi, \\
& = & \Psi'(A,\mathit{append}(l,u'::l')\ .
\end{array}
$$

■

Corollary 28 *$\Psi(app(A))(u \cdot l) \to_h \Psi'(A, u :: l)$, for all A in $\lambda\mathcal{N}h$, all u, l in $\lambda\mathcal{P}h$.*

Proof: Immediate from Lemma 53, when $l = []$. ■

The following two lemmas are generalisations of Lemmas 42 and 43. Observe that in these lemmas what matters is not the notion of reduction R, but instead the definition of Ψ and Θ and the closure rules with which $\to_R$ is defined.

Lemma 54 *In $\lambda\mathcal{P}h$, if $l \to_R l'$, then $\Psi'(A,l) \to_R \Psi'(A,l')$ (for all A in $\lambda\mathcal{N}h$, $R \in \{\beta 1, \beta 2, h\}$).*

Proof: By an induction on A similar to that of Lemma 42. Case $A = A'N$ is as before, because $\to_R$ in $\lambda\mathcal{P}h$ is also closed under $Lft2$.

Case $A = MN$. $\Psi'(MN,l) = \Psi M(\Psi N \cdot l) \to_R \Psi M(\Psi N \cdot l') = \Psi'(MN,l)$, where the reduction step is by $l \to_R l'$ and closure of $\to_R$ in $\lambda\mathcal{P}h$ under $HdCut3$.
■

Lemma 55 *In $\lambda\mathcal{N}h$, if $A \to_R A'$, then $\Theta'(A,l) \to_R \Theta'(A',l)$ (for all l in $\lambda\mathcal{P}h$, $R \in \{\beta 1, \beta 2, h\}$).*

Proof: Identical to the proof of Lemma 43. ■

There are two ways of understanding next lemma. The first is by observing that the LHS and RHS, so to speak, of the h-step it claims are exactly the images under Θ of the LHS and RHS, respectively, of the h-step proved in Lemma 53. Then second is as an adaptation to $\lambda\mathcal{N}h$ of part 2. of Corollary 18.

Lemma 56 *$\Theta'(\Theta'(A,l)\Theta u',l') \to_h \Theta'(A, append(l, u' :: l'))$, for all A in $\lambda\mathcal{N}h$, all u', l, l' in $\lambda\mathcal{P}h$.*

Proof: By induction on l.

Case $l = []$.

$$\begin{array}{rcl}
\Theta'(\Theta'(A,l)\Theta u',l') & = & \Theta'(\Theta'(A,[])\Theta u',l') \\
 & = & \Theta'(app(A)\Theta u',l'), \text{ by def. of } \Theta, \\
 & \to_h & \Theta'(A\Theta u',l'), \text{ by Lemma 55}, \\
 & = & \Theta'(A, u' :: l'), \text{ by def. of } \Theta, \\
 & = & \Theta'(A, append([], u' :: l')), \text{ by def. of } append, \\
 & = & \Theta'(A, append(l, u' :: l')) \ .
\end{array}$$

Case $l = u_0 :: l_0$

$$\begin{array}{rcl}
\Theta'(\Theta'(A,l)\Theta u',l') & = & \Theta'(\Theta'(A,u_0 :: l_0)\Theta u',l') \\
 & = & \Theta'(\Theta'(A\Theta u_0,l_0)\Theta u',l'), \text{ by def. of } \Theta, \\
 & \to_h & \Theta'(A\Theta u_0, append(l_0, u' :: l')), \text{ by IH}, \\
 & = & \Theta'(A, u_0 :: append(l_0, u' :: l')), \text{ by def. of } \Theta, \\
 & = & \Theta'(A, append(u_0 :: l_0, u' :: l')), \text{ by def. of } append, \\
 & = & \Theta'(A, append(l, u' :: l')) \ .
\end{array}$$

■

Now the first half of the isomorphism.

Theorem 11 *Let $R \in \{\beta 1, \beta 2, h\}$. If $M \to_R M'$ in $\lambda\mathcal{N}h$ then $\Psi M \to_R \Psi M'$ in $\lambda\mathcal{P}h$.*

Proof: We prove the claim and also that

$$\text{if } A \to_R A' \text{ in } \lambda\mathcal{N}h, \text{ then } \Psi'(A,l) \to_R \Psi'(A',l) \text{ in } \lambda\mathcal{P}h, \text{ for all } l \text{ in } \lambda\mathcal{P}h,$$

by simultaneous induction, similar to the proof of Theorem 4, on $M \to_R M'$ and $A \to_R A'$, with induction hypotheses IH1 and IH2, respectively. Cases correspond to closure rules, according to Definition 17.

Case $\beta 1$: as in Theorem 4, but by Lemma 52, instead of Lemma 41.

Case $\beta 2$:

$$\begin{array}{rcl}
\Psi'(((\lambda x.M)N)N', l) & = & \Psi'(((\lambda x.M)N), \Psi N' :: l), \text{ by def of } \Psi, \\
& = & (\lambda x.\Psi M)(\Psi N \cdot (\Psi N' :: l)), \text{ by def of } \Psi, \\
& & \\
& \to_{\beta 2} & subst(\Psi N, x, \Psi M)(\Psi N' \cdot l) \\
& = & \Psi(M[N/x])(\Psi N' \cdot l), \text{ by Lemma 52}, \\
& = & \Psi'(M[N/x]N', l), \text{ by def of } \Psi.
\end{array}$$

Case h:

$$\begin{array}{rcl}
\Psi'(app(A)N, l) & = & \Psi(app(A))(\Psi N :: l), \text{ by def. of } \Psi, \\
& \to_h & \Psi'(A, \Psi N :: l), \text{ by Corollary 28}, \\
& = & \Psi'(AN, l), \text{ by def. of } \Psi.
\end{array}$$

Case *Intro*: As in Theorem 4, by closure of $\to_R$ in $\lambda\mathcal{P}h$ under *Right*.

Case *App*: As in Theorem 4.

Case *HdElim*1: Suppose $\Psi M \to_R \Psi M'$ (IH1).

$$\begin{array}{rcl} \Psi'(MN, l) & = & \Psi M(\Psi N \cdot l), \text{ by def. of } \Psi, \\ & \to_R & \Psi M'(\Psi N \cdot l) \ (*) \\ & = & \Psi'(M'N, l), \text{ by def. of } \Psi, \end{array}$$

where the reduction step is by IH1 and closure of $\to_R$ in $\lambda\mathcal{P}h$ under $HdCut1$.

Case $HdElim2$: Similarly, but by closure of $\to_R$ in $\lambda\mathcal{P}h$ under $HdCut2$.

Case $TailElim1$: As in Theorem 4.

Case $TailElim2$: As in Theorem 4, but by Lemma 54 and closure of $\to_R$ in $\lambda\mathcal{P}h$ under $Lft1$. ■

Finally, the second half of the isomorphism.

Theorem 12 *Let $R \in \{\beta 1, \beta 2, h\}$. If $t \to_R t'$ in $\lambda\mathcal{P}h$ then $\Theta t \to_R \Theta t'$ in $\lambda\mathcal{N}h$.*

Proof: We prove the claim and also that

if $l \to_R l'$ in $\lambda\mathcal{P}h$, then $\Theta'(A, l) \to_R \Theta(A, l')$ in $\lambda\mathcal{N}h$, for all A in $\lambda\mathcal{N}h$,

by simultaneous induction, similar to that of Theorem 5, on $t \to_R t'$ and $l \to_R l'$. Cases correspond to closure rules, according to Definition 6.

Case $\beta 1$: As in Theorem 5, but by Corollary 27 instead of Corollary 19.

Case $\beta 2$:

$$\begin{array}{rcl} \Theta((\lambda x.t)(v \cdot (u :: l))) & = & \Theta'((\lambda x.\Theta t)\Theta v, u :: l), \text{ by def. of } \Theta, \\ & = & \Theta'(((\lambda x.\Theta t)\Theta v)\Theta u, l), \text{ by def. of } \Theta, \\ & \to_{\beta 2} & \Theta'((\Theta t[\Theta v/x])\Theta u, l), \text{ by Lemma 55}, \\ & = & \Theta'(\Theta(subst(v, x, t))\Theta u, l), \text{ by Corollary 27}, \\ & = & \Theta(subst(v, x, t)(u :: l)) \ . \end{array}$$

Case h:

$$\begin{array}{rcl}\Theta((t(u\cdot l))(u'\cdot l')) & = & \Theta'(\Theta(t(u\cdot l))\Theta u', l'), \text{ by def. of } \Theta,\\ & = & \Theta'(\Theta'(\Theta t\Theta u, l)\Theta u', l'), \text{ by def. of } \Theta,\\ & \to_h & \Theta'(\Theta t\Theta u, \mathit{append}(l, u' :: l')), \text{ by Lemma 56},\\ & = & \Theta(t(u\cdot \mathit{append}(l, u' :: l'))), \text{ by def. of } \Theta.\end{array}$$

Case *Right*: As in Theorem 5, by closure of $\to_R$ in $\lambda\mathcal{N}h$ under *Intro*.

Case $HdCut1$: Suppose $\Theta t \to_R \Theta t'$ (IH1).

$$\begin{array}{rcl}\Theta(t(u\cdot l)) & = & \Theta'(\Theta t\Theta u, l), \text{ by def. of } \Theta,\\ & \to_R & \Theta'(\Theta t'\Theta u, l) \ (*)\\ & = & \Theta(t'(u\cdot l)), \text{ by def. of } \Theta,\end{array}$$

where the step (*) is by Lemma 55, IH1 and closure of $\to_R$ in $\lambda\mathcal{N}h$ under $HdElim1$.

Case $HdCut2$: Similarly, but by closure of $\to_R$ in $\lambda\mathcal{N}h$ under $HdElim2$.

Case $HdCut3$: Suppose $\Theta'(A, l) \to_R \Theta'(A, l')$, all A (IH2).

$$\begin{array}{rcl}\Theta(t(u :: l)) & = & \Theta'(\Theta t\Theta u, l), \text{ by def. of } \Theta,\\ & \to_R & \Theta'(\Theta t\Theta u, l'), \text{ by IH2}\\ & = & \Theta(t(u :: l')), \text{ by def. of } \Theta.\end{array}$$

■

Corollary 29 (Isomorphism) *Let* $R \in \{\beta 1, \beta 2, h\}$.

1. $M \to_R M'$ *in* $\lambda\mathcal{N}h$ *iff* $\Psi M \to_R \Psi M'$ *in* $\lambda\mathcal{P}h$.

2. $t \to_R t'$ *in* $\lambda\mathcal{P}h$ *iff* $\Theta t \to_R \Theta t'$ *in* $\lambda\mathcal{N}h$.

Corollary 30

1. *$\lambda\mathcal{N}h$ is confluent.*

2. *If M is typable in $\lambda\mathcal{N}h$, then M is strongly normalising.*

3. *$\lambda\mathcal{N}h$ satisfies subject reduction.*

Proof: Because these properties hold of $\lambda\mathcal{P}h$ and may be easily transferred from $\lambda\mathcal{P}h$ to $\lambda\mathcal{N}h$ with the help of Ψ and Θ. ■

We can, so to speak, reuse conservativeness of $\lambda\mathcal{P}h$ over $\lambda\mathcal{P}$ through isomorphisms Ψ and Θ.

Corollary 31 *$\lambda\mathcal{N}h$ is internally conservative over $\lambda\mathcal{N}$.*

Proof: Conservativeness is by

$$\begin{array}{ll} & M \to^* M' \text{ in } \lambda\mathcal{N} \\ \text{iff} & \Psi M \to^* \Psi M' \text{ in } \lambda\mathcal{P}, \text{ by Corollary 20,} \\ \text{iff} & i(\Psi M) \to^* i(\Psi M') \text{ in } \lambda\mathcal{P}h, \text{ as } \lambda\mathcal{P}h \text{ is conservative over } \lambda\mathcal{P}, \\ \text{iff} & \Psi(i(M)) \to^* \Psi(i(M')) \text{ in } \lambda\mathcal{P}h, \text{ by Proposition 40,} \\ \text{iff} & \Theta\Psi(i(M)) \to^* \Theta\Psi(i(M')) \text{ in } \lambda\mathcal{N}h, \text{ by Corollary 29,} \\ \text{iff} & i(M) \to^* i(M') \text{ in } \lambda\mathcal{N}h, \text{ as } \Theta \circ \Psi = id. \end{array}$$

Internal conservativeness is by Proposition 39. ■

6.2 Explicit substitutions

In the last section, we built in the natural deduction side a perfect counterpart to the relationship between $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$. Now we want to do the same for calculi $\lambda\mathcal{P}h\mathtt{x}$ and $\lambda\mathcal{P}h$. It turns out that the problem is now simpler. $\lambda\mathcal{P}h\mathtt{x}$ stands for $\lambda\mathcal{P}h$ as the $\lambda\mathtt{x}$-calculus [Rose, 1996b, Bloo, 1997] (a calculus of explicit substitutions) stands for λ. To illustrate this, we briefly recall $\lambda\mathtt{x}$.

The definition of $\lambda\mathtt{x}$ is given in Table 6.4. The calculus may be seen as a version of λ in which substitution is internalised. A new constructor, called explicit substitution and written $M\langle x := N\rangle$, is added and rule β is replaced by rule b, which, instead of calling substitution, generates an explicit substitution. Extra rules for the explicit, stepwise performance of substitution are included (rules $\mathtt{x}1, ..., \mathtt{x}4$). The typing rule for explicit substitution used here (and also in [Bloo, 1997]) is straightforward[1], in that we simply internalise the admissible rule for substitution in λ

$$\frac{\Gamma \vdash N : A \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash M[N/x] : B}\, x \notin \Gamma \ .$$

Now, in $\lambda\mathcal{P}h\mathtt{x}$, rules $\beta1$ and $\beta2$ of $\lambda\mathcal{P}h$ are replaced by rules $b1$ and $b2$ in which the call of operator *subst* (a substitution operator in $\lambda\mathcal{P}h$) is replaced by a mid cut, a new constructor of the calculus that acts as an explicit *subst*. The performance of *subst* is internalised in $\lambda\mathcal{P}h\mathtt{x}$ by means of new reduction rules $\mathtt{x}1, ..., \mathtt{x}4$.

We will come back later (see subsection "Completing the picture") to this analogy between how $\lambda\mathtt{x}$ stands for λ, on the one hand, and how $\lambda\mathcal{P}h\mathtt{x}$ stands for $\lambda\mathcal{P}h$, on the other hand. For the moment, as we said above, we are interested in finding the calculus that is the natural deduction counterpart to $\lambda\mathcal{P}h\mathtt{x}$. The obvious guess now is that such calculus is a version of $\lambda\mathcal{N}h$ in which substitution is made explicit.

Explicit substitutions for $\lambda\mathcal{N}h$

We define a version of $\lambda\mathcal{N}h$ with explicit substitutions, named $\lambda\mathcal{N}h\mathtt{x}$, in Table 6.5. Typing rules are in Table 6.6. The definition is straightforward. A constructor for substitution is added to the syntax and old reduction rules calling meta-substitution are replaced by similar rules generating explicit substitution. Rules for the stepwise performance of substitution are added. Similarly to $\lambda\mathcal{P}h\mathtt{x}$, an operation remains in the meta-language, namely the operation *sub* that dis-

[1] A slight variations of this typing rule has appeared in [di Cosmo and Kesner, 1997].

$$(Terms) \qquad M, N \quad ::= \quad x \mid \lambda x.M \mid MN \mid M\langle x := N\rangle$$

$$Var\,\frac{}{\Gamma, x : B \vdash x : B}$$

$$Intro\,\frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x.M : B \supset C}\, x \notin \Gamma \qquad Elim\,\frac{\Gamma \vdash M : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash MN : C}$$

$$ExSubst\,\frac{\Gamma, x : C \vdash M : B \quad \Gamma \vdash N : C}{\Gamma \vdash M\langle x := N\rangle : B}\, x \notin \Gamma$$

$$\begin{array}{llcl}
(b) & (\lambda x.M)N & \rightarrow & M\langle x := N\rangle \\
 & & & \\
(\mathtt{x1}) & x\langle x := N\rangle & = & N \\
(\mathtt{x2}) & y\langle x := N\rangle & = & y,\ y \neq x \\
(\mathtt{x3}) & (\lambda y.M)\langle x := N\rangle & = & \lambda y.M\langle x := N\rangle \\
(\mathtt{x4}) & (MM')\langle x := N\rangle & = & (M\langle x := N\rangle)(M'\langle x := N\rangle)
\end{array}$$

Table 6.4: The λx-calculus

Table 6.5: The $\lambda\mathcal{N}h\mathtt{x}$-calculus

$$
\begin{array}{rrcl}
(\mathit{Terms}) & M, N & ::= & x \mid \lambda x.M \mid \mathit{app}(A) \mid M\langle x := N\rangle \\
(\mathit{Apps}) & A & ::= & MN \mid AN
\end{array}
$$

$$
\begin{array}{rrcl}
(b1) & \mathit{app}((\lambda x.M)N) & \to & M\langle x := N\rangle \\
(b2) & ((\lambda x.M)N)N' & \to & M\langle x := N\rangle N' \\
(h) & \mathit{app}(A)N & \to & AN \\
\\
(\mathtt{x}1) & x\langle x := N\rangle & = & N \\
(\mathtt{x}2) & y\langle x := N\rangle & = & y,\ y \neq x \\
(\mathtt{x}3) & (\lambda y.M)\langle x := N\rangle & = & \lambda y.M\langle x := N\rangle \\
(\mathtt{x}4) & (\mathit{app}(A))\langle x := N\rangle & = & \mathit{app}(\mathit{sub}(N, x, A))
\end{array}
$$

where

$$
\begin{array}{rcl}
\mathit{sub}(N, x, M_1 M_2) & = & M_1\langle x := N\rangle M_2\langle x := N\rangle \\
\mathit{sub}(N, x, AM) & = & \mathit{sub}(N, x, A)M\langle x := N\rangle
\end{array}
$$

tributes a substitution through an application A. Constructor $M\langle x := N\rangle$ binds x in M. By variable convention, x does not occur in N.

Definition 19 (Compatible closure) *Given a pair R of binary relations, the first on Terms and the second on Apps, the compatible closure $\to_R$ is the least pair of relations $\to$, the first on Terms and containing the first relation of R, the second on Apps and containing the second relation of R, closed under:*

Table 6.6: Typing rules for $\lambda\mathcal{N}h\mathtt{x}$

$$Var\,\frac{}{\Gamma, x : B \vdash x : B} \qquad Intro\,\frac{\Gamma, x : B \vdash M : C}{\Gamma \vdash \lambda x.M : B \supset C}\, x \notin \Gamma$$

$$App\,\frac{\Gamma \vdash A : B}{\Gamma \vdash app(A) : B} \qquad ExSubst\,\frac{\Gamma, x : C \vdash M : B \quad \Gamma \vdash N : C}{\Gamma \vdash M\langle x := N\rangle : B}\, x \notin \Gamma$$

$$HdElim\,\frac{\Gamma \vdash M : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash MN : C}$$

$$TailElim\,\frac{\Gamma \vdash A : B \supset C \quad \Gamma \vdash N : B}{\Gamma \vdash AN : C}$$

$$Intro\,\frac{M \to M'}{\lambda x.M \to \lambda x.M'} \qquad App\,\frac{A \to A'}{app(A) \to app(A')}$$

$$ExSubst1\,\frac{M \to M'}{M\langle x := N\rangle \to M'\langle x := N\rangle} \qquad ExSubst2\,\frac{N \to N'}{M\langle x := N\rangle \to M\langle x := N'\rangle}$$

$$HdElim1\,\frac{M \to M'}{MN \to M'N} \qquad HdElim2\,\frac{N \to N'}{MN \to MN'}$$

$$TailElim1\,\frac{A \to A'}{AN \to A'N} \qquad TailElim2\,\frac{N \to N'}{AN \to AN'}$$

For instance, for defining $\to_b$, take $R = (b1, b2)$ in Definition 19. That is, in $\lambda\mathcal{N}h\mathtt{x}$ we set $b = (b1, b2)$. One can define $\to_{b1}$ (resp. $\to_{b2}$) by taking $R = (b1, \emptyset)$ (resp. $R = (\emptyset, b2)$), or define $\to_h$ by taking $R = (\emptyset, h)$. The definition of $\to_{\mathtt{x}i}$ ($i = 1, 2, 3, 4$) is by choosing $R = (\mathtt{x}i, \emptyset)$. We will also let

$$\mathtt{x} = \mathtt{x}1 \cup \mathtt{x}2 \cup \mathtt{x}3 \cup \mathtt{x}4$$

and, thus, $\to_{\mathtt{x}}$ is defined by taking $R = (\mathtt{x}, \emptyset)$.

$$
\begin{array}{rcl}
x^\flat & = & x \\
(\lambda x.M)^\flat & = & \lambda x.M^\flat \\
(app(A))^\flat & = & app(A^\flat) \\
(M\langle x := N\rangle)^\flat & = & M^\flat[N^\flat/x] \\
\\
(MN)^\flat & = & M^\flat N^\flat \\
(AN)^\flat & = & A^\flat N^\flat
\end{array}
$$

Table 6.7: From $\lambda\mathcal{N}h\mathtt{x}$ to $\lambda\mathcal{N}h$

By analogy with mapping $(\_)^\flat : \lambda\mathcal{P}h\mathtt{x} \to \lambda\mathcal{P}h$, there is a mapping $(\_)^\flat : \lambda\mathcal{N}h\mathtt{x} \to \lambda\mathcal{N}h$, defined in Table 6.7, that replaces explicit substitution by meta-substitution in $\lambda\mathcal{N}h$.

Mappings Ψ and Θ, between $\lambda\mathcal{N}h$ and $\lambda\mathcal{P}h$, are extended to mappings between $\lambda\mathcal{N}h\mathtt{x}$ and $\lambda\mathcal{P}h\mathtt{x}$ by

$$
\begin{array}{rcl}
\Psi(M\langle x := N\rangle) & = & \Psi M\{x := \Psi N\} \\
\Theta(t\{x := u\}) & = & \Theta t\langle x := \Theta u\rangle \ .
\end{array}
$$

Given the correctness of Ψ and Θ between $\lambda\mathcal{N}h$ and $\lambda\mathcal{P}h$, and given the typing rules for explicit substitutions and mid-cuts, the correctness of these extensions is routine.

Explicit substitutions are mapped to mid-cuts and *vice-versa*. The following proposition is even simpler than Proposition 40.

Proposition 46 *If i denotes both the inclusion of $\lambda\mathcal{N}h$ in $\lambda\mathcal{N}h\mathtt{x}$ and of $\lambda\mathcal{P}h$ in $\lambda\mathcal{P}h\mathtt{x}$, then $\Psi(i(M)) = i(\Psi M)$ and $\Theta(i(t)) = i(\Theta t)$, for all M in $\lambda\mathcal{N}h$ and t in $\lambda\mathcal{P}h$.*

Proposition 47 $(\Psi(M))^\flat = \Psi(M^\flat)$*, for all M in $\lambda\mathcal{N}h\mathtt{x}$.*

Proof: One proves the claim and also $(\Psi'(A, l))^\flat = \Psi'(A^\flat, l^\flat)$, for all A in $\lambda\mathcal{N}h\mathtt{x}$ and all l in $\lambda\mathcal{P}h\mathtt{x}$, by simultaneous induction on M and A, with induction hypotheses IH1 and IH2, respectively.

Case $M = app(A)$:

$$
\begin{array}{rcl}
(\Psi(M))^\flat & = & (\Psi(app(A)))^\flat \\
& = & (\Psi'(A, []))^\flat, \text{ by def. of } \Psi, \\
& = & \Psi'(A^\flat, []^\flat), \text{ by IH2}, \\
& = & \Psi'(A^\flat, []), \text{ by def. of } (\_)^\flat, \\
& = & \Psi(app(A^\flat)), \text{ by def. of } \Psi, \\
& = & \Psi((app(A))^\flat), \text{ by def. of } (\_)^\flat, \\
& = & \Psi(M^\flat) \ .
\end{array}
$$

Case $M = M_0\langle x := N_0\rangle$:

$$
\begin{array}{rcl}
(\Psi(M))^\flat & = & (\Psi(M_0\langle x := N_0\rangle))^\flat \\
& = & ((\Psi M_0)\{x := (\Psi N_0)\})^\flat, \text{ by def. of } \Psi, \\
& = & subst((\Psi N_0)^\flat, x, (\Psi M_0)^\flat), \text{ by def. of } (\_)^\flat, \\
& = & subst(\Psi(N_0^\flat), x, \Psi(M_0^\flat)), \text{ by IH1}, \\
& = & \Psi(M_0^\flat[N_0^\flat/x]), \text{ by Lemma 52}, \\
& = & \Psi((M_0\langle x := N_0\rangle)^\flat), \text{ by def. of } (\_)^\flat, \\
& = & \Psi(M^\flat) \ .
\end{array}
$$

The remaining cases are straightforward. ■

Corollary 32 *If $\lambda\mathcal{N}h\mathtt{x}$ derives $\Gamma \vdash M : A$, then $\lambda\mathcal{N}h$ derives $\Gamma \vdash M^\flat : A$.*

Proof: Obtained from correctness of $(\_)^\flat : \lambda\mathcal{P}h\mathtt{x} \to \lambda\mathcal{P}h$ (Proposition 9) by the method of Corollary 26. ■

Now we lift the isomorphism $\lambda\mathcal{N}h \cong \lambda\mathcal{P}h$.

Yet another isomorphism

The goal is now to prove $\lambda\mathcal{N}h\mathtt{x} \cong \lambda\mathcal{P}h\mathtt{x}$. We will try to follow closely the developments that led from Proposition 44 to Corollary 29, emphasizing the differences.

Proposition 48

1. $\Theta \circ \Psi = id$ *and* $\Theta \circ \Psi' = \Theta'$.
2. $\Psi \circ \Theta = id$ *and* $\Psi \circ \Theta' = \Psi'$.

Proof: The novelty relatively to Propositions 44 and 45 is the cases of explicit substitution and mid cut. The result follows because one constructor is mapped to the other and *vice versa*. ∎

Since there are no substitution operators in $\lambda\mathcal{N}h\mathtt{x}$ and $\lambda\mathcal{P}h\mathtt{x}$, there are no analogues of Lemma 52 and Corollary 27. Instead, we need the following two results.

Lemma 57 $\Psi'(A,l)\{x := \Psi N\} \rightarrow_{\mathtt{x4}} \Psi'(sub(N,x,A),x,sub(\Psi N,x,l))$*, for all* A,N *in* $\lambda\mathcal{N}h\mathtt{x}$*, all* l *in* $\lambda\mathcal{P}h\mathtt{x}$*.*

Proof: By induction on A.

Case $A = M'N'$.

$$
\begin{array}{ll}
 & \Psi'(A,l)\{x := \Psi N\} \\
= & \Psi'(M'N',l)\{x := \Psi N\} \\
= & (\Psi M'(\Psi N' \cdot l))\{x := \Psi N\}, \text{ by def. of } \Psi, \\
\rightarrow_{\mathtt{x4}} & ((\Psi M')\{x := \Psi N\})(((\Psi N')\{x := \Psi N\}) \cdot sub(\Psi N,x,l)) \\
= & (\Psi(M'\langle x := N\rangle))((\Psi(N'\langle x := N\rangle)) \cdot sub(\Psi N,x,l)), \text{ by def. of } \Psi, \\
= & \Psi'((M'\langle x := N\rangle)(N'\langle x := N\rangle), sub(\Psi N,x,l)), \text{ by def. of } \Psi, \\
= & \Psi'(sub(N,x,M'N'), sub(\Psi N,x,l)), \text{ by def. of } sub, \\
= & \Psi'(sub(N,x,A), sub(\Psi N,x,l))\ .
\end{array}
$$

Case $A = A'N'$:

$$
\begin{array}{rl}
 & \Psi'(A,l)\{x := \Psi N\} \\
= & \Psi'(A'N',l)\{x := \Psi N\} \\
= & \Psi'(A',\Psi N' :: l)\{x := \Psi N\}, \text{ by def. of } \Psi, \\
\to_{\mathtt{x4}} & \Psi'(sub(N,x,A'), sub(\Psi N, x, \Psi N' :: l)), \text{ by IH}, \\
= & \Psi'(sub(N,x,A'), ((\Psi N')\{x := \Psi N\}) :: sub(\Psi N,x,l)), \text{ by def. of } sub, \\
= & \Psi'(sub(N,x,A'), (\Psi(N'\langle x := N\rangle)) :: sub(\Psi N,x,l)), \text{ by def. of } \Psi, \\
= & \Psi'(sub(N,x,A')(N'\langle x := N\rangle), sub(\Psi N,x,l)), \text{ by def. of } \Psi, \\
= & \Psi'(sub(N,x,A'N'), sub(\Psi N,x,l)), \text{ by def. of } sub, \\
= & \Psi'(sub(N,x,A), sub(\Psi N,x,l)) \ .
\end{array}
$$

■

One way of understanding next lemma is by observing that it asserts a reduction between two terms that are the images under Θ of the two terms involved in the reduction asserted by the previous lemma.

Lemma 58 $\Theta'(A,l)\langle x := \Theta v\rangle \to_{\mathtt{x4}} \Theta'(sub(\Theta v,x,A), sub(v,x,l))$, *for all A in $\lambda\mathcal{N}h\mathtt{x}$, all v,l in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: By induction on l.

Case $l = []$.

$$
\begin{array}{rl}
 & \Theta'(A,l)\langle x := \Theta v\rangle \\
= & \Theta'(A,[])\langle x := \Theta v\rangle \\
= & app(A)\langle x := \Theta v\rangle, \text{ by def. of } \Theta, \\
\to_{\mathtt{x4}} & app(sub(\Theta v,x,A)) \\
= & \Theta'(sub(\Theta v,x,A),[]), \text{ by def. of } \Theta, \\
= & \Theta'(sub(\Theta v,x,A), sub(v,x,[])), \text{ by def. of } sub, \\
= & \Theta'(sub(\Theta v,x,A), sub(v,x,l)) \ .
\end{array}
$$

Case $l = u' :: l'$.

$$
\begin{array}{rl}
& \Theta'(A,l)\langle x := \Theta v\rangle \\
= & \Theta'(A,u' :: l')\langle x := \Theta v\rangle \\
= & \Theta'(A\Theta u',l')\langle x := \Theta v\rangle, \text{ by def. of } \Theta, \\
\to_{\mathtt{x4}} & \Theta'(sub(\Theta v,x,A\Theta u'),sub(v,x,l')), \text{ by IH}, \\
= & \Theta'(sub(\Theta v,x,A)(\Theta u'\langle x := \Theta v\rangle),sub(v,x,l')), \text{ by def. of } sub, \\
= & \Theta'(sub(\Theta v,x,A)\Theta(u'\{x := v\}),sub(v,x,l')), \text{ by def. of } \Theta, \\
= & \Theta'(sub(\Theta v,x,A),(u'\{x := v\}) :: sub(v,x,l')), \text{ by def. of } \Theta, \\
= & \Theta'(sub(\Theta v,x,A),sub(v,x,u' :: l')), \text{ by def. of } sub, \\
= & \Theta'(sub(\Theta v,x,A),sub(v,x,l)) \ .
\end{array}
$$

The following five results are an immediate lifting of, and have exactly the same proofs as the corresponding results we have seen before.

Lemma 59 *$\Psi'(A,l)(u' \cdot l') \to_h \Psi'(A, append(l,u' :: l')$, for all A in $\lambda\mathcal{N}h\mathtt{x}$, all u',l,l' in $\lambda\mathcal{P}h\mathtt{x}$.*

Corollary 33 *$\Psi(app(A))(u \cdot l) \to_h \Psi'(A, u :: l)$, for all A in $\lambda\mathcal{N}h\mathtt{x}$, all u, l in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: From Lemma 59. ■

Lemma 60 *In $\lambda\mathcal{P}h\mathtt{x}$, if $l \to_R l'$, then $\Psi'(A,l) \to_R \Psi'(A,l')$ (for all A in $\lambda\mathcal{N}h\mathtt{x}$, $R \in \{b1, b2, h, \mathtt{x1}, \mathtt{x2}, \mathtt{x3}, \mathtt{x4}\}$).*

Lemma 61 *In $\lambda\mathcal{N}h\mathtt{x}$, if $A \to_R A'$, then $\Theta'(A,l) \to_R \Theta'(A',l)$ (for all l in $\lambda\mathcal{P}h\mathtt{x}$, $R \in \{b1, b2, h, \mathtt{x1}, \mathtt{x2}, \mathtt{x3}, \mathtt{x4}\}$).*

Lemma 62 *$\Theta'(\Theta'(A,l)\Theta u',l') \to_h \Theta'(A, append(l,u' :: l'))$, for all A in $\lambda\mathcal{N}h\mathtt{x}$, all u',l,l' in $\lambda\mathcal{P}h\mathtt{x}$.*

Proof: Using Lemma 61. ■

Finally, we are ready for the isomorphism theorems.

Theorem 13 *Let* $R \in \{b1, b2, h, \mathtt{x1}, \mathtt{x2}, \mathtt{x3}, \mathtt{x4}\}$*. If* $M \to_R M'$ *in* $\lambda\mathcal{N}h\mathtt{x}$ *then* $\Psi M \to_R \Psi M'$ *in* $\lambda\mathcal{P}h\mathtt{x}$*.*

Proof: We prove the claim and also that

if $A \to_R A'$ in $\lambda\mathcal{N}h\mathtt{x}$, then $\Psi'(A, l) \to_R \Psi'(A', l)$ in $\lambda\mathcal{P}h\mathtt{x}$, for all l in $\lambda\mathcal{P}h\mathtt{x}$,

by simultaneous induction, similar to the proof of Theorem 11, on $M \to_R M'$ and $A \to_R A'$, with induction hypotheses IH1 and IH2, respectively. Cases correspond to closure rules, according to Definition 19.

Case $b1$:

$$
\begin{array}{rcl}
\Psi(app((\lambda x.M)N)) & = & \Psi'((\lambda x.M)N, []), \text{ by def. of } \Psi, \\
& = & (\lambda x.\Psi M)(\Psi N \cdot []), \text{ by def. of } \Psi, \\
& \to_{b1} & (\Psi M)\{x := \Psi N\} \\
& = & \Psi(M\langle x := N\rangle), \text{ by def. of } \Psi.
\end{array}
$$

Case $b2$:

$$
\begin{array}{rcl}
\Psi'(((\lambda x.M)N)N', l) & = & \Psi'(((\lambda x.M)N), \Psi N' :: l), \text{ by def. of } \Psi, \\
& = & (\lambda x.\Psi M)(\Psi N \cdot (\Psi N' :: l)), \text{ by def. of } \Psi, \\
& \to_{b2} & ((\Psi M)\{x := \Psi N\})(\Psi N' \cdot l) \\
& = & \Psi(M\langle x := N\rangle)(\Psi N' \cdot l), \text{ by def. of } \Psi, \\
& = & \Psi'(M\langle x := N\rangle N', l), \text{ by def. of } \Psi.
\end{array}
$$

Case h: Exactly as in Theorem 11, but using Corollary 33 instead of Corollary 28.

Cases $\mathtt{x1}$, $\mathtt{x2}$ and $\mathtt{x3}$: Straightforward.

Case $\mathtt{x4}$:

$$\begin{array}{rcl}\Psi(app(A)\langle x := N\rangle) & = & \Psi(app(A))\{x := \Psi N\}, \text{ by def. of } \Psi, \\ & = & \Psi'(A, [])\{x := \Psi N\}, \text{ by def. of } \Psi, \\ & \rightarrow_{\mathtt{x4}} & \Psi'(sub(N, x, A), sub(\Psi N, x, [])), \text{ by Lemma 57}, \\ & = & \Psi'(sub(N, x, A), []), \text{ by def. of } sub, \\ & = & \Psi(app(sub(N, x, A))), \text{ by def. of } \Psi.\end{array}$$

Case $Intro$: As in Theorem 11, by closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $Right$.

Case App: As in Theorem 11.

Case $HdElim1$: As in Theorem 11, by closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $HdCut1$.

Case $HdElim2$: As in Theorem 11, by closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $HdCut2$.

Case $TailElim1$: As in Theorem 11.

Case $TailElim2$: As in Theorem 11, by closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $Lft1$, but by Lemma 60 instead of Lemma 54.

Case $ExSubst1$: Suppose $\Psi M \rightarrow_R \Psi M'$ (IH1).

$$\begin{array}{rcl}\Psi(M\langle x := N\rangle) & = & (\Psi M)\{x := \Psi N\}, \text{ by def. of } \Psi, \\ & \rightarrow_R & (\Psi M')\{x := \Psi N\} \ (*) \\ & = & \Psi(M'\langle x := N\rangle), \text{ by def. of } \Psi,\end{array}$$

where the reduction step (*) is by IH1 and closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $MidCut1$.

Case $ExSubst2$: Similarly, by closure of $\rightarrow_R$ in $\lambda\mathcal{P}h\mathtt{x}$ under $MidCut2$. ■

Theorem 14 *Let $R \in \{b1, b2, h, \mathtt{x1}, \mathtt{x2}, \mathtt{x3}, \mathtt{x4}\}$. If $t \rightarrow_R t'$ in $\lambda\mathcal{P}h\mathtt{x}$ then $\Theta t \rightarrow_R \Theta t'$ in $\lambda\mathcal{N}h\mathtt{x}$.*

Proof: We prove the claim and also that

if $l \rightarrow_R l'$ in $\lambda\mathcal{P}h$, then $\Theta'(A, l) \rightarrow_R \Theta(A, l')$ in $\lambda\mathcal{N}h\mathtt{x}$, for all A in $\lambda\mathcal{N}h\mathtt{x}$, by simultaneous induction, similar to that of Theorem 12, on $t \to_R t'$ and $l \to_R l'$. Cases correspond to closure rules, according to Definition 7.

Case $b1$:

$$\begin{array}{rcl}\Theta((\lambda x.t)(u \cdot [])) & = & \Theta'((\lambda x.\Theta t)\Theta u, [])\text{by def. of }\Theta, \\ & = & app((\lambda x.\Theta t)\Theta u)\text{by def. of }\Theta, \\ & \to_{b1} & \Theta t\langle x := \Theta u\rangle \\ & = & \Theta(t\{x := u\})\text{by def. of }\Theta.\end{array}$$

Case $b2$:

$$\begin{array}{rcl}\Theta((\lambda x.t)(v \cdot (u :: l))) & = & \Theta'((\lambda x.\Theta t)\Theta v, u :: l),\text{ by def. of }\Theta, \\ & = & \Theta'(((\lambda x.\Theta t)\Theta v)\Theta u, l),\text{ by def. of }\Theta, \\ & \to_R & \Theta'((\Theta t\langle x := \Theta v\rangle)\Theta u, l),\text{ by Lemma 61}, \\ & = & \Theta'(\Theta(t\{x := v\})\Theta u, l),\text{ by def. of }\Theta, \\ & = & \Theta((t\{x := v\})(u :: l)),\text{ by def. of }\Theta.\end{array}$$

Case h: Exactly as in Theorem 12, but using Lemma 62 instead of Lemma 56.

Cases $\mathtt{x1}$, $\mathtt{x2}$ and $\mathtt{x3}$: Straightforward.

Case $\mathtt{x4}$:

$$\begin{array}{rl} & \Theta((t(u \cdot l)\{x := v\})) \\ = & \Theta(t(u \cdot l))\langle x := \Theta v\rangle,\text{ by def. of }\Theta, \\ = & \Theta'(\Theta t\Theta u, l)\langle x := \Theta v\rangle,\text{ by def. of }\Theta, \\ \to_{\mathtt{x4}} & \Theta'(sub(\Theta v, x, \Theta t\Theta u), sub(v, x, l)),\text{ by Lemma 58}, \\ = & \Theta'((\Theta t)\langle x := \Theta v\rangle(\Theta u)\langle x := \Theta v\rangle, sub(v, x, l)),\text{ by def. of }sub, \\ = & \Theta'((\Theta(t\{x := v\})\Theta(u\langle x := v\}), sub(v, x, l)),\text{ by def. of }\Theta, \\ = & \Theta((t\{x := v\})(u\{x := v\} \cdot sub(v, x, l))),\text{ by def. of }\Theta.\end{array}$$

Case *Right*: As in Theorem 12, by closure of $\to_R$ in $\lambda\mathcal{N}h\mathtt{x}$ under *Intro*.

Case *HdCut*1: As in Theorem 12, by closure of $\to_R$ in $\lambda\mathcal{N}h\mathtt{x}$ under *HdElim*1, but by Lemma 61 instead of Lemma 55.

Case *HdCut*2: As in Theorem 12, by closure of $\to_R$ in $\lambda\mathcal{N}h\mathtt{x}$ under *HdElim*2.

Case *HdCut*3: As in Theorem 12.

Case *MidCut*1: Suppose $\Theta t \to_R \Theta t'$ (IH1).

$$\begin{array}{rcl}\Theta(t\{x := u\}) & = & \Theta t\langle x := \Theta u\rangle, \text{ by def. of } \Theta, \\ & \to_R & \Theta t'\langle x := \Theta u\rangle \ (*) \\ & = & \Theta t'\{x := \Theta u\}, \text{ by def. of } \Theta,\end{array}$$

where the reduction step (*) is by IH1 and closure of $\to_R$ in $\lambda\mathcal{N}h\mathtt{x}$ under *ExSubst*1.

Case *MidCut*2: Similarly, but by closure of $\to_R$ in $\lambda\mathcal{N}h\mathtt{x}$ under *ExSubst*2. ■

Corollary 34 (Isomorphism) *Let* $R \in \{b1, b2, h, \mathtt{x}1, \mathtt{x}2, \mathtt{x}3, \mathtt{x}4\}$.

1. $M \to_R M'$ *in* $\lambda\mathcal{N}h\mathtt{x}$ *iff* $\Psi M \to_R \Psi M'$ *in* $\lambda\mathcal{P}h\mathtt{x}$.

2. $t \to_R t'$ *in* $\lambda\mathcal{P}h\mathtt{x}$ *iff* $\Theta t \to_R \Theta t'$ *in* $\lambda\mathcal{N}h\mathtt{x}$.

Corollary 35

1. $\lambda\mathcal{N}h\mathtt{x}$ *is confluent.*

2. *If* M *is typable in* $\lambda\mathcal{N}h\mathtt{x}$*, then* M *is strongly normalising.*

3. $\lambda\mathcal{N}h\mathtt{x}$ *satisfies subject reduction.*

Proof: Because these properties hold of $\lambda\mathcal{P}h\mathtt{x}$ and may be easily transferred from $\lambda\mathcal{P}h\mathtt{x}$ to $\lambda\mathcal{N}h\mathtt{x}$ with the help of Ψ and Θ. ■

The next two results are proved in view of obtaining a third, asserting internal conservativeness of $\lambda\mathcal{N}h\mathtt{x}$ over $\lambda\mathcal{N}h$. The idea of proofs is, as it were, to reuse known properties through isomorphisms Ψ and Θ.

Proposition 49 *$\to_{\mathtt{x}}$ in $\lambda\mathcal{N}h\mathtt{x}$ is confluent.*

Proof: Suppose $M_0 \to^*_{\mathtt{x}} M_1, M_2$ in $\lambda\mathcal{N}h\mathtt{x}$. By Theorem 13, $\Psi M_0 \to^*_{\mathtt{x}} \Psi M_1, \Psi M_2$ in $\lambda\mathcal{P}h\mathtt{x}$. By Corollary 7, there is t in $\lambda\mathcal{P}h\mathtt{x}$ such that $\Psi M_1, \Psi M_2 \to^*_{\mathtt{x}} t$. Then, by Theorem 14, and since $\Theta\Psi M = M$, we get $M_1, M_2 \to^*_{\mathtt{x}} \Theta t$. ■

Therefore, we may refer to the normal-form mapping $\downarrow_{\mathtt{x}}$.

Proposition 50 *$M^\flat = \downarrow_{\mathtt{x}} (M)$, for all M in $\lambda\mathcal{N}h\mathtt{x}$.*

Proof: Since $M^\flat$ is $\mathtt{x}$-normal, it suffices to prove $M \to^*_{\mathtt{x}} M^\flat$. Now, by Propositions 11 and 47, $\Psi M \to^*_{\mathtt{x}} (\Psi M)^\flat = \Psi(M^\flat)$. Therefore, by Theorem 14, we get $M = \Theta\Psi M \to^*_{\mathtt{x}} \Theta\Psi(M^\flat) = M^\flat$. ■

Corollary 36 *$\lambda\mathcal{N}h\mathtt{x}$ is internally conservative over $\lambda\mathcal{N}h$.*

Proof: Conservativeness is by

$$
\begin{array}{ll}
 & M \to^* M' \text{ in } \lambda\mathcal{N}h \\
\text{iff} & \Psi M \to^* \Psi M' \text{ in } \lambda\mathcal{P}h\text{, by Corollary 29,} \\
\text{iff} & i(\Psi M) \to^* i(\Psi M') \text{ in } \lambda\mathcal{P}h\mathtt{x}\text{, as } \lambda\mathcal{P}h\mathtt{x} \text{ is conservative over } \lambda\mathcal{P}h, \\
\text{iff} & \Psi(i(M)) \to^* \Psi(i(M')) \text{ in } \lambda\mathcal{P}h\mathtt{x}\text{, by Proposition 46,} \\
\text{iff} & \Theta\Psi(i(M)) \to^* \Theta\Psi(i(M')) \text{ in } \lambda\mathcal{N}h\mathtt{x}\text{, by Corollary 34,} \\
\text{iff} & i(M) \to^* i(M') \text{ in } \lambda\mathcal{N}h\text{, as } \Theta \circ \Psi = id.
\end{array}
$$

Internal conservativeness is by Proposition 50. ■

Completing the picture

Recall from section 5.1 that Gentzen's mapping $\mathcal{G}$ is an isomorphism between λ and $\lambda\mathcal{G}$, the latter being the ::-free subsystem of $\lambda\mathcal{P}h$, in which, therefore, only reduction rule $\beta 1$ makes sense. The isomorphism amounts to the rephrasing of application as the head-cut $t[t']$ (this is short for $t(t' \cdot []))$.

Now, the same fragment exists in $\lambda\mathcal{P}h\mathtt{x}$. Terms are of the form

$$t, u, v ::= x \,|\, \lambda x.t \,|\, t[u] \,|\, t\{x := u\} \ ,$$

reduction rules $b2$ and h are dropped (as they require ::) and all the remaining rules $b1, \mathtt{x}1, \mathtt{x}2, \mathtt{x}3, \mathtt{x}4$ are retained. The fragment is indeed closed for these rules. Only rule $\mathtt{x}4$ requires a verification:

$$\begin{array}{rcl} (t[u])\{x := v\} & = & (t(u \cdot []))\{x := v\} \\ & \rightarrow_{\mathtt{x}4} & (t\{x := v\})(u\{x := v\} \cdot sub(v, x, [])) \\ & = & (t\{x := v\})(u\{x := v\} \cdot []) \\ & = & (t\{x := v\})[u\{x := v\}] \ . \end{array}$$

This fragment, named $\lambda\mathcal{G}\mathtt{x}$, is nothing but a rephrasing of $\lambda\mathtt{x}$[2]. Application is rephrased as before and explicit substitution is rephrased as mid-cut. Typing rules are the same, except that, in $\lambda\mathcal{G}\mathtt{x}$, sequents have the form $\Gamma; - \vdash t : A$. The rephrasing mapping $\mathcal{G} : \lambda\mathtt{x} \rightarrow \lambda\mathcal{G}\mathtt{x}$ is an extension of $\mathcal{G} : \lambda \rightarrow \lambda\mathcal{G}$ defined by

$$\mathcal{G}(M\langle x := N\rangle) \ = \ \mathcal{G}M\{x := \mathcal{G}N\} \ .$$

By analogy with mapping $(\_)^\flat : \lambda\mathcal{P}h\mathtt{x} \rightarrow \lambda\mathcal{P}h$, let us define mappings $(\_)^\flat$: $\lambda\mathcal{G}\mathtt{x} \rightarrow \lambda\mathcal{G}$ and $(\_)^\flat : \lambda\mathtt{x} \rightarrow \lambda$ simply by translating, in the first case, mid-cut as substitution (or rather *subst*) in $\lambda\mathcal{G}$, and translating, in the second case, explicit substitution as substitution in λ. The situation is as follows:

[2]Therefore, this may be regarded as a logical reconstruction of $\lambda\mathtt{x}$.

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[d, "(\_)^\flat"'] \arrow[dr, phantom, "(1a)"] & \lambda\mathcal{G}\mathtt{x} \arrow[l, "i"'] \arrow[d, "(\_)^\flat"] \arrow[dr, phantom, "(1b)"] & \lambda\mathtt{x} \arrow[l, "\mathcal{G}"'] \arrow[d, "(\_)^\flat"] \\
\lambda\mathcal{P}h & \lambda\mathcal{G} \arrow[l, "i"] & \lambda \arrow[l, "\mathcal{G}"]
\end{tikzcd}
\tag{6.2}
$$

Square $(1b)$ is commutative because rephrasing $\mathcal{G}$ clearly commutes with $(\_)^\flat$. As to square $(1a)$, recall that $(\_)^\flat : \lambda\mathcal{P}h\mathtt{x} \to \lambda\mathcal{P}h$ is determined by the translation of mid-cut as *subst* in $\lambda\mathcal{P}h$. Consider the restriction of this mapping to $\lambda\mathcal{G}\mathtt{x}$. Its range is not the whole $\lambda\mathcal{P}h$ but only $\lambda\mathcal{G}$. This is so because $\lambda\mathcal{G}$ is closed for *subst* of $\lambda\mathcal{P}h$. Since *subst* in $\lambda\mathcal{G}$ (in terms of which we defined $(\_)^\flat : \lambda\mathcal{G}\mathtt{x} \to \lambda\mathcal{G}$) is the restriction to $\lambda\mathcal{G}$ of *subst* in $\lambda\mathcal{P}h$, the commutativity of $(1a)$ follows.

Let $\mathtt{x} = \mathtt{x1} \cup \mathtt{x2} \cup \mathtt{x3} \cup \mathtt{x4}$. From the construction of diagram (6.2) and Corollary 8, it is clear that $\to_{\mathtt{x}}$ is confluent in each $\lambda\mathcal{P}h\mathtt{x}$, $\lambda\mathcal{G}\mathtt{x}$ and $\lambda\mathtt{x}$, and that $(\_)^\flat = \downarrow_{\mathtt{x}}$, for each $(\_)^\flat$. We could also infer, from diagram (6.2) and conservativeness of $\lambda\mathcal{P}h\mathtt{x}$ over $\lambda\mathcal{P}h$, the conservativeness of $\lambda\mathcal{G}\mathtt{x}$ (resp. $\lambda\mathtt{x}$) over $\lambda\mathcal{G}$ (resp. λ). Anyway, conservativeness of $\lambda\mathtt{x}$ over λ is not new [Rose, 1996b].

Now we prove that square

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[d, "(\_)^\flat"'] & \lambda\mathcal{N}h\mathtt{x} \arrow[l, "\Psi"'] \arrow[d, "(\_)^\flat"] \\
\lambda\mathcal{P}h & \lambda\mathcal{N}h \arrow[l, "\Psi"']
\end{tikzcd}
\tag{6.3}
$$

(which is the back face of (6.5) below) generalises square $(1b)$ of diagram (6.2) (which is the front face of (6.5) below). This is interesting because it shows that the perfect match between mid-cuts in $\lambda\mathcal{G}$ and explicit substitutions of $\lambda\mathtt{x}$ extends from the ::-free and tail-application-free fragments to include, on the one hand, full head-cuts and left permutation, and, on the other hand, the distinction between head and tail eliminations and the associated new notion of reduction h. Accordingly, the perfect match will be established by the not so trivial Ψ, instead of mere rephrasing $\mathcal{G}$.

First, observe that, since both Ψ and $\mathcal{G}$ send explicit substitutions to mid-cuts, Ψ keeps being coherent with $\mathcal{G}$ (recall Proposition 43). The following is immediate.

Proposition 51 *$\Psi(\iota(M)) = \mathcal{G}(M)$, for all M in $\lambda\mathtt{x}$.*

Second, by analogy with square $(1a)$ of diagram (6.2), regarding the ::-free fragments of $\lambda\mathcal{P}h\mathtt{x}$ and $\lambda\mathcal{P}h$, there is a square

$$\begin{tikzcd} \lambda\mathcal{N}h\mathtt{x} \arrow[d, "(\_)^\flat"'] & \lambda\mathtt{x} \arrow[l, "\iota"'] \arrow[d, "(\_)^\flat"] \arrow[dl, phantom, "(2)"] \\ \lambda\mathcal{N}h & \lambda \arrow[l, "\iota"] \end{tikzcd} \tag{6.4}$$

regarding the tail-application-free fragments of $\lambda\mathcal{N}h\mathtt{x}$ and $\lambda\mathcal{N}h$. Mapping $\iota : \lambda \to \lambda\mathcal{N}h$ is lifted to a mapping $\iota : \lambda\mathtt{x} \to \lambda\mathcal{N}h\mathtt{x}$ by putting

$$\iota(M\langle x := N\rangle) \quad = \quad \iota M\langle x := \iota N\rangle \ .$$

What remains to be proved is that this last square commutes. By gluing it with diagrams (6.3) and (6.2), we obtain the cube

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[dd, "(\_)^\flat"'] & & \lambda\mathcal{N}h\mathtt{x} \arrow[ll, "\Psi"'] \arrow[dd, "(\_)^\flat"] & \\
& \lambda\mathcal{G}\mathtt{x} \arrow[ul, "i"'] \arrow[dd, "(\_)^\flat"'] & & \lambda\mathtt{x} \arrow[ll, "\mathcal{G}"'] \arrow[ul, "\iota"'] \arrow[dd, "(\_)^\flat"] \\
\lambda\mathcal{P}h & & \lambda\mathcal{N}h \arrow[ll, "\Psi"'] & \\
& \lambda\mathcal{G} \arrow[ul, "i"'] & & \lambda \arrow[ll, "\mathcal{G}"'] \arrow[ul, "\iota"']
\end{tikzcd}
\tag{6.5}
$$

We have seen that all faces of this cube, except one, commute. The bottom face (resp. the top face) commutes by Proposition 43 (resp. Proposition 51). The front face is square ($1b$) in diagram (6.2), whereas the back face is square (6.3), which commutes by Proposition 47. Two faces remain: one is the commutative square ($1a$) of diagram (6.2). The other is square (6.4), whose commutativity we seek. This commutativity follows by a diagram chase and the fact that $\Psi : \lambda\mathcal{N}h \to \lambda\mathcal{P}h$ is an isomorphism.

Logical content of explicit substitutions

The idea of making substitution explicit has a *logical* appeal that asks for the understanding of the proof-theoretical status of calculi of explicit substitutions. Yet, we could classify as *computational* the initial motivations and goals of explicit substitution calculi. This is so because initially [Abadi et al., 1991] they intended to serve as an intermediate formalism between usual λ-calculus and its actual implementations [3].

[3]One is interested in actual implementations of the λ-calculus because the problem of implementing real-world functional programming languages can usually be reduced to the problem

In an implementation, on the one hand, substitution (and renaming of bound variables) cannot be left in some informal limbo; on the other hand, if a calculus is to reflect the extant practice, substitution has to happen in a controlled way. In fact, in textbooks on (implementation of) functional languages [Henderson, 1980, Peyton Jones, 1987, Field and Harrison, 1988], one finds, among others[4], three kinds of implementations: evaluation by an interpreter (this goes back to the first paper on LISP [McCarthy, 1960]), compilation to an abstract machine like the SECD-machine [Landin, 1964], or graph reduction, invented in [Wadsworth, 1971]. The first two techniques are *environment-based*, in that arguments of a function are stored, together with the bound variable, in a separate list of *bindings*, called an *environment*, rather than immediately substituted in the body of the function. The second technique is based on the idea of substituting *pointers* (rather than the actual arguments) for the formal parameters of a function. Common to these techniques is the fact that substitution is delayed and copying of arguments is avoided. The motivations for doing so are clear: copying is typically a waste of space, and may be a waste of time if the copied arguments contain redexes whose different copies will have to be reduced separately later.

Other computational motivations are the following facts: (1) the number of β-steps is not a good measure of the cost of computation [Rose, 1996b]; (2) explicit substitution is a way of recovering confluence of weak reduction[5] in λ-calculus [Curien et al., 1996].

Because of these computational goals, the earlier developments in explicit substitution calculi had to introduce complications that, without destroying the logical content behind explicit substitutions, actually by adding extra ingredients to that content, made it less obvious and simple. The first complication is the use of de Bruijn indices. Computationally, this means that, not only substitution, but also renaming of bound variables is made explicit. This is what Rose calls "explicit naming" in [Rose, 1996b] and "explicit binding" in [Rose, 1996a]. Logically, this is related to an explicit management of weakening [Vestergaard and Wells, 1999].

---

of implementing the λ-calculus [Field and Harrison, 1988, Peyton Jones, 1987]

[4] *E.g.* compilation into combinators.

[5] In weak reduction, reduction under λ-abstractions is forbidden.

The second complication is the introduction of a separate syntactical class of substitutions, typical of the $\lambda\sigma$-calculus [Abadi et al., 1991] and its descendants [Lescanne, 1994]. A substitution s is typed with a list of types and a "closure" $t[s]$ is typed with a kind of simultaneous cut [Abadi et al., 1991, Pagano, 1998]

$$\frac{\Gamma \vdash s : A_1, ...A_n \quad A_1, ..., A_n \vdash t : B}{\Gamma \vdash t[s] : B}$$

The logical impact of introducing explicit substitutions is best understood in a simpler (perhaps the simplest) setting, namely the $\lambda\mathtt{x}$-calculus, where none of these complications is present. This calculus adds two ingredients to natural deduction.

The first ingredient is a new constructor, a form of cut,

$$\frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma \vdash B}\, x \notin \Gamma \ , \tag{6.6}$$

which is the logical content of the typing rule for explicit substitutions. Observe that, as stressed in [Negri and von Plato, 2001], this is a very particular kind of cut because formulas in the LHS of sequents are introduced only by axioms [6].

Nevertheless, the inclusion of such a constructor seems to have an advantage (besides the fact that the relationship with sequent calculus improves). Natural deduction is equipped with a way of reusing (or sharing) proofs. This is the old idea that the cut-formula is like a lemma in informal proofs. Gentzen observed in §2.2 of [Gentzen, 1935] that a sharing mechanism was missing in natural deduction, but this was the price to pay for keeping the tree format of natural deduction proofs. Indeed, if natural deduction is defined, not in "sequent style", but, instead, with the traditional trees of formulas, then a rule like (6.6) (or rather its formulation with trees of formulas) breaks the tree format because the conclusion of the proof of A has to be "linked" to several assumptions in the proof of B, as many as the elements in the assumption class A^x.

The second ingredient $\lambda\mathtt{x}$ adds to natural deduction is a new normalisation procedure, which is new, not only because the set of proofs it acts upon has

[6] The "strength" of the cut rule has nothing to do with the rule itself, but rather with the system to which the rule is adjoined.

changed, but also and mainly because it is a "small-step" procedure defined by means of local transformations of derivations.

Traditionally [Gentzen, 1935, Girard et al., 1989, Gallier, 1993], the content of cut-elimination proofs amounted to procedure consisting of local transformation of derivations[7], whereas normalisation [Prawitz, 1965] called external routines (like substitution) for performing global transformations of proofs. Lately, things have changed. In the sequent calculus side, starting with [Danos et al., 1997], cut-elimination procedures were proposed containing global operations like the complete, upward, right or left permutation of a cut, which were performed in "natural deduction style", that is, as if executed in one go by some external routine. See also [Urban and Bierman, 1999, Espírito Santo, 2000]. Explicit substitutions in $\lambda\mathtt{x}$ represent, in turn, an approximation of natural deduction to the spirit of original cut-elimination procedures, with normalisation completely internalised and broken down into local steps of reduction.

But there is more than an analogy between the spirit of normalisation in $\lambda\mathtt{x}$ and the spirit of small-step cut-elimination. The $\mathtt{x}$-rules do perform cut-elimination, where cut here is precisely the new constructor (6.6). This is true, not only up to $\lambda\mathcal{G}\mathtt{x} \cong \lambda\mathtt{x}$, but also, crucially, and before anything else, by an analysis in $\lambda\mathtt{x}$ of the effect of reduction rules in derivations. Therefore, let us emphasize that $\lambda\mathtt{x}$ really adds to natural deduction a cut rule and a procedure for its stepwise elimination, and this is *de facto* and not only up to interpretation.

All this holds of $\lambda\mathcal{N}h\mathtt{x}$. The difference is that this calculus represents a natural deduction system in which the two new ingredients introduced by $\lambda\mathtt{x}$ are combined with other new ingredients already present in $\lambda\mathcal{N}h$, namely the distinction between head and tail elimination, together with the related new notion of reduction.

[7] These procedures act on a system with a contraction rule and eliminate a generalisation of cut called multicut or mix. Recently [Dragalin, 1988, Dyckhoff, 1997, Troelstra and Schwitchtenberg, 2000], procedures were defined for contraction-free system which eliminate cut instead of mix. These procedures are defined in terms of local transformations of derivations together with uses of admissibility of contraction. The latter hides global transformations of derivations. Anyway, in both cases some atomic operations, which may not be regarded as "local", remain in the meta-language, like the duplication or the erasing of an entire derivation.

We are led to conclude the following: the inclusion of a cut rule, together with reduction rules for its stepwise elimination, is a feature of "calculi of sequents" like sequent calculus or natural deduction in "sequent style". It is not exclusive of sequent calculus. It does not tell sequent calculus from natural deduction. Usually natural deduction is not recognised as having a cut rule because, in the traditional presentation of natural deduction with trees of formulas, the cut rule would break the tree format, as explained above. Yet, $\lambda\mathtt{x}$ is a presentation of natural deduction (but as a "calculus of sequents") with a constructor - explicit substitution - which acts logically as a cut.

Similarly, the inclusion of a constructor for explicit substitution, together with rules for its stepwise propagation and performance, is a feature of term calculi, *e.g.* term calculi associated with sequent calculus or term calculi associated with natural deduction. Although the Curry-Howard correspondence traditionally associates a term calculus (possibly containing explicit substitutions) to a natural deduction system, explicit substitutions are not an exclusive of natural deduction. Usually sequent calculus is not recognised as having explicit substitutions because traditional presentations of sequent calculus do not emphasize the related term calculus, and, therefore, it becomes difficult to recognise that right permutation of cuts is related to a substitution operator. Yet, this is what may be observed in $\lambda\mathcal{P}h\mathtt{x}$, a presentation of a sequent calculus (together with a term calculus) in which the constructor for mid-cuts acts as an explicit *subst*, where *subst* is the substitution operator of $\lambda\mathcal{P}h$.

Therefore, explicit substitution is *not* an issue in the relationship between sequent calculus and natural deduction[8]. This is best seen by observing the cube (6.5). Mappings $\mathcal{G}$ and Ψ mediate between the left face and the right face of the cube, one corresponding to sequent calculus, the other to natural deduction. The issue here (see Chapter 7) is how to represent applicative terms. Mappings i and ι mediate between the front face and the back face. The former is a degenerate case of the latter, corresponding to the ::-free and tail-aplication-free fragments. Hence, the issue here is whether :: and tail-aplications occur or not. Finally,

[8]Although explicit substitution is an issue in the computational interpretation of sequent calculus (and natural deduction!).

mappings $(\_)^\flat$ mediate between the top face and the bottom face. The issue here is, logically, whether a cut rule is included or not; and, at the term calculus level, whether substitution is explicit or not. This applies equally to systems in the sequent calculus side and to systems in the natural deduction side. It is an issue which is, as it were, orthogonal to the sequent calculus versus natural deduction divide.

6.3 A new landscape

Summing up

We pause to observe the proof-theoretical landscape resulting from gluing diagrams (6.1) and (6.5), as depicted in Fig.6.1.

In this Figure, if $\mathtt{x}$ (resp. h) labels an arrow, then the arrow stands for $\downarrow_{\mathtt{x}}$, which is the same as $(\_)^\flat$ (resp. $\downarrow_h$, which is the same as $(\_)^-$). Vertical arrows represent projections, horizontal arrows represent isomorphisms. All squares and triangles commute.

Let us refer to $\lambda\mathcal{N}$, $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}h\mathtt{x}$ as $\mathcal{N}$*-systems*, and to $\lambda\mathcal{P}$, $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}h\mathtt{x}$ as $\mathcal{P}$*-systems*.

The resulting diagram may be seen as diagram (6.1) topped with a layer of explicit substitutions (the topmost face). Another point of view shows the diagram as consisting of a left half (corresponding to sequent calculus) and a right half (corresponding to natural deduction), with Ψ, Θ, $\mathcal{G}$ and $\mathcal{G}^{-1}$ mediating between these halves. Finally, the diagram may be considered as consisting of the back-most squares, together with a degenerate layer (the front-most square, where λ lives), corresponding to the ::-free and tail-application-free fragments.

Figure 6.1 is a visual summary of the main claims we are making in this chapter, namely that the natural deduction "space of calculi" may be expanded so as to provide perfect counterparts to calculi with different levels of explicitness, pertaining to the canonical fragment of sequent calculus.

Natural deduction could have been expanded a little bit further, so as to simulate auxiliary mid-cuts and their explicit elimination. That is, there is a

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[rr, leftrightarrow, "\Psi, \Theta"] \arrow[dd, "\mathtt{x}"'] & & \lambda\mathcal{N}h\mathtt{x} \arrow[dd, "\mathtt{x}"] & \\
& \lambda\mathcal{G}\mathtt{x} \arrow[ul, "i"'] \arrow[rr, leftrightarrow, "\mathcal{G}, \mathcal{G}^{-1}"] \arrow[dd, "\mathtt{x}"'] & & \lambda\mathtt{x} \arrow[ul, "\iota"'] \arrow[dd, "\mathtt{x}"] \\
\lambda\mathcal{P}h \arrow[rr, leftrightarrow, "\Psi, \Theta"] \arrow[dd, "h"'] & & \lambda\mathcal{N}h \arrow[dd, "h"'] & \\
& \lambda\mathcal{G} \arrow[ul, "i"'] \arrow[rr, leftrightarrow, "\mathcal{G}, \mathcal{G}^{-1}"] \arrow[dl, "(\_)^{-}"] & & \lambda \arrow[ul, "\iota"'] \arrow[dl, "\mathcal{N}"] \\
\lambda\mathcal{P} \arrow[rr, leftrightarrow, "\Psi, \Theta"] & & \lambda\mathcal{N} &
\end{tikzcd}
$$

Figure 6.1: Natural deduction counterparts

$\mathcal{N}$-system corresponding to the system obtained from $\lambda\mathcal{P}h\mathtt{x}$ by making explicit meta-operator *sub*. The idea is simply to make *sub* of $\lambda\mathcal{N}h\mathtt{x}$ explicit as well. However, there seems to be a point to which the explicitness of cut-elimination cannot be taken without breaking the perfect match between sequent calculus and natural deduction. This is when auxiliary head-cuts $l(u \cdot l')$ and their elimination are made explicit, as they are in $\overline{\lambda}\mathcal{P}h\mathtt{x}$. That is, we do not see what the natural deduction counterpart to the explicit and stepwise append of two list could be.

A distortion

Now we will argue that Fig. 6.1 contains a distortion relatively to the true proof-theoretical landscape defined by the relationship between $\mathcal{P}$-systems and $\mathcal{N}$-systems. Later we will make a proposal as to what the true landscape should be.

To see this, we have to start by treating equally the embeddings $\mathcal{G} : \lambda \to \lambda\mathcal{P}h$ and $\iota : \lambda \to \lambda\mathcal{N}h$. In Fig. 6.1, the isomorphic copy of λ in $\lambda\mathcal{P}h$ ($\lambda\mathcal{G}$) is visible, whereas the isomorphic copy of λ in $\lambda\mathcal{N}h$ ($\lambda\iota$) is not. The simpler solution is to hide $\lambda\mathcal{G}$ and $\lambda\mathcal{G}\mathtt{x}$ in Fig. 6.1. The result is Fig. 6.2.

Alternatively, we can make $\lambda\iota$ and the new $\lambda\iota\mathtt{x}$ visible in Fig.6.1 (as expected, $\lambda\iota\mathtt{x}$ is simply a rephrasing of $\lambda\mathtt{x}$, where application is written $app(MN)$). This causes a long, yet straightforward, chain of refinements.

First, we have the commutative triangles

$$\begin{tikzcd} \lambda\mathcal{N}h & \\ \lambda\iota \arrow[u, "i"] & \lambda \arrow[l, "\iota"] \arrow[ul, "\iota"'] \end{tikzcd} \qquad \begin{tikzcd} \lambda\mathcal{N}h\mathtt{x} & \\ \lambda\iota\mathtt{x} \arrow[u, "i"] & \lambda\mathtt{x} \arrow[l, "\iota"] \arrow[ul, "\iota"'] \end{tikzcd} \tag{6.7}$$

which are self-explanatory. Second, the commutative triangle asserted by Proposition 38 has to be decomposed as

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[r, leftrightarrow, "\Psi, \Theta"] \arrow[dd, "\mathtt{x}"'] & \lambda\mathcal{N}h\mathtt{x} \arrow[dd, "\mathtt{x}"] & \\
 & & \lambda\mathtt{x} \arrow[ul, "\iota"'] \arrow[ull, "\mathcal{G}"'] \arrow[dd, "\mathtt{x}"] \\
\lambda\mathcal{P}h \arrow[r, leftrightarrow, "\Psi, \Theta"] \arrow[dd, "h"'] & \lambda\mathcal{N}h \arrow[dd, "h"] & \\
 & & \lambda \arrow[ul, "\iota"'] \arrow[ull, "\mathcal{G}"'] \arrow[dl, "\mathcal{N}"] \\
\lambda\mathcal{P} \arrow[r, leftrightarrow, "\Psi, \Theta"] & \lambda\mathcal{N} &
\end{tikzcd}
$$

Figure 6.2: The old landscape (simple version)

$$\begin{tikzcd}
\lambda\mathcal{N}h \arrow[dd, "(_)^{-}"'] & & \\
& \lambda\iota \arrow[ul, "i"'] \arrow[dl, "(_)^{-}"] & \\
\lambda\mathcal{N} & & \lambda \arrow[ll, "\mathcal{N}"] \arrow[ul, "\iota"']
\end{tikzcd} \tag{6.8}$$

Mapping $(\_)^{-} : \lambda\iota \to \lambda\mathcal{N}$ is simply the restriction to $\lambda\iota$ of mapping $(\_)^{-} : \lambda\mathcal{N}h \to \lambda\mathcal{N}$. Hence, the upper triangle in diagram (6.8) commutes. Since the larger triangle of (6.8) commutes (by Proposition 38 and commutativity of the left triangle in (6.7)), the lower triangle in (6.8) commutes as well.

Third, square (6.4) as to be decomposed, very much like diagram (6.2), as follows:

$$\begin{tikzcd}
\lambda\mathcal{N}h\mathtt{x} \arrow[d, "(_)^{\flat}"'] \arrow[dr, phantom, "(2a)"] & \lambda\iota\mathtt{x} \arrow[l, "i"'] \arrow[d, "(_)^{\flat}"] \arrow[dr, phantom, "(2b)"] & \lambda\mathtt{x} \arrow[l, "\iota"'] \arrow[d, "(_)^{\flat}"] \\
\lambda\mathcal{N}h & \lambda\iota \arrow[l, "i"] & \lambda \arrow[l, "\iota"]
\end{tikzcd}$$

The proof that these two squares commute is exactly as the proof that the two squares in (6.2) commute. Alternatively, use the commutativity of (6.4), the trivial commutativity of square $(2b)$ and the fact that ι is an isomorphism to get commutativity of square $(2a)$.

Fourth, define both $\tilde{\mathcal{G}} : \lambda\mathcal{G} \to \lambda\iota$ and $\tilde{\mathcal{G}} : \lambda\mathcal{G}\mathtt{x} \to \lambda\iota\mathtt{x}$ by

$$\tilde{\mathcal{G}} \quad = \quad \mathcal{G} \circ \iota^{-1} \quad ,$$

and both $\tilde{\iota} : \lambda\iota \to \lambda\mathcal{G}$ and $\tilde{\iota} : \lambda\iota\mathtt{x} \to \lambda\mathcal{G}\mathtt{x}$ by

$$\tilde{\iota} \;=\; \iota \circ \mathcal{G}^{-1} \;.$$

Hence, $\tilde{\mathcal{G}}$ and $\tilde{\iota}$ are mutually inverse. This gives the commutative triangles

$$\begin{tikzcd}
\lambda\mathcal{G} \arrow[rr, leftrightarrow, "{\tilde{\mathcal{G}}, \tilde{\iota}}"] & & \lambda\iota \\
& & & \lambda \arrow[ulll, "\mathcal{G}"] \arrow[ul, "\iota"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\lambda\mathcal{G}\mathtt{x} \arrow[rr, leftrightarrow, "{\tilde{\mathcal{G}}, \tilde{\iota}}"] & & \lambda\iota\mathtt{x} \\
& & & \lambda\mathtt{x} \arrow[ulll, "\mathcal{G}"] \arrow[ul, "\iota"']
\end{tikzcd}$$

Let us glue

$$\begin{tikzcd}
\lambda\mathcal{G}\mathtt{x} \arrow[d, "\mathtt{x}"'] \arrow[dr, phantom, "(1b)"] & \lambda\mathtt{x} \arrow[l, "\mathcal{G}"'] \arrow[d, "\mathtt{x}"] \arrow[dr, phantom, "(2b)'"] & \lambda\iota\mathtt{x} \arrow[l, "\iota^{-1}"'] \arrow[d, "\mathtt{x}"] \\
\lambda\mathcal{G} & \lambda \arrow[l, "\mathcal{G}"] & \lambda\iota \arrow[l, "\iota^{-1}"]
\end{tikzcd}$$

where $(2b)'$ is like $(2b)$, but with ι^{-1} instead of ι. Since $(2b)'$ stays commutative, and since $\tilde{\mathcal{G}}$ and $\tilde{\iota}$ are mutually inverse, we get the two commutative squares

$$\begin{tikzcd}
\lambda\mathcal{G}\mathtt{x} \arrow[r, leftrightarrow, "{\tilde{\mathcal{G}}, \tilde{\iota}}"] \arrow[d, "\mathtt{x}"'] & \lambda\iota\mathtt{x} \arrow[d, "\mathtt{x}"] \\
\lambda\mathcal{G} \arrow[r, leftrightarrow, "{\tilde{\mathcal{G}}, \tilde{\iota}}"] & \lambda\iota
\end{tikzcd}$$

Fifth, observe that the two squares in

$$
\begin{tikzcd}
\lambda\mathcal{G} \arrow[d, "(\_)^-"'] & \lambda \arrow[l, "\mathcal{G}"'] \arrow[d, "\mathcal{N}"] & \lambda\iota \arrow[l, "\iota^{-1}"'] \arrow[d, "(\_)^-"] \\
\lambda\mathcal{P} & \lambda\mathcal{N} \arrow[l, "\Psi"'] \arrow[r, equal] & \lambda\mathcal{N}
\end{tikzcd}
$$

commute. The left one is by Theorem 7 (it is the "λ-square"), the right one is by the commutativity of the lower triangle in diagram (6.8). Since $\tilde{\mathcal{G}}$ and $\tilde{\iota}$, on the one hand, and Ψ and Θ, on the other hand, are mutually inverse, the following two squares commute

$$
\begin{tikzcd}
\lambda\mathcal{G} \arrow[r, leftrightarrow, "\tilde{\mathcal{G}}, \tilde{\iota}"] \arrow[d, "(\_)^-"'] & \lambda\iota \arrow[d, "(\_)^-"] \\
\lambda\mathcal{P} \arrow[r, leftrightarrow, "\Psi, \Theta"] & \lambda\mathcal{N}
\end{tikzcd}
$$

Finally, the two squares in

$$
\begin{tikzcd}
\lambda\mathcal{P}h & \lambda\mathcal{N}h \arrow[l, "\Psi"'] \arrow[r, equal] & \lambda\mathcal{N}h \\
\lambda\mathcal{G} \arrow[u, "i"] & \lambda \arrow[l, "\mathcal{G}"'] \arrow[u, "\iota"] & \lambda\iota \arrow[l, "\iota^{-1}"'] \arrow[u, "i"']
\end{tikzcd}
$$

commute. The left one is by Proposition 43, the right one by the commutativity of the left triangle in (6.7). Since $\tilde{\mathcal{G}}$ and $\tilde{\iota}$, on the one hand, and Ψ and Θ, on the other hand, are mutually inverse, the following two squares commute

$$
\begin{tikzcd}
\lambda\mathcal{P}h \arrow[r, leftrightarrow, "\Psi, \Theta"] & \lambda\mathcal{N}h \\
\lambda\mathcal{G} \arrow[u, "i"] \arrow[r, leftrightarrow, "\tilde{\mathcal{G}}, \tilde{\iota}"] & \lambda\iota \arrow[u, "i"']
\end{tikzcd}
$$

Similarly, using Proposition 51 and the commutativity of the right triangle in (6.7), one proves the commutativity of

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[r, leftrightarrow, "\Psi, \Theta"] & \lambda\mathcal{N}h\mathtt{x} \\
\lambda\mathcal{G}\mathtt{x} \arrow[u, "i"] \arrow[r, leftrightarrow, "\tilde{\mathcal{G}}, \tilde{\iota}"] & \lambda\iota\mathtt{x} \arrow[u, "i"']
\end{tikzcd}
$$

We sum up these facts in Fig. 6.3, a detailed version of Fig. 6.1 in which all squares and triangles remain commutative.

A proposal

We are now ready to propose an alternative architecture for the "space of calculi". Figures 6.2 and 6.3 do not treat the sequent calculus and the natural deduction sides symmetrically. They are biased towards the natural deduction side. This asymmetry is the graphical manifestation of a preconception, namely that the calculi $\lambda\mathcal{N}$, $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}h\mathtt{x}$ are "natural deduction systems" in the same sense that λ and $\lambda\mathtt{x}$ are natural deduction systems, and, therefore, that the latter should be "close" to the former.

However, we propose that λ (and $\lambda\mathtt{x}$, if substitution is to be explicit) is "equidistant" to $\lambda\mathcal{P}$, $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}h\mathtt{x}$, on the one hand, and to $\lambda\mathcal{N}$, $\lambda\mathcal{N}h$ and

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[rr, leftrightarrow, "\Psi,\Theta"] \arrow[dd, "\mathtt{x}"'] & & \lambda\mathcal{N}h\mathtt{x} \arrow[dd, "\mathtt{x}"] & & \\
& \lambda\mathcal{G}\mathtt{x} \arrow[ul, "i"'] \arrow[rr, leftrightarrow, "\tilde{\mathcal{G}},\tilde{\iota}"] \arrow[dd, "\mathtt{x}"] & & \lambda\iota\mathtt{x} \arrow[ul, "i"'] \arrow[dd, "\mathtt{x}"] & \\
\lambda\mathcal{P}h \arrow[rr, leftrightarrow, "\Psi,\Theta"] \arrow[dd, "h"'] & & \lambda\mathcal{N}h \arrow[dd, "h"] & & \lambda\mathtt{x} \arrow[ul, "\iota"'] \arrow[ulll, "\mathcal{G}"] \arrow[dd, "\mathtt{x}"] \\
& \lambda\mathcal{G} \arrow[ul, "i"'] \arrow[rr, leftrightarrow, "\tilde{\mathcal{G}},\tilde{\iota}"] \arrow[dl, "(\_)^-"] & & \lambda\iota \arrow[ul, "i"'] \arrow[dl, "(\_)^-"] & \\
\lambda\mathcal{P} \arrow[rr, leftrightarrow, "\Psi,\Theta"] & & \lambda\mathcal{N} & & \lambda \arrow[ul, "\iota"'] \arrow[ulll, "\mathcal{G}"]
\end{tikzcd}
$$

Figure 6.3: The old landscape (detailed version)

$$
\begin{tikzcd}
\lambda\mathcal{P}h\mathtt{x} \arrow[rr, leftrightarrow, "\Psi,\Theta"] \arrow[dd, "\mathtt{x}"'] & & \lambda\mathcal{N}h\mathtt{x} \arrow[dd, "\mathtt{x}"] \\
& \lambda\mathtt{x} \arrow[ul, "\mathcal{G}"'] \arrow[ur, "\iota"] \arrow[dd, "\mathtt{x}"] & \\
\lambda\mathcal{P}h \arrow[dd, "h"'] & {} \arrow[l, "\Psi"'] \arrow[r, "\Theta"] & \lambda\mathcal{N}h \arrow[dd, "h"] \\
& \lambda \arrow[ul, "\mathcal{G}"'] \arrow[ur, "\iota"] \arrow[dl, "\mathcal{P}"] \arrow[dr, "\mathcal{N}"'] & \\
\lambda\mathcal{P} \arrow[rr, leftrightarrow, "\Psi,\Theta"'] & & \lambda\mathcal{N}
\end{tikzcd}
$$

Figure 6.4: The new landscape

$\lambda\mathcal{N}h\mathtt{x}$, on the other hand. Indeed, if λ, $\lambda\iota$ and $\lambda\mathcal{G}$ are mere rephrasings of each other, if they are the same object, why should one consider λ and $\lambda\iota$ to be "closer" to $\mathcal{N}$-systems than $\lambda\mathcal{G}$? Therefore, we think that the true proof-theoretical landscape (in its simple version) is as shown in Fig. 6.4.

If $\mathcal{N}$-systems are regarded as "natural deduction" systems (as we do in this thesis, because they prefer elimination rules to left rules), then λ and $\lambda\mathtt{x}$ should be regarded as systems of a neutral kind, belonging to the intersection of sequent calculus and natural deduction. Notice that, with this taxonomy, λ belongs to the natural deduction side as much as it belongs to the sequent calculus side - something that sounds like a heresy [9].

[9] An alternative taxonomy is to consider as being "natural deduction" systems those calculi whose purpose is to model informal reasoning (this is the sense of the word "natural" in

Our proposal implies that there are pairs of systems and pairs of morphisms with homologous (let us say "dual") roles. For instance, the dual of Gentzen's embedding $\mathcal{G}$ is ι. The most interesting example concerns $\mathcal{N}$ and Prawitz's mapping $\mathcal{P}$. Prawitz's mapping is the "diagonal" of the λ-square

$$\begin{tikzcd}
\lambda\mathcal{G} \arrow[d, "(_)^-"'] & \lambda \arrow[l, "\mathcal{G}"'] \arrow[d, "\mathcal{N}"] \arrow[dl, "\mathcal{P}"'] \\
\lambda\mathcal{P} & \lambda\mathcal{N} \arrow[l, "\Psi"]
\end{tikzcd}$$

Similarly, there is a "dual" square

$$\begin{tikzcd}
\lambda \arrow[r, "\iota"] \arrow[d, "\mathcal{P}"'] \arrow[dr, "\mathcal{N}"] & \lambda\iota \arrow[d, "(_)^-"] \\
\lambda\mathcal{P} \arrow[r, "\Theta"'] & \lambda\mathcal{N}
\end{tikzcd}$$

whose "diagonal" is $\mathcal{N}$. Indeed, the lower triangle commutes because $\mathcal{P} = \Psi \circ \mathcal{N}$ and $\Theta \circ \Psi = id$, whereas the upper triangle is the lower triangle of (6.8).

[Gentzen, 1935, Prawitz, 1965]). In this case, λ and $\lambda\mathtt{x}$ keep being natural deduction systems (perhaps $\lambda\mathtt{x}$ even more than λ, because of the presence of a form of sharing). However, similarly to sequent calculus, $\mathcal{N}$-systems become "unnatural" because the distinction between two kinds of elimination rules seems artificial from the strict point of view of modelling informal reasoning. With this taxonomy, natural deduction is simultaneously a fragment of sequent calculus and a fragment of $\mathcal{N}$-systems.

Chapter 7

Two applications

In this chapter we give computational applications for two of the contributions of this thesis. First, we show that the new assignment Θ, particularly the extended λ-calculi that constitute its range, provide a language with which one can refine Curien and Herbelin's interpretation of sequent calculi in the canonical fragment. Second, we will show that the λ-calculi we defined for the canonical fragment have a remarkable relation with call-by-name abstract machines.

7.1 Refinement of computational interpretation

In this section we consider the assignment $\mathcal{Q}$ (which amounts to the traditional assignment φ - see Proposition 36) and the new assignment Θ, explaining what new insights the latter brings to the computational interpretation of sequent calculus.

We will focus on the relation between $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$, on the one hand, and $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$, on the other hand. This is so because it is in the differences between these systems that the distinction between sequent calculus and natural deduction is expressed, particularly in the opposition between two ways of representing applicative terms. As became clear in the preceding chapters, λ-abstraction and mid-cuts (or explicit substitution) are constructors that exist in both kinds of systems.

It is useful to situate ourselves with the following diagram

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[rr, "\Theta_2"] \arrow[dr, "\mathcal{Q}_2"] & & \lambda\mathcal{N}h \\
& \lambda & \\
\lambda\mathcal{P} \arrow[ur, "\mathcal{Q}_1"] \arrow[rr, "\Theta_1"] & & \lambda\mathcal{N}
\end{tikzcd}$$

where we distinguished mappings with the same name by means of indexes. Recall that $\mathcal{Q}_1$ and both Θ_1 and Θ_2 are isomorphisms, but $\mathcal{Q}_2$ is not. Typically, $\mathcal{Q}_2$ collapses $\rightarrow_h$-steps.

We will adapt to $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$ the computational interpretation contained in some remarks and insights due to Herbelin and Curien. Since these are in terms of evaluation contexts, we explain the latter first.

An evaluation, or applicative, context is an expression generated by the grammar

$$E ::= [-] \mid EN \tag{7.1}$$

Informally, it is an applicative λ-term with a "hole" $[-]$ in the head position. These evaluation context are call-by-name, as opposed to call-by-value ones in *e.g.* [Felleisen et al., 1986]. Filling the hole of $E = [-]N_1...N_k$ with M results in the applicative term $MN_1...N_k$ of the λ-calculus, denoted $E[M]$. A hole $[-]$ is itself a context. Given a context E and a term N , we can form another context $E[[-]N]$ such that $E[[-]N][M] = E[MN]$. Given contexts E, E', there is a context $E \circ E'$ satisfying $(E \circ E')[M] = E'[E[M]]$.

Herbelin and Curien's insights [Herbelin, 1995, Curien and Herbelin, 2000] are as follows: (1) the interpretation of a list is an evaluation context. (2) $[]$ is $[-]$. (3) $u :: l$ is $E[[-]N]$, where E, N are the interpretations of l, u. (4) a head-cut tl is interpreted as the result of filling the hole of E with M, if the interpretation of l, t is E, M.

In our setting, head-cuts are always of the form $t(u \cdot l)$, hence we refine (4) as: a head-cut $t(u \cdot l)$ is interpreted as the result of filling the hole of E with the *application* MN, if the interpretation of l, t, u is E, M, N. Therefore, we will only fill holes with applications. Moreover, it is clear that $append(l, l')$ corresponds to $E \circ E'$.

Now, interpretation (4) of a head-cut $t = t_0(u_1 \cdot [u_2, ..., u_k])$ is nothing but $\mathcal{Q}(t)$. Indeed, if $\mathcal{Q}t_0 = M$ and $\mathcal{Q}u_i = N_i$ then

$$\begin{array}{rcl} \mathcal{Q}(t_0(u_1 \cdot [u_2, ..., u_k])) & = & \mathcal{Q}'(M, N_1, [u_2, ..., u_k]) \\ & = & MN_1N_2...N_k \\ & = & ([-]N_2...N_k)[MN_1] \ . \end{array} \tag{7.2}$$

But interpreting $t_0(u_1 \cdot [u_2, ..., u_k])$ as $MN_1N_2...N_k$ seems more like saying what the head-cut is *not*. Indeed, if we stare enough at (7.2) we conclude that head-cut $t_0(u_1 \cdot [u_2, ..., u_k])$ is as if $MN_1N_2...N_k$ was decomposed into the application MN_1 and the evaluation context $[-]N_2...N_k$. Thus $\lambda\mathcal{P}h$ may be seen as a version of λ which, instead of application, includes a construction

$$\underline{\mathcal{Q}'}(MN, E) \ , \tag{7.3}$$

representing an applicative term. MN is its head application. E ranges over expressions

$$E ::= [-] \,|\, N :: E$$

representing evaluation contexts (the meaning of $N :: E$ is $E[[-]N]$). The head application MN and E contain the information needed for reconstructing the applicative term. The point of (7.3) is that the head application, deeply buried in traditional syntax, is brought to the surface. This is a known theme [Dyckhoff and Pinto, 1998]. We will also refer to MN in (7.3) as the *focus*.

We call (7.3) a $\mathcal{Q}$-*expression*, and it would be an interesting exercise to rewrite the definition of $\lambda\mathcal{P}h$ with this new syntax for head-cuts.

When the focused application is not a value application, we say that the focus is *imperfect*, and the $\mathcal{Q}$-expression is of the form

$$\underline{\mathcal{Q}'}(\underline{\mathcal{Q}'}(MN, E)N', E') \ .$$

This is precisely how a h-redex is written in this syntax. Then, a $\rightarrow_h$-step looks like

$$\underline{\mathcal{Q}'}(\underline{\mathcal{Q}'}(MN, E)N', E') \rightarrow_h \underline{\mathcal{Q}'}(MN, E \circ (N' :: E')) \ , \tag{7.4}$$

which may be interpreted as *improvement of focus*, a necessary feature when imperfect focus is allowed. However, by Lemma 50,

$$\mathcal{Q}'(\mathcal{Q}'(M, N, l), \mathcal{Q}u', l') = \mathcal{Q}'(M, N, \mathit{append}(l, u' :: l')) \ .$$

Actually, both members of this equation are one and the same λ-term

$$(MNN_2...N_k)N'N_2'...N_m' = MNN_2...N_kN'N_2'...N_m' \ .$$

In $\lambda\mathcal{P}$, $\mathcal{Q}$-expressions are restricted to the case when the focused application is a value application. The isomorphism $\mathcal{Q}_1$ means that these are enough and that, conversely, there is a canonical way of writing an application as a $\mathcal{Q}$-expression. Actually, $\lambda\mathcal{P}$ may be seen as a formalisation of the vector syntax of [Joachimski and Matthes].

With the interpretation of head-cuts as $\mathcal{Q}$-expressions, we sum up four formulations of the λ-calculus with applicative terms

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[dr, "\mathcal{Q}_2"] & & \lambda\mathcal{N}h \arrow[dl, "|_|_2"'] \\
& \lambda \arrow[dl, dotted, shift left, "\mathcal{P}"] \arrow[dr, dotted, shift right, "\mathcal{N}"'] & \\
\lambda\mathcal{P} \arrow[ur, shift left, "\mathcal{Q}_1"] & & \lambda\mathcal{N} \arrow[ul, shift right, "|_|_1"']
\end{tikzcd}$$

In $\mathcal{N}$-systems, applicative terms are of the form $app(A)$. In $\mathcal{P}$-systems, they are $\mathcal{Q}$-expressions. If an applicative term is mapped to an application MN in λ, we say that the former is an *unfolding* of the latter. Unfoldings of the same application may be linearly ordered: the smaller the head application, the bigger the unfolding. Mappings $\mathcal{P}$ and $\mathcal{N}$ send each application to its maximal unfolding.

Commutative triangles

$$\begin{tikzcd}
 & \lambda & \\
\lambda\mathcal{P} \arrow[ur, "\mathcal{Q}_1"] \arrow[rr, leftrightarrow, "{\Psi_1, \Theta_1}"'] & & \lambda\mathcal{N} \arrow[ul, "|_|_1"']
\end{tikzcd}$$

say that $\lambda\mathcal{P}$ and $\lambda\mathcal{N}$ are not only isomorphic, but also coherent as formulations of λ with applicative terms. Indeed, if two (representations of) applicative terms are related by Ψ, Θ, they are unfoldings of the same application. If we define $|\_|_2$ as the composition of $(\_)^- : \lambda\mathcal{N}h \to \lambda\mathcal{N}$ with $|\_|_1 : \lambda\mathcal{N} \to \lambda$, and if we recall the commutativity of

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[r, leftrightarrow, "{\Psi_2, \Theta_2}"] \arrow[d, "(_)^-"'] & \lambda\mathcal{N}h \arrow[d, "(_)^-"] \\
\lambda\mathcal{P} \arrow[r, leftrightarrow, "{\Psi_1, \Theta_1}"'] & \lambda\mathcal{N}
\end{tikzcd}$$

we get the commutativity of

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[rr, leftrightarrow, "{\Psi_2, \Theta_2}"] \arrow[dr, "\mathcal{Q}_2"'] & & \lambda\mathcal{N}h \arrow[dl, "|_|_2"] \\
 & \lambda &
\end{tikzcd}$$

which says that $\lambda\mathcal{P}h$ and $\lambda\mathcal{N}h$ are not only isomorphic, but also coherent as extensions of λ with applicative terms.

Summarising our quest for the computational ingredient that tells the canonical fragment of sequent calculus from natural deduction: if the λ-calculus is the only representative of the natural deduction world, then the formalisation of applicative terms that may be found in $\mathcal{P}$-systems is entirely due to the change to a sequent calculus format. If $\mathcal{N}$-systems are allowed in the natural deduction world, then the difference between $\mathcal{P}$-systems and natural deduction is merely in the *representation* of applicative terms, and what is typical of $\mathcal{P}$-systems is the focus on the head application.

Let us see yet another interpretation of $\mathcal{P}$-systems. This time, head-cuts in $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$ are interpreted as evaluation contexts, not for λ, but for $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$, respectively.

We define evaluation contexts for $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$ exactly as in (7.1), with the proviso that EN is to be understood as *tail* application. We are supposed to fill the hole of these contexts with *head* applications, and the result of filling MN_1 in the hole of $E = []N_2...N_k$ is the applicative term $app(MN_1N_2...N_k)$.

Then, if $\Theta t_0 = M$ and $\Theta u_i = N_i$,

$$\begin{array}{rcl} \Theta(t_0(u_1 \cdot [u_2, ..., u_k])) & = & \Theta'(MN_1, [u_2, ..., u_k]) \\ & = & app(MN_1N_2...N_k) \\ & = & ([-]N_2...N_k)[MN_1] \ . \end{array}$$

This suggests considering $\lambda\mathcal{P}h$ and $\lambda\mathcal{P}$ as versions of $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$, respectively, in which an applicative term (now in the formal sense of $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$) is decomposed into its *head* application (again in the formal sense of $\lambda\mathcal{N}h$ and $\lambda\mathcal{N}$) and an evaluation context (for $\lambda\mathcal{N}h$ or $\lambda\mathcal{N}$). Head-cuts become expressions displaying this information, which we name Θ-*expressions* and look like

$$\underline{\Theta'}(MN, E) \ . \tag{7.5}$$

Both in the case of $\lambda\mathcal{N}h$ and of $\lambda\mathcal{N}$ there is a mapping, actually an isomorphism, that decomposes each applicative term in a canonical way, by putting in the focus the head application. This mapping is Ψ. The picture now looks like

$$\begin{tikzcd}
\lambda\mathcal{P}h \arrow[rr, shift left, "\Theta_2"] & & \lambda\mathcal{N}h \arrow[ll, dotted, shift left, "\Psi_2"] \arrow[dl, "|_|_2"] \\
& \lambda & \\
\lambda\mathcal{P} \arrow[rr, shift left, "\Theta_1"] & & \lambda\mathcal{N} \arrow[ll, dotted, shift left, "\Psi_1"] \arrow[ul, "|_|_1"']
\end{tikzcd}$$

$\mathcal{N}$-systems are now responsible for the formalisation of applicative terms. $\mathcal{P}$-systems are isomorphic versions of $\mathcal{N}$-systems in which applicative are represented with focus on the head application, with the meaning of "applicative term" and "head application" given in each $\mathcal{N}$-system.

A $\rightarrow_h$-step now reads

$$\underline{\Theta'}(\underline{\Theta'}(MN, E)N', E') \rightarrow_h \underline{\Theta'}(MN, E \circ (N' :: E')) \ , \tag{7.6}$$

but this is simply a $\rightarrow_h$-step in $\lambda\mathcal{N}h$ written with Θ-expressions. Indeed, it follows from Lemma 56 that

$$\Theta'(\Theta'(MN, l)\Theta u', l') \rightarrow_h \Theta'(MN, \mathit{append}(l, u' :: l')) \ ,$$

which is, perhaps, more easily understood as the reduction step

$$\mathit{app}(\mathit{app}(MNN_2...N_k)N'N'_2...N'_m) \rightarrow_h \mathit{app}(MNN_2...N_kN'N'_2...N'_m) \ . \tag{7.7}$$

The interpretation of (7.6) as improvement of focus is now somewhat non-primitive. We must not forget that what happens in (7.6) is the same as in (7.7). That is, when we recognise that the head application of an applicative term is not a value application, we are able to reorganise the applicative term so that the head application becomes simpler. The meaning of h in both $\lambda\mathcal{N}h$ and $\lambda\mathcal{P}h$ is to simplify the head application and ultimately turn it into a value application. However, in $\lambda\mathcal{N}h$, the effect is that the value application becomes even more deeply buried (because the applicative term becomes longer). In $\lambda\mathcal{P}h$, by the fortunate conjugation with the focus on the head application, the value application is brought closer to the surface of the applicative term.

It is no surprise, then, that this effect of bringing the value application closer to the surface has applications in the setting of weak head evaluation. This is what we are going to study in the next section.

7.2 Call-by-name abstract machines

In this section, we show that the λ-calculi we have defined in Chapter 3 for the canonical fragment have the remarkable property that their reductions rules may be seen as transitions rules for call-by-name abstract machines. In the case of $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$, this is exactly so. An explanation for this is already present in the observation that lists in Herbelin-style calculi model stacks in environment machines [Curien and Herbelin, 2000]. In the case of $\lambda\mathcal{P}h\mathtt{x}$, there is only a feature that is not immediately modelled by a reduction rule of the calculus, namely the search for a t buried inside a mid-cut like

$$t\{x_1 := u_1\}...\{x_n := u_n\} \ .$$

Here it is easy to speculate that the problem lies in the fact that there is no independent syntactic class for lists of bindings (*i.e.* environments) in $\lambda\mathcal{P}h\mathtt{x}$, as there would be if explicit substitutions in this calculus were in the style of the $\lambda\sigma$-calculus [Abadi et al., 1991].

Simple machines

We show what abstract machines are associated with $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$. These machines will be *simple*, in the sense that if the program is not fully evaluated, then it is itself the redex to be reduced next. Therefore, redexes do not have to be searched.

Definition 20 *In λ, the* weak head *(or* call-by-name*) reduction (notation: $\to_{CBN}$) is the least binary relation $\to$ on terms closed under*

$$\beta\,\frac{}{(\lambda x.M)N \to M[N/x]} \qquad Elim1\,\frac{M \to M'}{MN \to M'N}$$

A λ-term is a weak head normal form *(abbrev. whnf) if it is of the form $xN_1...N_k$ or $\lambda x.M$ (any $x, N_1, ..., N_k, M$ in λ).*

Relation $\to_{CBN}$ is actually a partial function also named CBN . A λ-term M is a whnf iff $CBN(M)$ is undefined. We define the weak head (or CBN) reduction of M as the repeated application of CBN starting from M. This process either terminates with a whnf of M, or diverges. The following basic result asserts the *completeness* of $\to_{CBN}$.

Theorem 15 *In λ, if $M \to_{\beta}^{*} N$ and N is a whnf, then the weak head reduction of M terminates (with a whnf of M).*

Therefore, CBN may be seen as a rudimentary machine. We may use terminology accordingly. For instance, we may say that CBN is *loaded* with M, meaning that CBN is applied to M. Although rudimentary, the machine will always obtain the whnf of a given term, if there is one to be found.

Definition 21 *In $\lambda\mathcal{N}$, we define the* weak head *(or* call-by-name*) reduction (notation: $\to_{CBN}$) over terms and over application as the least pair of binary relations $\to$ closed under*

$$\beta 1\,\frac{}{app((\lambda x.M)N) \to M[N/x]} \qquad App\,\frac{A \to A'}{app(A) \to app(A')}$$

$$\beta 2\,\frac{}{((\lambda x.M)N)N' \to M[N/x]@N'} \qquad AElim1\,\frac{A \to A'}{AN \to A'N}$$

Definition 22 *In $\lambda\mathcal{P}$, the* weak head *(or* call-by-name*) reduction (notation: $\to_{CBN}$) is the least binary relation $\to$ on terms closed under*

$$\beta 1 \, \frac{}{(\lambda x.t)(u \cdot []) \to subst(u,x,t)}$$

$$\beta 2 \, \frac{}{(\lambda x.t)(u \cdot (v :: l)) \to insert(v,l,subst(u,x,t))}$$

A $\lambda\mathcal{P}$-term is a whnf if it is of the form x or $x(u \cdot l)$ or $\lambda x.t$ (any x, u, l, t in $\lambda\mathcal{P}$).

In $\lambda\mathcal{P}$, the weak head reduction is simply the notions of reduction βi, *i.e.* $\to_{\beta i}$ restricted to the "top level" or "empty context".

Relation $\to_{CBN}$ in $\lambda\mathcal{P}$ is again a partial function also named CBN . A $\lambda\mathcal{P}$-term t is a whnf iff $CBN(t)$ is undefined. We define the weak head (or CBN) reduction of t as the repeated application of CBN starting from t. This process either terminates with a whnf of t, or diverges. Again, we regard CBN as a rudimentary machine.

We now prove that the isomorphism between $\to_\beta$ in λ and in $\lambda\mathcal{N}$ restricts to weak head reduction, and later we prove the same for $\to_\beta$ in $\lambda\mathcal{N}$ and $\lambda\mathcal{P}$.

To begin with, we need a restricted form of Lemma 39.

Lemma 63 *In $\lambda\mathcal{N}$, if $M \to_{CBN} M'$, then $M@N \to_{CBN} M'@N$.*

Proof: It is an adaptation of the proof of part 1. of Lemma 39, and is by case analysis of M.

$M = x$ and $M = \lambda x.M_0$ are vacuous cases. Let $M = app(A)$. Then, again there are two subcases.

Subcase 1: $A \to_{CBN} A'$ and $M' = app(A')$. Follows by closure under $AElim1$.

Subcase 2: $A = (\lambda x.M_0)N_0$ and $M' = M_0[N_0/x]$. Follows by closure under $\beta 2$. ■

Theorem 16 *The following holds:*

1. *$M_1 \to_{CBN} M_2$ in λ iff $\mathcal{N}M_1 \to_{CBN} \mathcal{N}M_2$ in $\lambda\mathcal{N}$.*

2. *$M_1 \to_{CBN} M_2$ in $\lambda\mathcal{N}$ iff $|M_1| \to_{CBN} |M_2|$ in λ.*

Proof: 1. "Only if": the proof is by induction on $M_1 \rightarrow_{CBN} M_2$ in λ, as in the proof of Theorem 3. However, only first and third cases are relevant. The former now follows by closure of $\rightarrow_{CBN}$ in $\lambda\mathcal{N}$ under $\beta 1$, whereas the latter requires closure under *App* and Lemma 63.

2. "Only if": the claim is proved together with the claim that if $A_1 \rightarrow_{CBN} A_2$ in $\lambda\mathcal{N}$, then $|A_1| \rightarrow_{CBN} |A_2|$ in λ by simultaneous induction on $M_1 \rightarrow_{CBN} M_2$ and $A_1 \rightarrow_{CBN} A_2$ in $\lambda\mathcal{N}$, as in the proof of Theorem 2. This time, only cases $\beta 1$, *App*, $\beta 2$ and *AElim*1 are relevant. Cases $\beta 1$ and $\beta 2$ follow by closure of $\rightarrow_{CBN}$ in λ under β. Case *AElim*1 requires closure under *Elim*1 and IH2. Case *App* is by IH2.

The "if" part of 1. follows from the "only if" part of 2. and $|\mathcal{N}(M)| = M$, whereas part "if" of 2. follows from the "only if" part of 1. and $\mathcal{N}|M| = M$. ■

We now need a restricted form of Lemma 43.

Lemma 64 *If $A \rightarrow_{CBN} A'$, then $\Theta'(A, l) \rightarrow_{CBN} \Theta'(A', l)$.*

Proof: Again by a straightforward induction on l. Case $l = []$ requires closure of $\rightarrow_{CBN}$ in $\lambda\mathcal{N}$ under *App*, whereas case $l = u_0 :: l_0$ requires closure under *AElim*1. ■

Theorem 17 *The following holds:*

1. *$M \rightarrow_{CBN} M'$ in $\lambda\mathcal{N}$ iff $\Psi M \rightarrow_{CBN} \Psi M'$ in $\lambda\mathcal{P}$.*

2. *$t \rightarrow_{CBN} t'$ in $\lambda\mathcal{P}$ iff $\Theta t \rightarrow \Theta t'$ in $\lambda\mathcal{P}$.*

Proof: 1. "Only if": the claim is proved together with the claim that if $A \rightarrow_{CBN} A'$ in $\lambda\mathcal{N}$, then $\Psi'(A, l) \rightarrow_{CBN} \Psi(M', l)$ (all l) in $\lambda\mathcal{P}$, by simultaneous induction on $M \rightarrow_{CBN} M'$ and $A \rightarrow_{CBN} A'$, as in the proof of Theorem 4. This time, only cases i, iii, iv and $vii-a$ are relevant. Cases i and iv require top level $\rightarrow_{\beta_H}$-steps, whereas case iii and $vii - a$ follow by IH2.

2. "Only if": As in case i of Theorem 5. Instead of Lemma 43, use Lemma 64.

The "if" part of 1. follows from the "only if" part of 2. and $\Theta\Psi M = M$, whereas part "if" of 2. follows from the "only if" part of 1. and $\Psi\Theta t = t$. ■

Recall that $\mathcal{P} : \lambda \to \lambda\mathcal{P}$ is $\Psi \circ \mathcal{N}$ and its inverse is denoted $\mathcal{Q}$. Observe that both $\mathcal{P}$ and $\mathcal{Q}$ send whnfs to whnfs. We say that they *preserve whnf*.

One may use CBN in $\lambda\mathcal{P}$ for reducing λ-terms to whnf. Given M in λ, load the machine of $\lambda\mathcal{P}$ with $\mathcal{P}(M)$ and perform in $\lambda\mathcal{P}$ the weak head reduction. If this terminates with u, say, return $\mathcal{Q}u$. Thus, the machine is correct.

Theorem 18 (Completeness) *In $\lambda\mathcal{P}$, if $t \to_\beta^* u$ and u is a whnf, then the weak head reduction of t terminates (with a whnf of t).*

Proof: Since $\mathcal{Q}$ is an isomorphism and preserves whnf, $\mathcal{Q}t \to_\beta^* \mathcal{Q}u$ in λ and $\mathcal{Q}u$ whnf. By completeness of $\to_{CBN}$ in λ, the weak head reduction of $\mathcal{Q}t$ terminates. By part 1. of Theorems 16 and 17, and the fact that $\mathcal{P}$ preserves whnfs, the weak head reduction of t terminates. ■

There is an advantage of CBN in $\lambda\mathcal{P}$ over CBN of λ: in the former, we do not have to search for the redex to be reduced next. If a term is not a whnf, the term itself is the redex to be reduced next.

Now, how about $\lambda\mathcal{P}h$? Is there a simple weak head evaluator associated with this calculus? The answer is affirmative.

Definition 23 *In $\lambda\mathcal{P}h$, the* weak head *(or* call-by-name*) reduction (notation: $\to_{CBN}$) is the least binary relation $\to$ on terms closed under*

$$\beta 1\ \frac{}{(\lambda x.t)(u \cdot []) \to subst(u, x, t)}$$

$$\beta 2\ \frac{}{(\lambda x.t)(u \cdot (v :: l)) \to subst(u, x, t)(v \cdot l)}$$

$$h\ \frac{}{t(u \cdot l)(u' \cdot l') \to t(u \cdot append(l, u' :: l'))}$$

A $\lambda\mathcal{P}$-term is a whnf if it is of the form x or $x(u \cdot l)$ or $\lambda x.t$ (any x, u, l, t in $\lambda\mathcal{P}h$).

In $\lambda\mathcal{P}h$, the weak head reduction is simply the union of the notions of reduction βi and h, *i.e.* $\rightarrow_{\beta i}$ and $\rightarrow_h$ restricted to the "top level" or "empty context".

Relation $\rightarrow_{CBN}$ in $\lambda\mathcal{P}h$ is again a partial function also named CBN . A $\lambda\mathcal{P}h$-term t is a whnf iff $CBN(t)$ is undefined. We define the weak head (or CBN) reduction of t as the repeated application of CBN starting from t. This process either terminates with a whnf of t, or diverges. Again, we regard CBN as a simple machine.

Recall mapping $(\_)^- : \lambda\mathcal{P}h \rightarrow \lambda\mathcal{P}$.

Lemma 65 *The weak head reduction of t in $\lambda\mathcal{P}h$ terminates iff the weak head reduction of t^- in $\lambda\mathcal{P}$ terminates. Moreover, if the former terminates with u, the latter terminates with u^-.*

Proof: One goes back to Proposition 5 and Lemma 19 and checks that a $\rightarrow_{\beta i}$-step in $\lambda\mathcal{P}h$ (at the top level) is mapped by $(\_)^-$ to a similar step in $\lambda\mathcal{P}$ (at the top level) and that a $\rightarrow_h$-step in $\lambda\mathcal{P}h$ is collapsed by $(\_)^-$ in $\lambda\mathcal{P}$. Moreover, $(\_)^-$ preserves whnf. This is sufficient for the "only if" part and the second statement.

As to the "if" part, first observe that, if $t^- \rightarrow_{CBN} v$ in $\lambda\mathcal{P}$, then there is t_k such that $t \rightarrow_{CBN} t_k$ in $\lambda\mathcal{P}h$ and $t_k^- = v$. Indeed, if t^- is not whnf, neither is t, because $(\_)^-$ preserves whnf. Since $\rightarrow_h$ is terminating and $(\_)^-$ collapses h-steps, there is $k \geq 1$ such that the weak head reduction of t looks like $t = t_0 \rightarrow_h \ldots \rightarrow_h t_{k-1} \rightarrow_{\beta i} t_k$. Moreover, $t_j^- = t^-$, when $0 \leq j < k$, and $t_{k-1}^- \rightarrow_{CBN} t_k^-$. But $\rightarrow_{CBN}$ is a function, hence $t_k^- = v$.

Now, suppose the weak head reduction of t^- in $\lambda\mathcal{P}$ terminates. From the last paragraph, it terminates with t_k^-, for some t_k such that $t \rightarrow_{CBN}^* t_k$. Is t_k a whnf in $\lambda\mathcal{P}h$? If it is not, the weak head reduction of t_k can at most perform $\rightarrow_h$-steps (otherwise t_k^- would not be whnf in $\lambda\mathcal{P}$). But $\rightarrow_h$ is terminating. ■

Hence, we can use the machine of $\lambda\mathcal{P}h$ to perform weak head reduction of $\lambda\mathcal{P}$. Load the former with a $\lambda\mathcal{P}$-term t and perform weak head reduction in $\lambda\mathcal{P}h$. If this terminates with u, return u^-. This works because of the previous lemma and $t^- = t$. Thus, the machine is correct.

Theorem 19 (Completeness) *In $\lambda\mathcal{P}h$, if $t \to^* u$ and u is a whnf, then the weak head reduction of t terminates (with a whnf).*

Proof: By the properties of $(\_)^-$, $t^- \to^* u^-$ and u^- is a whnf. By Theorem 18, the weak head reduction of t^- in $\lambda\mathcal{P}$ terminates. Hence, by Lemma 65, the weak head reduction of t in $\lambda\mathcal{P}h$ terminates. ■

Notice that the machine associated with $\lambda\mathcal{P}h$ is very much like a Krivine machine without environments [Krivine][Curien and Herbelin, 2000], except that the former runs programs in $\lambda\mathcal{P}h$, whereas the latter runs programs in λ (*i.e.* $\lambda\mathcal{G}$).

Similarly to $\lambda\mathcal{P}$, a term in $\lambda\mathcal{P}h$ is either a whnf or is itself the redex to be reduced next. In the machine for $\lambda\mathcal{P}$, this redex is always a βi-redex. In the machine for $\lambda\mathcal{P}h$, this is not the case, it may be a h-redex. But the effect of h-reduction is to bring the value application to the surface, that is, to bring the applicative term closer to the form of a βi-redex.

A Herbelin-style abstract machine

Now we want to obtain the weak head reducer associated with $\lambda\mathcal{P}h\mathtt{x}$. We will present the definition of the weak head reduction in $\lambda\mathcal{P}h\mathtt{x}$ in the form of an abstract machine, which we name the Herbelin-style abstact machine (HAM). The states, or dumps, of the HAM are defined as follows (where $Term$ is the class of terms of $\lambda\mathcal{P}h\mathtt{x}$):

$$
\begin{array}{rcl}
Dump & = & Term \times Environment \times Stack \\
Environment & = & Binding^* \\
Stack & = & Term^* \\
Binding & = & Var \times Term
\end{array}
$$

The main point is that dumps should be regarded as $\lambda\mathcal{P}h\mathtt{x}$-terms. To make this more apparent, we will use a notation for bindings, environments and stacks that makes the reading of dumps as terms easier. Environments are ranged over by e, e', etc. Bindings are written $\{x := t\}$ and ranged over by b. The cons of a new binding is written $\{x := t\}e$, the empty environment is written as a blank or as $-$ and append of environments as ee'. Stacks are ranged over by l, l', etc. Let d, d', etc. range over dumps. The cons of a new term is written as $u :: l$ and the empty stack as $[]$. The term associated to the dump

$$\langle t, \{x_1 := u_1\}...\{x_k := u_k\}, l\rangle$$

is

$$(...(t\{x_1 := u_1\})...\{x_k := u_k\}) \odot l$$

where

$$\begin{array}{rcl} t \odot [] & = & t \\ t \odot (u :: l) & = & t(u \cdot l) \ . \end{array}$$

If we allow tb to represent a mid-cut (although, crucially, in $\lambda\mathcal{P}h\mathtt{x}$ there is no separate syntactic class of bindings), then the reading of dump as terms is easily specified by: read $\langle t, b_1...b_k, l\rangle$ as $(...(tb_1)...b_k) \odot l$.

The transition rules of the machine may be found in Table 7.1. Each row, except the rows defining the stopping or *final* states, defines a transition rule and displays the states of the machine before and after the transition. We write $d \to d'$ when d and d' are related by some transition rule. In rule $H3$, if $b = \{x := v\}$, $sub(b, l)$ means $sub(v, x, l)$.

It should be clear that, when dumps are seen as $\lambda\mathcal{P}h\mathtt{x}$-terms, there is a correspondence between transition rules and reduction rules of $\lambda\mathcal{P}h\mathtt{x}$ as follows:

$$\begin{array}{ccccccc} H1 & - & \mathtt{x2} & \quad & H2 & - & \mathtt{x1} \\ H3 & - & \mathtt{x4} & & H4 & - & h \\ H5 & - & \mathtt{x3} & & H6 & - & bj \end{array}$$

Rule $H7$ corresponds to no reduction rule. As opposed to rule $H4$ (the other rule that pushes an object on top of a list) the effect of $H7$ is not observed in $\lambda\mathcal{P}h\mathtt{x}$.

$(H1)$	x	$\{y := t\}e$	l	x	e	l
$(H2)$	x	$\{x := t\}e$	l	t	e	l
$(STOP)$	x		l	$-$	$-$	$-$
$(H3)$	$t(u \cdot l)$	be	l'	$tb(ub \cdot sub(b, l))$	e	l'
$(H4)$	$t(u \cdot l)$		l'	t		$u :: append(l, l')$
$(STOP)$	$\lambda x.t$	e	$[]$	$-$	$-$	$-$
$(H5)$	$\lambda x.t$	be	$u :: l$	$\lambda x.tb$	e	$u :: l$
$(H6)$	$\lambda x.t$		$u :: l$	t	$\{x := u\}$	l
$(H7)$	$t\{x := u\}$	e	l	t	$\{x := u\}e$	l

Table 7.1: Transition rules for the Herbelin-style Abstract Machine

The reason is clear. The stack of arguments is a syntactic object of its own in the syntax of $\lambda\mathcal{P}h\mathtt{x}$, whereas the list of bindings is not. Rule $H6$ corresponds to either $b1$ or $b2$ according to whether l is $[]$ or not.

Contrary to the simple machines for $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$, the redex to be reduced next is not always found at the top level. However, observe that such desirable situation only fails for transition rules that correspond to $\mathtt{x}i$ reduction rules.

Let us give some operational intuitions for the HAM. We have in mind starting the machine with an *initial* state, *i.e.* a dump of the form $\langle t, -, []\rangle$, for some $\lambda\mathcal{P}h\mathtt{x}$-term t. A dump consists of a term, which we might call the *program*, a list of bindings, called the environment, and a stack of arguments. The program operates over the environment and the stack, but this is only a partially correct metaphor because rules $H3$ and $H5$ use the program as a temporary store. What is true is that, in the same way as we use the stack for storing arguments as we go deeper in head-cuts (rule $H4$), we use the environment for storing bindings as we go deeper in mid-cuts (rule $H7$). Rules $H1$ and $H2$ perform a look-up in the environment. Rule $H6$ is the usual rule creating a new binding and popping the stack.

The peculiarity of the HAM is seen in rules $H3$ and $H5$ and it derives from the fact that dumps must be interpreted as $\lambda\mathcal{P}h\mathtt{x}$-terms, and in $\lambda\mathcal{P}h\mathtt{x}$ we cannot separate the term and the bindings of a mid-cut. A transition $H3$ is a preliminary step for $H4$. The point is that, if the environment is not empty, there is no *single* reduction rule in $\lambda\mathcal{P}h\mathtt{x}$ that allows the argument to pass over the bindings. That is why, by repeated application of $H3$, we "hide" the environment, performing at the same time its duplication. Similarly, a transition $H5$ is a preliminary step for $H6$. Before the application of $H6$ we have to hide the environment behind the λ by repeated application of $H5$.

Given a dump d, at most one transition rule applies. We also want to apply rules to $\lambda\mathcal{P}h\mathtt{x}$-terms. This is done by fixing a canonical way of seeing a term t as a dump $d(t)$. Outer bindings and "arguments" go to the environment and the stack, respectively. For instance $(\lambda x.t)\{y := u\}(u \cdot l)$ becomes $\langle \lambda x.t, \{y := u\}, u :: l\rangle$. Therefore, given a term, at most one of the rules $H1$ to $H6$ applies.

Definition 24 *In $\lambda\mathcal{P}h\mathtt{x}$, the* weak head *(or* call-by-name*) reduction (notation: $\rightarrow_{CBN}$) is defined as follows: $t \rightarrow_{CBN} t'$ if $d(t) \rightarrow_{Hi} d'$, for some $i \in \{1, 2, 3, 4, 5, 6\}$ and d' such that d' is t' when seen as a term. A $\lambda\mathcal{P}h\mathtt{x}$-term is a* whnf *if it has one of the forms x, $x(u \cdot l)$ or $(\lambda x.t)e$ (any x, u, l, t, e in $\lambda\mathcal{P}h\mathtt{x}$).*

Of course, $(\lambda x.t)e$ is an iterated mid-cut. Relation $\rightarrow_{CBN}$ in $\lambda\mathcal{P}h\mathtt{x}$ is again a partial function also named CBN . A $\lambda\mathcal{P}h\mathtt{x}$-term t is a whnf iff $CBN(t)$ is undefined. We define the weak head (or CBN) reduction of t as the repeated application of CBN starting from t. This process either terminates with a whnf of t, or diverges.

This time, if we regard CBN as a machine, it is a version HAM' of the HAM in which $H7$ steps are silent. Of course, the HAM and the HAM' are essentially the same machine because the HAM can only perform a finite number of consecutive $H7$ transitions and a $H7$ transition does not change the reading of the dump as a term.

The difference between the HAM and the HAM' is quite revealing. It shows that, contrary to $\lambda\mathcal{P}$ and $\lambda\mathcal{P}h$, there is an ingredient missing in $\lambda\mathcal{P}h\mathtt{x}$, namely the immediate availability of the scope t of a "closure"

$$t\{x_1 := u_1\}...\{x_n := u_n\} \ .$$

In the HAM', the search for t is implicit. In the HAM, it is done by the rule $H7$, which is not a reduction rule of $\lambda\mathcal{P}h\mathtt{x}$.

We now prove correctness and completeness of the HAM'. The proofs are analogous to Lemma 65 and Theorem 19. Recall mapping $(\_)^\flat : \lambda\mathcal{P}h\mathtt{x} \rightarrow \lambda\mathcal{P}h$.

Let $\mathtt{x} = \mathtt{x1} \cup \mathtt{x2} \cup \mathtt{x3} \cup \mathtt{x4}$. Then, $\rightarrow_{\mathtt{x}}$ is terminating. One sees this by going to Chapter 3 and recalling how $\mathtt{x}i$ steps are mapped first to $\overline{\lambda}\mathcal{P}h\mathtt{x}$ and later to $\overline{\lambda}_3$ and finally reusing the termination result in [Dyckhoff and Urban, 2001] for explicit substitution rules and commuting conversion.

Lemma 66 *The weak head reduction of t in $\lambda\mathcal{P}h\mathtt{x}$ terminates iff the weak head reduction of $t^\flat$ in $\lambda\mathcal{P}h$ terminates. Moreover, if the former terminates with u, the latter terminates with $u^\flat$.*

Proof: One goes back to Proposition 10 and Lemma 30 and checks that a $\rightarrow_{\beta i}$ or $\rightarrow_h$ step in $\lambda\mathcal{P}h\mathtt{x}$ (at the top level) is mapped by $(\_)^\flat$ to a similar step in $\lambda\mathcal{P}h$ (at the top level) and that a $\rightarrow_{\mathtt{x}i}$-step in $\lambda\mathcal{P}h\mathtt{x}$ is collapsed by $(\_)^\flat$ in $\lambda\mathcal{P}h$. Moreover, $(\_)^\flat$ preserves whnf. This is sufficient for the "only if" part and the second statement.

As to the "if" part, first, by the same argument as in Lemma 65, using termination of $\rightarrow_{\mathtt{x}}$, one proves that if $t^\flat \rightarrow_{CBN} v$ in $\lambda\mathcal{P}h$, then there is t_k such that $t \rightarrow_{CBN} t_k$ in $\lambda\mathcal{P}h\mathtt{x}$ and $t_1^\flat = v$.

Now, suppose the weak head reduction of $t^\flat$ in $\lambda\mathcal{P}h$ terminates. From the last paragraph, it terminates with $t_k^\flat$, for some t_k such that $t \rightarrow^*_{CBN} t_k$. Is t_k a whnf in $\lambda\mathcal{P}h\mathtt{x}$? If it is not, the weak head reduction of t_k can at most perform $\rightarrow_{\mathtt{x}i}$-steps (otherwise $t_k^\flat$ would not be whnf in $\lambda\mathcal{P}h$). But $\rightarrow_{\mathtt{x}}$ is terminating. ∎

Hence, since $t^\flat = t$ when t is in $\lambda\mathcal{P}h$, both the HAM and the HAM' are correct machines for performing weak head reduction of $\lambda\mathcal{P}h$.

Theorem 20 (Completeness) *In $\lambda\mathcal{P}h\mathtt{x}$, if $t \to^* u$ and u is a whnf, then the weak head reduction of t terminates (with a whnf).*

Proof: By the properties of $(\_)^\flat$, $t^\flat \to^* u^\flat$ and $u^\flat$ is a whnf. By Theorem 19, the weak head reduction of $t^\flat$ in $\lambda\mathcal{P}h$ terminates. Hence, by Lemma 66, the weak head reduction of t in $\lambda\mathcal{P}h\mathtt{x}$ terminates. ■

One of the good characteristics of the HAM is its neat organisation. First, the relation between head-cuts and the stack, and between mid-cuts and the environment. Second, the understanding of rules $H1$, $H3$ and $H5$ as preliminary steps for $H2$, $H4$ and $H6$, respectively.

Another characteristic is the proximity with $\lambda\mathcal{P}h\mathtt{x}$, and hence with cut elimination. From a theoretical point of view, this is positive. From a practical point of view, this brings inefficiency. Since environments do not constitute a separate syntactic class in the syntax of $\lambda\mathcal{P}h\mathtt{x}$, the duplication of the environment by $H3$ is stepwise and rule $H6$ requires the environment to be hidden (by rule $H5$), so that it does not get in the way to the next argument.

Chapter 8

Conclusions

In this chapter we list the contributions of this thesis and propose future work.

8.1 Contributions

The contributions of this thesis are the following.

First, a systematic definition of calculi of cut-elimination for the canonical fragment. We considered several right protocols of cut-elimination, with increasing degree of explicitness and stepwise character, starting from $\lambda\mathcal{P}$ - a new isomorphic copy of λ-calculus as a calculus of cut-elimination - and ending in $\lambda\mathcal{P}h\mathtt{x}$. When the remaining meta-operators of the latter calculus are internalised, we obtain $\overline{\lambda}\mathcal{P}h\mathtt{x}$, a system close to Herbelin's $\overline{\lambda}$-calculus, but already outside the canonical fragment. This systematic process identified in $\overline{\lambda}\mathcal{P}h\mathtt{x}$ which (and a small number of) inter-permutation of cuts are required for simulating full β-reduction.

Second, a comprehensive study of the relationship between cut-elimination for the canonical fragment and normalisation. Results here include the isomorphism between $\lambda\mathcal{P}$ and λ, the identification of the ::-free fragment of the canonical fragment, the definition of a generalisation of Prawitz's mapping to non-normal proofs, the fact that both Gentzen's and Prawitz's mappings establish an isomorphism between normalisation and a certain cut-elimination procedure, and the identification of the relation between Prawitz's mapping and Gentzen's mapping.

Third, the introduction of a new proof-theoretical tool, namely certain conservative extensions of natural deduction (and corresponding extension of λ-calculus) based on the idea of a built-in distinction between applicative term and application, on the one hand, and between head and tail application, on the other hand. We proposed a new conceptual organisation of proof-systems and λ-calculi based on the observation that λ-calculus is in the intersection of the ::-free and tail-application-free fragments.

Fourth, a reassessment of the relationship between cut-elimination in the canonical fragment and normalisation, by virtue of the introduction of the mentioned conservative extensions of natural deduction. This included the definition of a mapping Θ that may be seen as a new assignment of λ-terms, taken from the extensions of the λ-calculus, to proofs in the canonical fragment of sequent calculus. The main property of this assignment is to be an isomorphism, both in the sense of sound bijection of proofs and isomorphism of normalisation procedures. Moreover, we had to consider the issue of explicitness also in the natural deduction side. The conclusion is that (the existence of) isomorphism Θ is insensitive to a varying degree of explicitness in the cut-elimination and normalisation procedures it bridges.

Fifth, a study of the proof-theoretical status of explicit substitutions, concluding that the issue of explicit substitution in a term calculus is correlated with the inclusion or not of a cut constructor in a proof-system, and that both issues are orthogonal to the sequent calculus versus natural deduction divide.

Sixth, contributions to the computational interpretation of sequent calculus. On the one hand, the λ-calculi for the canonical fragment were shown to be extensions of the λ-calculus, with a constructor for applicative terms. Relatively to calculi in the natural deduction side, the difference is that applicative terms are built with the help of evaluation contexts, instead of tail applications; and that, instead of being buried, the head application is available at the "surface" of the applicative term. This structural difference explains that some reduction rules of λ-calculi for the canonical fragment may be interpreted as transition rules for abstract call-by-name machines.

Seventh, from a strict λ-calculus point of view, the proposal of several extensions based on the idea of applicative term, and the proofs of conservativeness, subject reduction, confluence and strong normalisation of typable terms.

8.2 Future work

In this thesis, we restricted ourselves to (1) intuitionistic implicational logic. (2) right protocols of cut-elimination. (3) the canonical fragment of sequent calculus. Obeying such stringent constraints is methodologically correct. One must be modular and separate the problems. Accordingly, the next step should be to relax constraint (3) and study right protocols of cut-elimination on the whole set of sequent calculus derivations for intuitionistic implicational logic. We would be particularly interested in investigating whether there is an extension of natural deduction matching this step, and whether the good properties of Θ resist. Of course, when constraint (3) is relaxed, the permutability problem is back again. However, we believe that a clearer understanding of the permutation-free fragment will allow a fresh attack on that problem.

We regard the study of the computational interpretation of sequent calculus as a contribution to an useful, and perhaps unexpected, extension of the Curry-Howard isomorphism. Unexpected, because we are not referring to the "cross fertilisation" [Cardelli, 1997] between type theory and programming languages, by which strong logics offer sophisticated type systems and, conversely, programming features like, say, concurrency, challenge a logical understanding. The extension is of a different kind, observable even if we keep contenting ourselves with simple types and intuitionistic implicational logic. It is an extension from natural deduction to other kinds of proof-systems. What varies is not the logic, it is the system in which we write the proofs and where the normalisation procedure lives. And correspondingly, in the computational side, we seem to find different, useful approaches to λ-calculus with benefits for its *implementation.* Think of Hilbert systems and combinators, think of the canonical fragment of sequent calculus and environment machines, think even of linear logic and proof-nets, on the one hand, and graph reduction [Wadsworth, 1971] and sharing graphs [Asperti and Guerrini, 1998], on the other hand. We believe that the most important thing to do in the future is the full investigation of this new dimension in the expansion of the Curry-Howard correspondence.

Bibliography

[Abadi et al., 1991] Abadi, M., Cardelli, L., Curien, P.-L., and Lévy, J.-J. (1991). Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416.

[Abramsky, 1993] Abramsky, S. (1993). Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57.

[Asperti and Guerrini, 1998] Asperti, A. and Guerrini, S. (1998). *The optimal implementation of programming languages*, volume 45 of *Cambridge Tracts in Theoretical ComputerScience*. Cambridge University Press.

[Barendregt, 1984] Barendregt, H. (1984). *The Lambda Calculus*. North-Holland.

[Barendregt, 1992] Barendregt, H. (1992). Lambda calculi with types. In S. Abramsky and D. M. Gabbay and T. S. E. Maibaum, editor, *Handbook of Logic in Computer Science*, volume 2, pages 118–309. Oxford University Press.

[Barendregt and Ghilezan, 2000] Barendregt, H. and Ghilezan, S. (2000). Lambda terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(1):121–134.

[Bloo, 1997] Bloo, R. (1997). *Preservation of termination for explicit substitution*. PhD thesis, Eindhoven University of Technology.

[Cardelli, 1997] Cardelli, L. (1997). Type systems. In *Handbook of Computer Science and Engineering*. CRC Press.

[Cerrito and Kesner, 1999] Cerrito, S. and Kesner, D. (1999). Pattern matching as cut elimination. In *Proceedings of 14th annual IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 98–108.

[Curien et al., 1996] Curien, P.-L., Hardin, T., and Lévy, J.-J. (1996). Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):389–402.

[Curien and Herbelin, 2000] Curien, P.-L. and Herbelin, H. (2000). The duality of computation. In *Proceedings of International Conference on Functional Programming 2000*. IEEE.

[Curry and Feys, 1958] Curry, H. B. and Feys, R. (1958). *Combinatory Logic*. Noth Holland, Amsterdam.

[Danos et al., 1995] Danos, V., Joinet, J.-B., and Schellinx, H. (1995). *LKQ* and *LKT*: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In Girard, J.-Y., Lafont, Y., and Regnier, L., editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 211–224. Cambridge University Press.

[Danos et al., 1997] Danos, V., Joinet, J.-B., and Schellinx, H. (1997). A new deconstructive logic: linear logic. *The Journal of Symbolic Logic*, 62(2):755–807.

[di Cosmo and Kesner, 1997] di Cosmo, R. and Kesner, D. (1997). Strong normalization of explicit substitutions via cut elimination in proof nets. In *Proceedings of LICS'97*.

[Dragalin, 1988] Dragalin, A. (1988). *Mathematical Logic: Introduction to Proof Theory*, volume 67 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, Rhode Island.

[Dyckhoff, 1997] Dyckhoff, R. (1997). Dragalin's proof of cut-admissibility for the intuitionistic sequent calculi G3i and G3i'. Technical Report CS/97/8, Computer Science Division, St. Andrews University.

[Dyckhoff and Pinto, 1998] Dyckhoff, R. and Pinto, L. (1998). Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60:107–118.

[Dyckhoff and Pinto, 1999] Dyckhoff, R. and Pinto, L. (1999). Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155.

[Dyckhoff and Urban, 2001] Dyckhoff, R. and Urban, C. (2001). Strong normalisation of Herbelin's explicit substitution calculus with substitution propagation. In *Fourth International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (WESTAPP'01)*.

[Espírito Santo, 2000] Espírito Santo, J. (2000). Revisiting the correspondence between cut-elimination and normalisation. In *Proceedings of ICALP'2000*, volume 1853 of *Lecture Notes in Computer Science*. Springer-Verlag.

[Felleisen et al., 1986] Felleisen, M., Friedman, D., Kohlbecker, E., and Duba, B. (1986). Reasoning with continuations. In *1st Symposium on Logic and Computer Science*. IEEE.

[Field and Harrison, 1988] Field, A. J. and Harrison, P. G. (1988). *Functional Programming*. Addison-Wesley.

[Gallier, 1993] Gallier, J. (1993). Constructive logics. Part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:248–339.

[Gentzen, 1935] Gentzen, G. (1935). Untersuchungen über das logische schliessen (Investigations into logical deduction). *Mathematische Zeitschrift*, pages 176–210,405–431. Translation in [Szabo, 1969].

[Girard, 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50(1):1–102.

[Girard, 1991] Girard, J.-Y. (1991). A new constructive logic: classic logic. *Mathematical Structures in Computer Science*, 1(3):255–296.

[Girard et al., 1989] Girard, J.-Y., Lafont, Y., and Taylor, P. (1989). *Proofs and Types*. Cambridge University Press.

[Griffin, 1990] Griffin, T. (1990). A formulae-as-types notion of control. In *ACM Conf. Principles of Programming Languages*. ACM Press.

[Henderson, 1980] Henderson, P. (1980). *Functional Programming: application and implementation*. Prentice-Hall International.

[Herbelin, 1995] Herbelin, H. (1995). A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In Pacholski, L. and Tiuryn, J., editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag.

[Hindley, 1997] Hindley, J. R. (1997). *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

[Howard, 1980] Howard, W. A. (1980). The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editor, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 480–490. Academic Press, New York.

[Joachimski and Matthes] Joachimski, F. and Matthes, R. (Accepted for publication). Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*.

[Kesner et al., 1995] Kesner, D., Puel, L., and Tannen, V. (1995). A typed pattern calculus. *Information and Computation*, 124(1).

[Kleene, 1952] Kleene, S. (1952). Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, pages 1–26.

[Krivine] Krivine, J.-L. (Unpublished). Un interpréteur du lambda-calcul.

[Lafont, 1989] Lafont, Y. (1989). Functional programming and linear logic. Lecture notes for the Summer School on Functional Programming and Constructive Logic, Glasgow, September 1989.

[Landin, 1964] Landin, P. J. (1964). The mechanical evaluation of expressions. *Computer Journal*, 6:308–320.

[Lescanne, 1994] Lescanne, P. (1994). From $\lambda\sigma$ to $\lambda\upsilon$: a journey through calculi of explicit substitutions. In *Proceedings of 21st Annual ACM symposium on Principles of Programming Languages (POPL'94)*.

[McCarthy, 1960] McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195.

[Mints, 1996] Mints, G. (1996). Normal forms for sequent derivations. In P. Odifreddi, editor, *Kreiseliana*, pages 469–492. A. K. Peters, Wellesley, Massachusetts.

[Negri and von Plato, 2001] Negri, S. and von Plato, J. (2001). *Structural Proof Theory*. Cambridge.

[Pagano, 1998] Pagano, B. (1998). An explicit natural deduction. In *Fist International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (WESTAPP'98)*.

[Parigot, 1992] Parigot, M. (1992). $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In *Int. Conf. Logic Prog. Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*. Springer Verlag.

[Peyton Jones, 1987] Peyton Jones, S. L. (1987). *The implementation of functional languages*. Prentice-Hall.

[Plotkin, 1975] Plotkin, G. (1975). Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159.

[Pottinger, 1977] Pottinger, G. (1977). Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357.

[Prawitz, 1965] Prawitz, D. (1965). *Natural Deduction. A Proof-Theoretical Study*. Almquist and Wiksell, Stockholm.

[Rose, 1996a] Rose, K. (1996a). Explicit substitutions: Tutorial & survey. Technical Report LS-96-3, BRICS.

[Rose, 1996b] Rose, K. H. (1996b). *Operational reduction models for functional programming languages*. PhD thesis, University of Copenhagen.

[Szabo, 1969] Szabo, M., editor (1969). *The collected papers of Gerhard Gentzen*. North Holland.

[Troelstra and Schwitchtenberg, 2000] Troelstra, A. and Schwitchtenberg, H. (2000). *Basic Proof Theory*. Cambridge University Press, second edition.

[Ungar, 1992] Ungar, A. (1992). *Normalization, Cut-eliminations and the Theory of Proofs*. Number 28 in CSLI Lecture Notes.

[Urban and Bierman, 1999] Urban, C. and Bierman, G. (1999). Strong normalisation of cut-elimination in classical logic. In *Proceedings of TLCA'99*, Lecture Notes in Computer Science. Springer-Verlag.

[Vestergaard and Wells, 1999] Vestergaard, R. and Wells, J. (1999). Cut rules and explicit substitutions. In *Second International Workshop on Explicit Substitutions*.

[Wadler, 1993] Wadler, P. (1993). A Curry-Howard isomorphism for sequent calculus. Manuscript.

[Wadsworth, 1971] Wadsworth, C. P. (1971). *Semantics and pragmatics of the lambda calculus*. D.Phil. thesis, Oxford.

[Zucker, 1974] Zucker, J. (1974). The correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–112.

Index