The λ_{Δ}-calculus

Niels Jakob Rehof & Morten Heine Sørensen

DIKU, Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen Ø, Denmark
Electronic mail: `rehof@diku.dk` & `rambo@diku.dk`

October 26, 1993

Contents

1 Background and motivation

The first subsection describes previous work in the Curry-Howard Isomorphism. The second subsection describes our contribution: a typed λ-calculus with a number of desirable properties, not all shared by the systems mentioned in the first subsection.

The Curry-Howard Isomorphism and classical logic

The so-called *Curry-Howard Isomorphism* states a correspondence between typed λ-calculi and systems of formal logic. Curry noted the connection between a typed combinator language and a Hilbert style formulation of minimal implicational logic in [Cur58]; Howard subsequently developed the idea and extended it to Heyting arithmetic in a paper from 1969, later published as [How80]. Other authors are also occassionally credited for work on the isomorphism in the Seventies and Eighties; [Bar84] appendix A.3 contains a brief history with references, and [Hin86] 14D contains yet additional references.

At the heart of the isomorphism is the perception of proofs as functions, as formalized in Kleene's Realizability Interpretation [Kle52]. At the syntactic level this phenemenon is reflected by the precise correspondence between normalization of (natural deduction) proofs and reduction on typed λ-terms. Specifically, if proof trees are represented by so-called constructions and normalization is stated in terms of constructions, then normalization and reduction are syntactically identical.[1]

Traditionally the isomorphism has been taken to hold for intuitionistic logics only. However, recently work has been done to extend the isomorphism to classical logics.

In *Computer Science* this idea originates with Griffin who in 1990, in an attempt to incorporate control operators into the world of typed λ-calculi, discoverered that Felleisen's $\mathcal{C}$-operator could be typed by the classical double-negation elimination rule [Gri90]. Using this rule does, however, lead to certain difficulties because typing is not in general preserved under reduction ("Failure of Subject Reduction.") This defect was repaired by Griffin via a so-called computational simulation.

Later, Murthy overcame the same difficulties by changing the type system into a so-called pseudo-classical logic [Mur90]. Applying conservativity results of classical logics over corresponding minimal logics Murthy showed in [Mur90] that for a certain class of classically provable formulae the Realizability Interpretation remains sound. This was done using CPS-translations of control operator calculi into pure λ-calculi.

However, none of the authors consider the exact relation between reduction in their respective systems and classical proof normalization as found in

[1]According to Gallier, "the correspondence between proof normalization and term reduction is the deepest and most fruitful aspect of the Curry-Howard Ismorphism" [Gal92] p8.

the proof theory literature, or to classical proof theory in general. This in fact applies to *all* the works on typing control operators that we have investigated [Gri90], [Mur90], [Mur91], [Mur91b], [Mur92a], [Dub90], [Har92], [Wer92], [Bar92]. Considering the importance of this particular aspect of the isomorphism we feel that the relation between reduction on constructions and proof normalization should be investigated. It is debatable whether an extension of the isomorphism can be claimed if a close correspondence is not found.

Another connection in the isomorphism is that between the definability of connectives, *e.g.* conjunction, in logic and the definability of constructions, *e.g.* pairs and projections, in λ-calculus. Griffin studied the question informally in [Gri90] but found that the desired derived reduction rules did not generally hold.

Both Griffin's and Murthy's work draw on the fundamental work of Felleisen and his co-workers on λ-calculi with control operators, which is conducted in an untyped setting *e.g.* [Fel87b]. Felleisen devised a control calculus, an extension of the λ-calculus, and carried out what could aptly be called *Plotkin's program* (see [Plo75]) for the study of the relation between calculi and programming languages (see also [Fel87b].) Originally, the rules of the control calculus were split into so-called *reduction rules* and *computation rules*. The former are completely compatible (applicable in any context), whereas the latter are restricted to the *top-level* of a program, *i.e.* applicable in the empty context only. However, the context sensitivity of the computation rules entails that the equational theory of the calculus is not sound with respect to observational equivalence (see [Fel89].) Therefore, in [Fel89], a revised calculus called $\lambda_v - C(d)$ was devised in which all rules are compatible and which is operationally sound. The relation between the revised calculus and the original one can be described as a computational simulation, similar to the one employed (independently) by Griffin. In fact, one could type Felleisen's revised calculus with Griffin's rule (double-negation elimination) with preservation of subject reduction.

For the present purposes we wish to draw attention to two points in particular concerning the $\lambda_v - C(d)$-calculus. Firstly, if a term contains a control application, then any reduction by the control rules will again lead to an expression with control in it. Clearly, it would be desirable if control could be eliminated at least for expressions in which control applications are intended to model meaningful control operations, such as `catch/throw`-pairs. Attempts to *extend* the $\lambda_v - C(d)$-calculus with rules that perform such eliminations are reported in [Fel89] to have failed because they lead to a break-down of the Church-Rosser property.[2] In this situation it seems natural to ask whether it would be possible to solve the problem by *restricting* the power of the control operators.

The second point we want to focus on is that the control rules of the $\lambda_v - C(d)$-calculus are not weakly normalizing due to the rule C_{top}, designed to

[2]See also [Mor93] where a similar situation (break-down of the Church-Rosser property) is reported in an attempt to devise a calculus for `call/cc`.

simulate the computation rule. This suggests that the calculus does not have an interesting weakly normalizing *typed subcalculus* which in turn precludes a correspondence to proof normalization. This is in contrast with the pure λ-calculus which has the simply typed λ-calculus as a strongly normalizing subcalculus, obtained by imposing a type discipline, not changing the reduction rules.

In *Logic* there does not seem to have been a similar effort to extend the isomorphism to classical logic. The texts on proof classical proof normalization that we are aware of [Pra65], [Pra71], [Sel86], [Sel89], [Sta91] invariably state their result in terms of standard natural deduction systems without constructions. Prawitz briefly considers constructions in [Pra71] but only for the minimal fragment of his classical systems.

Gabbay and de Queiroz have investigated extensions of the isomorphism to various non-intuitionistic logics, in particular the so-called resource logics [Gab92]. Their work is unrelated to the works on classical proof normalization mentioned above as well as to the works on typing control operators in Computer Science.

Recently our attention has been directed towards the works of Adrian Rezus by Chet Murthy and Matthias Felleisen. It would seem that there are indeed close connections between our work and his work, which we are currently studying.

Our contribution

In this paper we attempt to bring together the line of research in typed and untyped control operators conducted in Computer Science with that of classical proof normalization conducted in Logic. Specifically we describe a construction which

- is a call-by-name (in the sense of [Plo75]) control operator which is more than powerful enough to express the catch-throw paradigm using no simulation
- is Church-Rosser
- is Compatible (all reduction rules apply in arbitrary contexts)
- can be typed with a standard classical inference rule
- has Subject Reduction (using this typing)
- is Strongly Normalizing (using this typing)
- corresponds closely to known classical proof normalization procedures
- can be used to define pairs, projections and other constructions

Whereas Felleisen set out to build a calculus to serve as a reasoning system strong enough to model known control constructs such as `call/cc`, we take as our starting point a reasoning system enjoying a number of desirable properties.

The remainder of the paper is organized into sections as follows.

Section 2 presents the untyped λ_Δ-calculus, λ_Δ, which is the untyped λ-calculus extended with two control operators Δ, ∇ (the latter defined in terms of the former.) Adding basic constants and functions to λ_Δ, we get $\lambda_{\Delta\delta}$. We also consider a typed version, $\lambda_\Delta^{\perp,\supset}$, of $\lambda_\Delta$which we call the classically typed λ_Δ-calculus. The relation between the two is roughly the same as between the untyped λ-calculus and the simply typed λ-calculus. We also consider the extension of $\lambda_\Delta^{\perp,\supset}$with pair types and disjoint sum types and with types with first-order quantifiers $\forall$ and $\exists$. The result is $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. Finally we consider a number of related systems.

Section 3 shows that $\lambda_\Delta$and $\lambda_{\Delta\delta}$are Church-Rosser and that $\lambda_\Delta^{\perp,\supset}$in addition has Subject Reduction and is Strongly Normalizing.

Section 4-6 can be perceived as interpretations (in a very informal sense) of the formal system of section 2.

Section 4: (Control operator calculus) shows that $\lambda_\Delta$can express the catch and throw paradigm, and investigates more closely the relation to Felleisen's control operators $\mathcal{F}$ and $\mathcal{C}$ and the factorized versions thereof.

Section 5: (Classical proof theory) develops classical proof theory for $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$and related systems, and derives a number of well-known corollaries in the style of [Pra65] III. We also consider the exact correspondence between $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$and the classical proof normalization procedures of Stålmarck [Sta91] and Prawitz [Pra65], and we consider the role of the Inversion Principle well-known from minimal proof theory.

Section 6: (Explicit type coercion system) considers the notions of definability in logic (definability of connectives, eg. conjunction) and λ-calculus (definability of constructions eg. pairs and projections) and shows that in $\lambda_\Delta^{\perp,\supset}$one can define pairs, projections and other constructions. This section also considers the definition of other constructions in a style similar to that employed in Girard's F_2. Via this we are lead to the consideration of Δ and ∇ as subtyping operators. This in turn leads to a new perspective on the Inversion Principle.

Appendices: The paper ends with four appendices. The first three contain detailed induction proofs of lemmas and theorems which were merely stated in the text. The last appendix contains the first steps in an execution of Plotkin's program as carried out by Felleisen (see above.) Specifically, we give a Standardization Theorem and consider the Machine-Calculus correspondence question.

2 The λ_Δ-calculus

In this section we review the untyped λ_Δ-calculus, λ_Δ, along with a typed version of it, $\lambda_\Delta^{\perp,\supset}$, which we call the classically typed λ_Δ-calculus. The relation between the two is roughly the same as the relation between the untyped λ-calculus and the simply typed λ-calculus.

Subsection 1 describes the construction language and reduction relation of the untyped λ_Δ-calculus. Subsection 2 describes the extension of $\lambda_\Delta$with base constants and functions, called $\lambda_{\Delta\delta}$. Subsection 3 describes the type language and type inference system of the classically typed λ_Δ-calculus, $\lambda_\Delta^{\perp,\supset}$; the term language and reduction relation for $\lambda_\Delta^{\perp,\supset}$is the same as for the untyped calculus λ_Δ. Subsection 4 describes the extension of $\lambda_\Delta^{\perp,\supset}$with conjunction, disjunction and first-order quantifiers, $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. Subsection 5 describes a number of subsystems of $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$.

2.1 The untyped λ_Δ-calculus: λ_Δ

Definition 1 (Constructions.) We assume an infinite set of variables ranged over by lowercase letters *e.g.* x, y, z. Uppercase letters *e.g.* L, M, N range over constructions.

$$M \quad ::= \quad x \mid \lambda x.M \mid M\ N \mid \Delta x.M$$

This language of constructions is called Λ. □

Notation and Convention 1 We take the notion of free and bound variable for granted. The set of free variables of a construction M is written $FV(M)$. We identify constructions which differ only in the choice of names for bound variables, and it is always understood that bound variables are chosen to be different from any free variables in any set of constructions under consideration.

Substitution of construction N for all occurrences of a free variable x in a construction M is written $M\{x := N\}$. $M\{x := N, y := K\}$ denotes the simultaneous substitution of x and y. Substitution binds tighter than all other constructs; thus $\lambda x.M\{y := N\}$ means $\lambda x.(M\{y := N\})$, and $M\ N\{x := K\}$ means $M\ (N\{x := N\})$. In view of the above conventions, capturing of free variables by substitution is always impossible.

The usual conventions concerning the binding and associativity of abstraction and application are adopted. The conventions for λ carry over to Δ. □

Definition 2 The construction $\nabla(M)$ is an abbreviation for $\Delta d.M$ where $d \notin FV(M)$. □

Common Lisp [Ste84] programmers may read $\Delta x.x\ M$ as `catch` x `in` M and $\nabla(x\ N)$ as `throw` $x\ N$. ML [Mil90] programmers may read $\Delta x.x\ M$ as M `handle` $X(v)$ `=>` v and $\nabla(x\ N)$ as `raise` $X(N)$ where X is declared `exception` X `of` $\mathcal{P}$, $\mathcal{P}$ being the intented type of N. Readers familiar with Felleisen's control operators [Fel87b] may think of $\Delta x.M$ and $\nabla(K)$ as $\mathcal{F}(\lambda x.M)$ and $\mathcal{A}(K)$, respectively.

DEFINITION **3** (Reduction on constructions.)

$$\begin{array}{lll}
(1) & (\lambda x.M)\ N & \rightarrow & M\{x := N\} \\
(2) & (\Delta x.M)\ N & \rightarrow & \Delta z.M\{x := \lambda y.z\ (y\ N)\} \\
(3) & \Delta x.x\ M & \rightarrow & M \text{ provided } x \notin FV(M) \\
(4) & \Delta x.x\ \nabla(x\ M) & \rightarrow & M \text{ provided } x \notin FV(M)
\end{array}$$

□

NOTATION AND CONVENTION **2** Here and later we call $\rightarrow$ the *notion of reduction.* The compatible closure of $\rightarrow$ is denoted $\rhd$. We also call $\rhd$ the *one-step reduction.* A construction M with no N such that $M \rhd N$ is called *normal.* The reflexive, transitive closure of $\rhd$ is denoted $\rhd^*$. We also call $\rhd^*$ the *reduction relation.* If $M_0 \rhd M_1 \rhd \ldots \rhd M_n$, we write $M_0 \rhd^n M_n$. The reflexive, transitive, symmetric closure of $\rhd$ is denoted $=$. □

By the abbreviation for $\nabla(M)$ we have the derived rule

$$(5) \quad (\nabla(M))\ N \ \rightarrow \ \nabla(M)$$

In terms of the catch-throw paradigm, rule (2) states that `catch` x `in` M reduces to M if M contains no throw to M (normal return), and rule (3) states that `catch` x `in throw` x M reduces to M if M contains no throw to M (exceptional return.) Rule (5) allows a throw to skip one level of the surrounding context thereby approaching its corresponding catch so that rule (3) eventually applies. That $\rhd$ is compatible could also have been stated in terms of contexts.

DEFINITION **4** (General contexts)

$$C \ ::= \ [] \mid \lambda x.C \mid C\ M \mid M\ C \mid \Delta x.C$$

$C[M]$ denotes the result of subsituting M for $[]$ in C. □

Now $\rhd$ is the smallest relation such that $C[M] \rhd C[N]$ for all C whenever $M \rightarrow N$.

NOTATION AND CONVENTION **3** Subscripting a relation ($\rightarrow, \rhd, \rhd^*, =$) with one or more numbers indicate that the reduction may only use the rules indicated by these numbers. For instance, $(\lambda x.\lambda y.M)\ N\ K \rhd^*_{1,2} M\{x := N\}\{y := K\}$ but not $(\lambda x.\lambda y.M)\ N\ K \rhd^*_{3,4} M\{x := N\}\{y := K\}$. □

2.2 Adding basic constants and functions: $\lambda_{\Delta\delta}$

We consider the following enriched language of constructions, called Λ_δ:

DEFINITION **5**

$$M \quad ::= \quad \phi \mid b \mid x \mid \lambda x.M \mid M\ N \mid \Delta x.M$$

Here $\phi \in \mathit{FConsts}$ ranges over a collection of basic functions, $b \in \mathit{BConsts}$ over a collection of basic constants such as natural numbers. Values, ranged over by V, are:

$$V \quad ::= \quad \phi \mid b \mid x \mid \lambda x.M$$

We let V_0 range over those values that are closed terms. As a parameter of the enriched calculus we assume a function $\delta : \mathit{FConsts} \times \mathit{BConsts} \rightarrow V_0$. $\square$

We introduce booleans and the conditional as defined notions, using `True` $\equiv \lambda x.\lambda y.x$, `False` $\equiv \lambda x.\lambda y.y$, `if` $B\ M_1\ M_2 \equiv B\ M_1\ M_1$. To the rules of λ_Δ we add the following two new rules.

DEFINITION **6**

$$\begin{array}{lll} (6) & \phi\, b & \rightarrow \quad \delta(\phi, b) \\ (7) & \phi\, \Delta x.M & \rightarrow \quad \Delta\kappa.M\{x := \lambda f.\kappa\ (\phi\ f)\} \end{array}$$

$\square$

The scope of the old rules for λ_Δ are to be extended to the definition of Λ_δ, so we have for instance $(\Delta x.M)\, b \rhd \Delta\kappa.M\{x := \lambda f.\kappa\ (f\ b)\}$. Also, by the abbreviation for $\nabla(M)$ we have the derived rule

$$(5') \quad \phi\ \nabla(N) \quad \rightarrow \quad \nabla(N)$$

Note that we have `if` $\nabla(N)\ M_1\ M_2 \rhd^* \nabla(N)$ and $\phi\ \nabla(N) \rhd \nabla(N)$.

The function constants are curried, so that e.g. for $+ \in \mathit{FConsts}$ we read $(+\ a\ b)$ as $((+\ a)\ b)$, and we write $\delta(+, a) = +_a$ with the standard meaning.

2.3 The classically typed λ_Δ-calculus: $\lambda_\Delta^{\perp,\supset}$

DEFINITION **7** (Type language.[3]) We assume an infinite set of of type variables ranged over by P, Q, R. We use $\mathcal{P}$,$\mathcal{Q}$,$\mathcal{R}$ to range over types. The type language contains: (*i*) *type variables*; (*ii*) the *constant* falsity; (*iii*) *function type* $\supset$.

$$\mathcal{P} ::= P \mid \perp \mid \mathcal{P} \supset \mathcal{Q}$$

$\square$

[3]Due to the Curry-Howard Isomorphism we often use different notions interchangably: type/formula, reduction/normalization, *etc.*

In the inference system we use the means of managing assumptions usually employed in proof-theoretical texts (see [Gal92] for a comparison of different techniques.)

Definition 8 (Type Inference system.)

$$\dfrac{\begin{matrix}[x:\mathcal{P}]\\ \vdots \\ M:\mathcal{Q}\end{matrix}}{\lambda x.M:\mathcal{P}\supset\mathcal{Q}}\ (\supset\mathrm{I}) \qquad \dfrac{M:\mathcal{P}\supset\mathcal{Q}\quad N:\mathcal{P}}{M\ N:\mathcal{Q}}\ (\supset\mathrm{E}) \qquad \dfrac{\begin{matrix}[x:\mathcal{P}\supset\bot]\\ \vdots \\ M:\bot\end{matrix}}{\Delta x.M:\mathcal{P}}\ (\bot_c)$$

We take such notions as *derivation, open assumption, closed assumption* etc. for granted, see *e.g.* [Pra71]. For a construction M and formula $\mathcal{P}$ the existence of a derivation of $M:\mathcal{P}$ using the above rules with open assumptions contained in Γ, is stated $\Gamma\vdash M:\mathcal{P}$. □

While the simply typed λ-calculus (without pairs and sums), $\lambda^{\bot,\supset}$, via the Curry-Howard isomorphism corresponds to a natural deduction formulation of the implicational fragment of minimal propositional logic, $\lambda_{\Delta}^{\bot,\supset}$corresponds to the implicational fragment of classical propositional logic, see [Pra65] pp20-21.

2.4 The full first-order system: $\lambda_{\Delta}^{\bot,\supset,\wedge,\vee,\forall,\exists}$

In this subsection we consider the incorporation into $\lambda_{\Delta}^{\bot,\supset}$of pair and sum type $\wedge$, $\vee$ and first-order quantifiers $\forall,\exists$ with associated constructions, reductions and type inference rules. The result is $\lambda_{\Delta}^{\bot,\supset,\wedge,\vee,\forall,\exists}$.

For each notion in $\lambda_{\Delta}^{\bot,\supset}$, *e.g.* $\rightarrow$, there is a similar notion in $\lambda_{\Delta}^{\bot,\supset,\wedge,\vee,\forall,\exists}$. We use the same symbol in both systems as well as in the systems we shall consider in subsection 4. Whenever there is risk of confusion we make explicit which system is in play.

Definition 9 (First-order constructions.) We assume an infinite set of variables ranged over by lowercase letters *e.g.* x,y,z. We use uppercase letters *e.g.* L,M,N as metavariables for constructions. The construction language contains (*i*) variables; (*ii*) *constructors*: λ-abstraction, pair, injection, $\forall$-abstraction, brackets; (*iii*) corresponding *destructors*: λ-application, projection, case analysis, $\forall$-application and pattern-matching on brackets; (*iv*) the control operator Δ:

$$\begin{array}{rcl} N & ::= & x \\ & | & \lambda x.N \mid M\ N \\ & | & <M_1,M_2> \mid \pi_1(M) \mid \pi_2(M) \\ & | & \mathrm{in}_1(M) \mid \mathrm{in}_2(M) \mid \mathrm{case}(L;x_1.M;x_2.M_2) \\ & | & \lambda^{\forall}\alpha.M \mid M\ \tau \\ & | & [<\tau,M>] \mid \mathrm{let}\ [<\alpha,y>]=M\ \mathrm{in}\ N \\ & | & \Delta x.M \end{array}$$

$\nabla(M)$ is an abbreviation for $\Delta d.M$ where $d \notin FV(M)$. □

Definition 10 (Reduction on first-order constructions.)

$$\begin{array}{lll}
(1a) & (\lambda x.M)\ N & \rightarrow\ M x := N \\
(1b) & \pi_i(< M_1, M_2 >) & \rightarrow\ M_i \\
(1c) & \text{case}(\text{in}_i(L); x_1.M_1; x_2.M_2) & \rightarrow\ M_i\{x_i := L\} \\
(1d) & (\lambda^{\forall}\alpha.M)\ \tau & \rightarrow\ M\{\alpha := \tau\} \\
(1e) & \text{let } [< \alpha, y >] =< \tau, M > \text{ in } N & \rightarrow\ N\{\alpha := \tau, y := N\} \\
(2a) & (\Delta x.M)\ N & \rightarrow\ \Delta u.M\{x := \lambda y.u\ (y\ N)\} \\
(2b) & \pi_i(\Delta x.M) & \rightarrow\ \Delta u.M\{x := \lambda y.u\ \pi_i(y)\} \\
(2c) & case(\Delta x.M; y_1.M_1; y_2.M_2) & \rightarrow\ \Delta u.M\{x := \lambda y.u\ case(y; y_1.M_1; y_2.M_2)\} \\
(2d) & \Delta x.M\ \tau & \rightarrow\ \Delta u.M[\lambda y.u\ (y\ \tau)/x] \\
(2e) & let\, [< \alpha, v >] =\ \Delta x.M\ in\ N & \rightarrow\ \Delta u.M\{x := \lambda y.u\ let\, [< \alpha, v >] =\ y\ in\ N\} \\
(3) & \Delta x.x\ M & \rightarrow\ M \text{ provided } x \notin FV(M) \\
(4) & \Delta x.x\ \nabla(x\ M) & \rightarrow\ M \text{ provided } x \notin FV(M)
\end{array}$$

Each of the above constructions on the left hand side of $\rightarrow$ is called a *redex*. □

Using the notion of one-level destructor contexts the rules (2a)-(2e) can be collapsed into one rule.

Definition 11 (One-level Destructor contexts.)

$$\begin{array}{lll}
D^1 & ::= & [] \mid D^1\ M \mid \pi_i(D^1) \mid \text{case}(D^1; x_1.N_1; x_2.N_2) \mid D^1\ \tau \mid \text{let } [< \alpha, y >] = D^1 \text{ in } N \\
D & ::= & D^1 \mid D^1[D]
\end{array}$$

where as usual $D^1[D]$ denotes the result of substituting D for $[]$ in D^1. □

Now (2a)-(2e) reads

$$(2) \quad D^1[\Delta x.M] \ \rightarrow\ \Delta u.M\{x := \lambda y.u\ D^1[y]\}$$

By the abbreviation for $\nabla(M)$ we have the derived rule

$$(5) \quad D^1[\nabla(M)] \ \rightarrow\ \nabla(M)$$

which can also be written explicitely in five rules (5a)-(5e) without one-level contexts.

That $\rhd$ is compatible could also have been made explicit via the concept of general contexts.

Definition 12 (General contexts.)

$$\begin{array}{lll}
C & ::= & [] \\
 & \mid & \lambda x.C \mid C\ M \mid M\ C \\
 & \mid & < C, M > \ \mid\ < M, C > \ \mid \pi_i(C) \\
 & \mid & \text{in}_i(C) \mid \text{case}(C; x_1.N_1; x_2.N_2) \mid \text{case}(M; x_1.C; x_2.N_2) \mid \text{case}(M; x_1.N_1; x_2.C) \\
 & \mid & \lambda^{\forall}\alpha.C \mid C\ \tau \\
 & \mid & [< \tau, C >] \mid \text{let } [< \alpha, y >] = C \text{ in } N \mid \text{let } [< \alpha, y >] = M \text{ in } C \\
 & \mid & \Delta x.C
\end{array}$$

$C[M]$ denotes the result of subsituting M for $[]$ in C. $\square$

Now $\rhd$ is the smallest relation such that $C[M] \rhd C[N]$ for all C whenever $M \rightarrow N$.

Definition **13** (First-order formulae.) We assume an infinite set of eigenvariables ranged over by α, β and n-ary function symbols for each $n \geq 0$ ranged over by ϕ^n. The eigenterm language ranged over by τ consists of:

$$\tau \ ::= \ \alpha \mid \phi^n(\tau_1, \ldots, \tau_n)$$

We assume an infinite set of of n-ary predicate symbols ranged over by P^n for each $n \geq 0$. The set of atomic formulae ranged over by A consists of:

$$A ::= P^n(\tau_1, \ldots, \tau_n)$$

The set of first-order formulae, ranged over by $\mathcal{P}, \mathcal{Q}, \mathcal{R}$, contains: (i) *atomic formulae*; (ii) the *constant:* falsity; (iii) *connectives:* implication, conjunction and disjunction; (iv) *quantifiers:* universal and existential quantifier.

$$\mathcal{P} ::= A \mid \bot \mid \mathcal{P} \supset \mathcal{Q} \mid \mathcal{P} \wedge \mathcal{Q} \mid \mathcal{P} \vee \mathcal{Q} \mid \forall \alpha.\mathcal{P} \mid \exists \alpha.\mathcal{P}$$

We take the notions of free and bound eigenvariables and related hygiene conventions for granted. $\square$

DEFINITION **14** (Inference system for $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee,\forall,\exists}$.)

$$\begin{array}{c}[x:\mathcal{P}]\\ \vdots \\ \dfrac{M:\mathcal{Q}}{\lambda x.M:\mathcal{P}\supset\mathcal{Q}}\end{array} \qquad \dfrac{M:\mathcal{P}\supset\mathcal{Q}\quad N:\mathcal{P}}{M\;N:\mathcal{Q}}$$

$$\dfrac{M_1:\mathcal{P}_1\quad M_2:\mathcal{P}_2}{<M_1,M_2>:\mathcal{P}_1\wedge\mathcal{P}_2} \qquad \dfrac{L:\mathcal{P}_1\wedge\mathcal{P}_2}{\pi_1(L):\mathcal{P}_1} \quad \dfrac{L:\mathcal{P}_1\wedge\mathcal{P}_2}{\pi_2(L):\mathcal{P}_2}$$

$$\dfrac{M_1:\mathcal{P}_1}{\mathrm{in}_1(M_1):\mathcal{P}_1\vee\mathcal{P}_2} \quad \dfrac{M_2:\mathcal{P}_2}{\mathrm{in}_2(M_2):\mathcal{P}_1\vee\mathcal{P}_2} \qquad \dfrac{L:\mathcal{P}\vee\mathcal{Q}\quad \begin{array}{c}[x_1:\mathcal{P}]\\ \vdots\\ M_1:\mathcal{R}\end{array}\quad \begin{array}{c}[x_2:\mathcal{Q}]\\ \vdots\\ M_2:\mathcal{R}\end{array}}{\mathrm{case}(L;x_1.M_1;x_2.M_2):\mathcal{R}}$$

$$\dfrac{M:\mathcal{Q}}{\lambda^{\forall}\alpha.M:\forall\alpha.\mathcal{Q}} \qquad \dfrac{M:\forall\alpha.\mathcal{Q}}{M\;\tau:\mathcal{Q}[\tau/\alpha]}$$

$$\dfrac{M:\mathcal{Q}[\tau/\alpha]}{[<\tau,M>]:\exists\alpha.\mathcal{Q}} \qquad \dfrac{M:\exists\alpha.\mathcal{Q}\quad \begin{array}{c}[y:\mathcal{Q}]\\ \vdots\\ N:\mathcal{P}\end{array}}{\mathrm{let}\ [<\alpha,y>]=M\ \mathrm{in}\ N:\mathcal{P}}$$

$$\begin{array}{c}[x:\mathcal{P}\supset\perp]\\ \vdots \\ \dfrac{M:\perp}{\Delta x.M:\mathcal{P}}\end{array}$$

Disregarding the last rule, which we call $\perp_c$, the rules on the left [right] hand are said to be *introduction* [*elimination*] rules. Each introduction rule has in its conclusion a unique connective, and each elimination rule has in the leftmost premise a unique connective. The rules are named accordingly, *e.g.* the top leftmost rules is called $\supset I$, and the bottom rightmost is called $\exists E$. The type of leftmost premise of every elimination rule is called the *major premise*; the types of the remaining premises, if any, are called *minor premises.*

In $\forall$-introduction α must not occur free in any open assumption, and in $\exists$-elimination α must not occur free in $\mathcal{P}$ or in any open assumption. For a first order construction and formula M, $\mathcal{P}$, the existence of a derivation of $M:\mathcal{P}$ using the above rules with open assumptions contained in Γ, is stated $\Gamma\vdash M:\mathcal{P}$. □

This completes our description of CQ.

2.5 Relatives

We now introduce a number of other systems. For each of these there is a similar notion of $\Gamma \vdash M : \mathcal{P}$, $M \to N$, $M \rhd N$, $M \rhd^* N$, $M = N$. This is not made explicit in definitions.

The system $\lambda_\Delta^{\perp,\supset,\forall}$is obtained from $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$by erasing everything from $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$which concerns $\wedge, \vee, \exists$: (i) erase $\wedge, \vee, \exists$ from the type language; (ii) erase pairs, projections, injections, case-analysis, brackets and let from the construction language; (iii) remove $\wedge I$, $\wedge E$, $\vee I$, $\vee E$, $\exists I$, $\exists E$ from the type inference system; (iv) remove reduction rules (1b), (1c), (1e), (2b), (2c), (2e);

The system $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$is obtained from $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$by replacing atomic formulae in the definition of the type language with 0-ary predicate symbols, P, and erasing everything concerning $\forall, \exists$ from the type language, construction language, type inference system and notion of reduction in a manner similar to that explained above.

The system $\lambda_\Delta^{\perp,\supset}$already considered in subsection 2 can be obtained from $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$by erasing everything concerning $\wedge, \vee$ from the type language, construction language, type inference system and notion of reduction.

For each system $\lambda_\Delta^{\perp,\supset}$, $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$, $\lambda_\Delta^{\perp,\supset,\forall}$, $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$there is another system $\lambda^{\perp,\supset}$, $\lambda^{\perp,\supset,\wedge,\vee}$, $\lambda^{\perp,\supset,\forall}$, $\lambda^{\perp,\supset,\wedge,\vee,\forall,\exists}$, respectively, obtained by erasing $\Delta x.M$ from the construction language, removing the reduction rules (2)-(4) for $\Delta x.M$ and the inference rule $\perp_c$. (Here $\lambda^{\perp,\supset}$is what is traditionally called the simply typed λ-calculus.)

Via the Curry-Howard isomorphism, the system $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$is (full) first-order classical logic while $\lambda_\Delta^{\perp,\supset,\forall}$is the implicational (and universal) fragment of $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. Similarly $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$is (full) propositional classical logic while $\lambda_\Delta^{\perp,\supset}$is the implicational fragment of $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$. Both of the implicational fragments can work as the corresponding full system through suitable definitions, see section 6. The systems λ are minimal logics. For these, the implicational and implicational/universal fragments cannot play the role of the full systems, see section 6.

3 Fundamental Properties of λ_Δ, $\lambda_{\Delta\delta}$ and $\lambda_\Delta^{\perp,\supset}$

In this chapter we prove, after some preliminaries (first section), that the Church-Rosser property (abbreviated CR) holds for λ_Δ (second section) as well as for $\lambda_{\Delta\delta}$ (third section) and that the properties of Subject Reduction (abbreviated SR), Church-Rosser and Strong Normalization (abbreviated SN) hold for the typed calculus $\lambda_\Delta^{\perp,\supset}$ (fourth section.)

Preliminaries

A one step reduction $\rhd$ has the CR property if, whenever $M_1 \rhd^* M_2$ and $M_1 \rhd^* M_3$ there is an M_4 such that $M_2 \rhd^* M_4$ and $M_3 \rhd^* M_4$. For the Church-Rosser proof it is convenient to use the notation of [Bar84] for the *diamond property*: Given a notion of reduction R over Λ we write $R \models \diamond$ if, for all $M, M_1, M_2 \in \Lambda$, $M\,R\,M_1$ and $M\,R\,M_2$ implies that there is an $M_3 \in \Lambda$ such that $M_1\,R\,M_3$ and $M_2\,R\,M_3$. Then R is CR iff $R^* \models \diamond$. Note that $R \models \diamond$ implies $R^* \models \diamond$. We say that R_1 and R_2 *commute* if, whenever $M_1\,R_1\,M_2$ and $M_1\,R_2\,M_3$ there is an M_4 such that $M_2\,R_2\,M_4$ and $M_3\,R_1\,M_4$.

The proofs for the CR properties make heavy use of so-called parallel reductions all of which enjoy certain closure properties. In order to save space and to make comparison of the various parallel reductions easier we shall sometimes separate out a set of conditions satisfied by all of these relations. We define:

DEFINITION **15** (Basic conditions for parallelism)
Let R be any binary relation over Λ (Λ_δ). Then we say that R satisfies the basic conditions for parallelism, abbreviated $BCP(R)$, if the following hold for R, for any $M, N, M', N' \in \Lambda(\Lambda_\delta)$:

$$\begin{array}{lll} (BCP1) & M\,R\,M & \\ (BCP2) & M\,R\,M', N\,R\,N' & \Rightarrow \quad (M\ N)\,R\,(M'\ N') \\ (BCP3) & M\,R\,M' & \Rightarrow \quad \lambda x.M\,R\,\lambda x.M' \\ (BCP4) & M\,R\,M' & \Rightarrow \quad \Delta x.M\,R\,\Delta x.M' \end{array}$$

□

In particular, any relation R such that $BCP(R)$ is reflexive and compatible.

3.1 CR Property for λ_Δ

This section presents a proof that the calculus λ_Δ has the Church-Rosser property. We state all the main lemmas but leave out the longer induction proofs. Full proofs for these are given in Appendix A.

The overall idea of the proof is as follows. We define two relations $\gg_\gamma$ and $\gg_\delta$ of parallel reduction [4] such that

[4]It is standard to call relations such as $\gg_\gamma$ and $\gg_\delta$ parallel reductions. But note that

1. $\gg_\gamma$ is CR
2. $\gg_\delta$ is CR
3. $\gg_\gamma^*$ and $\gg_\delta^*$ commute
4. $(\gg_\gamma \cup \gg_\delta)^* = \rhd^*$

Each of these parallel reductions represents a fragment of the calculus which has a fairly simple Church-Rosser proof. The CR-property for the full calculus can then be inferred by the standard argument from commutativity. We use the following version of the Hindley-Rosen Lemma:

LEMMA **1** *(Hindley-Rosen)*
Let R_1 and R_2 be two notions of reduction over Λ. Suppose

1. R_1 and R_2 are CR.

2. R_1^* and R_2^* commute.

Then $R_1 \cup R_2$ is CR.

PROOF: See [Bar84], 3.3.5. □

Then we can conclude, by the Hindley-Rosen Lemma, that $\gg_\gamma \cup \gg_\delta$ is CR, *i.e.* , $(\gg_\gamma \cup \gg_\delta)^* \models \diamond$, hence also $\rhd^* \models \diamond$, so $\rhd$ is CR.
We define the parallel reduction $\gg_\gamma$ as follows:

DEFINITION **16**

$$\begin{array}{lll}
(1) & M \gg_\gamma M & \\
(2) & M \gg_\gamma M', N \gg_\gamma N' & \Rightarrow \quad (\lambda x.M)\, N \gg_\gamma M'\{x := N'\} \\
(3) & M \gg_\gamma M' & \Rightarrow \quad \Delta x.x\, M \gg_\gamma M'\,, \quad x \notin FV(M) \\
(4) & M \gg_\gamma M' & \Rightarrow \quad \Delta x.x\, \nabla(x\, M) \gg_\gamma M'\,, \quad x \notin FV(M) \\
(5) & M \gg_\gamma M' & \Rightarrow \quad \nabla(M)\, N \gg_\gamma \nabla(M') \\
(6) & BCP(\gg_\gamma) &
\end{array}$$

□

The parallel reduction $\gg_\delta$ is defined as follows:

DEFINITION **17**

$$\begin{array}{lll}
(1) & M \gg_\delta M & \\
(2) & M \gg_\delta M', N \gg_\delta N' & \Rightarrow \quad (\Delta x.M)\, N \gg_\delta \Delta\kappa.M'\{x := \lambda f.\kappa\, (f\, N')\} \\
(3) & BCP(\gg_\delta) &
\end{array}$$

□

they are not reduction relations in the technical sense introduced above, since they are not transitive.

Note that $\gg_\gamma$ contains (in rule (5)) a special case of rule (2) of $\gg_\delta$. The detailed reason for choosing this split is technical; it ensures that commutativity goes through.

Now, we can prove :

LEMMA **2** $\gg_\gamma \models \diamond$.

PROOF: Given in Appendix A. □

LEMMA **3** $\gg_\delta \models \diamond$.

PROOF: Given in Appendix A. □

Now consider $\gg_\gamma$ and $\gg_\delta$ as notions of reduction on Λ, and define $\gg_\gamma^*$ and $\gg_\delta^*$ to be the reduction relations induced by $\gg_\gamma$ and $\gg_\delta$, respectively. Aiming for invocation of the Hindley-Rosen Lemma we would like to prove that $\gg_\gamma^*$ commutes with $\gg_\delta^*$. The following property makes this possible:

LEMMA **4** *For any $M_1, M_2, M_3 \in \Lambda$: if $M_1 \gg_\gamma M_2$ and $M_1 \gg_\delta M_3$ then there is an $M_4 \in \Lambda$ such that $M_3 \gg_\gamma^* M_4$ and $M_2 \gg_\delta M_4$.*

PROOF: Given in Appendix A. □

To get commutativity from Lemma 4 we go by the following lemma. To state it we introduce, for an arbitrary binary relation R, notation for the reflexive closure of R, denoted R^r, the transitive closure of R, denoted R^t, and the reflexive transitive closure of R, denoted R^{rt}.

LEMMA **5** *Let R_1 and R_2 be two binary relations over a set X. Suppose that, for any $x_1, x_2, x_3 \in X$, $x_1\ R_1\ x_2$ and $x_1\ R_2\ x_3$ imply that there is an x_4 such that $x_2\ R_1^{rt}\ x_4$ and $x_3\ R_2^r\ x_4$. Then R_1^{rt} and R_2^{rt} commute.*

PROOF: Diagram chase. See [Bar84], 3.3.6. □

Now we can prove

LEMMA **6** *$\gg_\gamma^*$ and $\gg_\delta^*$ commute.*

PROOF: Since $\gg_\delta$ is reflexive and $\gg_\gamma^*$ is also reflexive, the claim follows from Lemma 4 and Lemma 5. □

From the lemmas obtained so far we can prove the desired result:

THEOREM **1** *$\rhd$ is CR.*

PROOF: From Lemma 2 and Lemma 3 it follows that $\gg_\gamma$ and $\gg_\delta$ are CR. By Lemma 6 $\gg_\gamma^*$ and $\gg_\delta^*$ commute. Therefore, by the Hindley-Rosen Lemma, $\gg_\gamma \cup \gg_\delta$ is CR, that is, $(\gg_\gamma \cup \gg_\delta)^* \models \diamond$. But clearly, $\rhd^= \subseteq (\gg_\gamma \cup \gg_\delta) \subseteq \rhd^*$, and so $(\gg_\gamma \cup \gg_\delta)^* = \rhd^*$. Consequently, $\rhd$ is CR. □

3.2 CR Property for $\lambda_{\Delta\delta}$

Here we show that the δ-rules (rule (6) and rule (7)) can be added to λ_Δ under preservation of the CR-property. The technique used in this proof is similar to the previous Church-Rosser proof, in that we essentially use the Hindley-Rosen Lemma to show that the addition of the δ-rules is safe. The main problem here is to show that the addition of rule (7) is safe. Note that, in this section, $\rhd$ denotes the full reduction of $\lambda_{\Delta\delta}$.

We split the calculus $\lambda_{\Delta\delta}$in two, each part represented by a parallel reduction. Define the parallel reductions over Λ_δ, $\gg$ and $\gg_{\phi\Delta}$, as follows:

Definition 18 Let $\gg$ be the least relation over Λ_δ satisfying

$$\begin{array}{lll}
(1) & M \gg M', N \gg N' & \Rightarrow \quad (\lambda x.M)\, N \gg M'\{x := N'\} \\
(2) & M \gg M', N \gg N' & \Rightarrow \quad (\Delta x.M)\, N \gg \Delta\kappa.M'\{x := \lambda f.\kappa\,(f\,N')\} \\
(3) & M \gg M' & \Rightarrow \quad \Delta x.x\, M \gg M'\,, \quad x \notin FV(M) \\
(4) & M \gg M' & \Rightarrow \quad \Delta x.x\, \nabla(x\, M) \gg M'\,, \quad x \notin FV(M) \\
(5) & M \gg M' & \Rightarrow \quad \phi\, \nabla(M) \gg \nabla(M') \\
(6) & BCP(\gg) &
\end{array}$$

□

Definition 19 Let $\gg_{\phi\Delta}$ be the least relation over Λ_δ satisfying

$$\begin{array}{lll}
(1) & M \gg_{\phi\Delta} M' & \Rightarrow \quad \phi\, \Delta x.M \gg_{\phi\Delta} \Delta\kappa.M'\{x := \lambda f.\kappa\,(\phi\, f)\} \\
(2) & BCP(\gg_{\phi\Delta}) &
\end{array}$$

□

Lemma 7 *$\gg$ is CR.*

Proof: Define the relation $\gg_\rhd$ by removing condition (5) from definition 18, so $\gg_\rhd^* = \rhd_{1,2,3,4}^*$. Define also the relation $\gg_{\phi\nabla}$ to be the least relation over Λ_δ satisfying condition (5) of definition 18 and $BCP(\gg_{\phi\nabla})$. Clearly, $\gg = (\gg_\rhd \cup \gg_{\phi\nabla})$. Now, we can show that $\gg_\rhd$ and $\gg_{\phi\nabla}$ commute, that is: If $M_1 \gg_\rhd M_2$ and $M_1 \gg_{\phi\nabla} M_3$ then there exists an M_4 such that $M_2 \gg_{\phi\nabla} M_4$ and $M_3 \gg_\rhd M_4$. This is seen easily by case analysis:

If $M_1 \gg_\rhd M_2$ is either $(\lambda x.P)Q \gg_\rhd P'\{x := Q'\}$ or $(\Delta x.P)Q \gg_\rhd \Delta\kappa.P'\{x := \lambda f.\kappa\,(f\,Q')\}$ with $P \gg_\rhd P'$ and $Q \gg_\rhd Q'$ then $M_1 \gg_{\phi\nabla} M_3$ must be by $BCP2$, and the result follows by induction hypothesis together with compatibility and substitutivity of $\gg_{\phi\nabla}$.

If $M_1 \equiv \Delta x.x\, P$ or $M_1 \equiv \Delta x.x\, \nabla(x\, P)$ and $P \gg_\rhd P'$, $x \notin FV(P)$, yielding $M_1 \gg_\rhd M_2 \equiv P'$ (by rule (3) or (4) of $\gg$) then it must be the case that $M_1 \gg_{\phi\nabla} \Delta x.x\, P'' \equiv M_3$ or $M_1 \gg_{\phi\nabla} \Delta x.x\, \nabla(x\, P'') \equiv M_3$ with $P \gg_{\phi\nabla} P''$. The result then follows by induction.

If $M_1 \gg_\rhd M_2$ is $(BCP2)$ $M_1 \equiv P\, Q \gg_\rhd P'\{x := Q'\} \equiv M_2$ with $P \gg_\rhd P'$ and $Q \gg_\rhd Q'$ and $M_1 \gg_{\phi\nabla} M_3 \equiv \nabla(Q'')$ with $P \equiv \phi$ and $Q \gg_{\phi\nabla} Q''$, then the desired conclusion follows easily by induction hypothesis.

The rest of the cases involve uses of BCP-rules only and here the result follows trivially or by easy induction.

Since we already know from Theorem 1 that $\gg_{\triangleright}$ is CR and it is easy to verify that $\gg_{\phi\nabla} \models \diamond$ and so also $\gg_{\phi\nabla}$ is CR, it follows by Lemma 5 and the Hindley-Rosen Lemma that $\gg_{\triangleright} \cup \gg_{\phi\nabla}$ is CR and consequently so is $\gg$. □

Remark **1** Note that, in the proof above, it is critically important that the *Control-right* rule $\phi\,\nabla(M) \gg_{\phi\nabla} M'$, $M \gg_{\phi\nabla} M'$, requires the operator to be a basic function, ϕ. Without such a restriction the CR-property brakes down, as otherwise we could have e.g. $M_1 \equiv \Delta x.x\ \nabla(x\ M) \gg_{\phi\nabla} \Delta x.\nabla(x\ M) \equiv M_3$ and $M_1 \gg_{\triangleright} M \equiv M_2$. This shows that we could not obtain a CR Call-by-Value calculus by simply adding a control-right rule (and imposing Call-by-Value restrictions on the reduction, in the manner of Felleisen.) One might venture to guess that such a control-right rule *could* be added under preservation of the CR property, *if* one were also to add more rules of the same kind as rule (3) and rule (4) (restrictions of the top level computation rule for $\mathcal{F}$.) One such extension is considered in the following section. □

Lemma **8** *$\gg_{\phi\Delta}$ is CR.*

Proof: It is easy to verify that $\gg_{\phi\Delta} \models \diamond$. □

The main lemma follows:

Lemma **9** *For any $M_1, M_2, M_3 \in \Lambda_\delta$: if $M_1 \gg M_2$ and $M_1 \gg_{\phi\Delta} M_3$ then there is an $M_4 \in \Lambda_\delta$ such that $M_3 \gg^* M_4$ and $M_2 \gg_{\phi\Delta} M_4$.*

Proof: Given in Appendix B. □

Now we can prove:

Lemma **10** *$\gg^*$ and $\gg^*_{\phi\Delta}$ commute.*

Proof: By Lemma 9 and Lemma 5, in analogy with the proof of Lemma 6. □

Now we can prove that adding rule (7) does not disturb the CR-property: Let $\triangleright_{1,2,3,4,7}$ denote the compatible closure of $\rightarrow_i$, $i = 1 \ldots 4$, and $\rightarrow_7$. Then we have :

Lemma **11** *$\triangleright_{1,2,3,4,7}$ is CR.*

Proof: From Lemma 7, Lemma 8 and Lemma 10 it follows by the Hindley-Rosen Lemma that $\gg \cup \gg_{\phi\Delta}$ is CR, i.e., $(\gg \cup \gg_{\phi\Delta})^* \models \diamond$. But clearly, $\triangleright^{=}_{1,2,3,4,7} \subseteq (\gg \cup \gg_{\phi\Delta}) \subseteq \triangleright^*_{1,2,3,4,7}$, and so $(\gg \cup \gg_{\phi\Delta})^* = \triangleright^*_{1,2,3,4,7}$. Consequently, $\triangleright_{1,2,3,4,7}$ is CR. □

Finally, we add the reduction $\triangleright_6$, thereby incorporating both δ-rules:

Lemma **12** *For any $M_1, M_2, M_3 \in \Lambda_\delta$: if $M_1 \rhd_{\beta(\phi\Delta)} M_2$ and $M_1 \rhd_6 M_3$ then there is an $M_4 \in \Lambda_\delta$ such that $M_3 \rhd_6^* M_4$ and $M_2 \rhd_{\beta(\phi\Delta)} M_4$.*

Proof: Easy, by inspection of the relative positions of redices for $\rhd_{1,2,3,4,7}$ and $\rhd_6$, respectively. □

Finally, we can prove

Theorem **2** *$\rhd$ is CR.*

Proof: From Lemma 12 we get, by diagram chase (or, alternatively, by Lemma 5 together with inspection of the case where $M_1 \equiv M_3$), that $\rhd_{1,2,3,4,7}^*$ and $\rhd_6$ commute. Since $\rhd_{1,2,3,4,7}$ is CR (by Lemma 11) and $\rhd_6$ is obviously also CR, it follows by the Hindley-Rosen Lemma that $\rhd = \rhd_{1,2,3,4,7} \cup \rhd_6$ is CR. □

3.3 SR and CR Properties for $\lambda_\Delta^{\perp,\supset}$

The SR property for $\lambda_\Delta^{\perp,\supset}$ allows us also to infer the CR property as an easy corollary of the CR property for λ_Δ.

Theorem **3** *If $\Gamma \vdash M : \mathcal{Q}$ and $M \rhd M'$ then $\Gamma \vdash M' : \mathcal{Q}$.*

Proof: First we show that, for any redex M such that $M \to M'$, the claim holds, and only the cases where M is a redex of rules (2) through (4) are interesting:

Suppose $\Gamma \vdash (\Delta x.M)\, N : \mathcal{Q}$. Then $\Gamma \vdash \Delta x.M : \mathcal{P} \supset \mathcal{Q}$ and $\Gamma \vdash N : \mathcal{P}$. Consequently, $\Gamma, x : \neg(\mathcal{P} \supset \mathcal{Q}) \vdash M : \perp$. Now, since $\Gamma, \kappa : \neg\mathcal{Q} \vdash \lambda f.\kappa\,(f\, N) : \neg(\mathcal{P} \supset \mathcal{Q})$ it follows that $\Gamma, \kappa : \neg\mathcal{Q} \vdash M\{x := \lambda f.\kappa\,(f\, N)\} : \perp$, and therefore $\Gamma \vdash \Delta\kappa.M\{x := \lambda f.\kappa\,(f\, N)\} : \mathcal{Q}$.

Suppose $\Gamma \vdash \Delta x.x\, M : \mathcal{Q}$ with $x \notin FV(M)$. Then $\Gamma, x : \neg\mathcal{Q} \vdash x\, M : \perp$ and consequently $\Gamma, x : \neg\mathcal{Q} \vdash M : \mathcal{Q}$. Since $x \notin FV(M)$ it follows that $\Gamma \vdash M : \mathcal{Q}$.

The case where $\Gamma \vdash \Delta x.x\, \nabla(x\, M) : \mathcal{Q}$ with $x \notin FV(M)$ is similar.

From the result above we get subject reduction by showing (easy inductions), firstly, that for any context C and term R, $\Gamma \vdash C[R] : \mathcal{P}$ implies that there is a $\mathcal{Q}$ such that $\Gamma, x : \mathcal{Q} \vdash C[x] : \mathcal{P}$ and $\Gamma \vdash R : \mathcal{Q}$; and secondly the standard substitutution lemma, i.e., if $\Gamma, x : \mathcal{Q} \vdash M : \mathcal{P}$ and $\Gamma \vdash N : \mathcal{Q}$ then $\Gamma \vdash M\{x := N\} : \mathcal{P}$. □

Now we have the CR-property for the typed calculus, *i.e.* if $\Gamma \vdash M_1 : \mathcal{P}$, $\Gamma \vdash M_2 : \mathcal{P}$, $\Gamma \vdash M_3 : \mathcal{P}$ and $M_1 \rhd^* M_2$ and $M_1 \rhd^* M_3$, then there is an M_4 such that $\Gamma \vdash M_4 : \mathcal{P}$ and $M_2 \rhd^* M_4$ and $M_3 \rhd^* M_4$.[5]

[5]This formulation is somewhat elaborate. For a slightly different formulation [Mit90] shows that the “folklore Theorem” stating that CR for the simply typed λ-calculus follows from CR for the untyped λ-calculus, is untrue.

COROLLARY 1 *$\lambda_\Delta^{\perp,\supset}$ has the CR property.*

PROOF: Follows from Theorem 1 and Theorem 3. □

3.4 SN Property for $\lambda_\Delta^{\perp,\supset}$

In this subsection we prove SN for $\lambda_\Delta^{\perp,\supset}$which, as the reader will recall, contains reduction rules (1)-(4). Below we adopt the convention of suffixing a system with a set of rules to indicate that we are studying the system with only that subset of its reduction rules, eg. $\lambda_\Delta^{\perp,\supset}$(1-2) means $\lambda_\Delta^{\perp,\supset}$with rules (1)-(2). In this section, $\vdash$ denotes type inference in $\lambda_\Delta^{\perp,\supset}$unless something is stated to the contrary.

Many SN proofs apply Prawitz' notion of validity [Pra71], Tait's notion of convertability [Tai67], Martin-löf's notion of computability [Mar70] or Girard's notion of reducibility [Gir71]. A proof of SN for $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$with among others the rules (1a)-(1e), (2a)-(2e) with slight modifications in (2c) and (2e), has been given by Stålmarck [Sta91]. This in particular yields SN for $\lambda_\Delta^{\perp,\supset}$(1-2).

Termination proofs for control operator languages similar to $\lambda_\Delta^{\perp,\supset}$have been given by Griffin [Gri90] and Murthy [Mur92a] who use CPS-translations to show termination for a particular reduction strategy [6] (and in the latter case for constructions only of a particular class of formulae.) Finally, Barbanera and Berardi [Bar92] have a very technical proof of a SN theorem for a calculus with call-by-value control operator rules but with nothing similar to our rules (3)-(4).

While all this provides us with a substantial tool-box of ideas for proving SN for $\lambda_\Delta^{\perp,\supset}$(1-4) we choose, in fact, to start all over and prove SN for $\lambda_\Delta^{\perp,\supset}$(1-4) in three stages starting from SN of $\lambda^{\perp,\supset}$(1): (*i*) SN for $\lambda_\Delta^{\perp,\supset}$(1); (*ii*) SN for $\lambda_\Delta^{\perp,\supset}$(1-2); (*iii*) SN for $\lambda_\Delta^{\perp,\supset}$(1-4). In each step the proof conveys a clear and simple intuition that the addition with which the step is concerned does not disturb the SN property of the preceding system. Having said this we should also admit that our proof takes advantage in a critical way of the simplifying facts that the calculus is call-by-name in Δ and that $\vee$ (and $\exists$) are not present in the language.

Strong Normalization for $\lambda_\Delta^{\perp,\supset}$(1)

The only difference between $\lambda^{\perp,\supset}$(1) and $\lambda_\Delta^{\perp,\supset}$(1) is that the latter system has an extra inference rule; there are no extra reduction rules. However, one might imagine that the extra inference rule allowed so many typed λ-terms that there were non-terminating sequences of β-reductions. Our job is to show that this is not the case.

[6]We deliberately avoid the phrase "SN proof" here, since SN and WN degenerates to the same notion when a particular reduction strategy is fixed.

The idea is that in $\lambda^{\bot,\supset}_{\Delta}(1)$ $\Delta x.\bullet$ is just a syntactic wrapper that can never be removed. This is similar to $y\ \bullet$ provided that nothing is ever subsituted for y. This is true *e.g.* for a construction of form $\lambda y.C[y\ \bullet]$. We exploit this idea below.

Definition **20** In the context of lists $V = [x_1, \dots x_n], W = [y_1 \dots y_n]$ of variables define

$$K^+ \equiv \lambda x_1 \dots . \lambda x_n . \lambda y_1 \dots . \lambda y_n . K$$

For $\vdash M : \mathcal{P}$, let all the variables in M which are bound by a Δ be contained in $V = [x_1 \dots x_n]$ (of types $\mathcal{P}_1 \dots \mathcal{P}_n$) and let $W = [y_1 \dots y_n]$ be variables not occurring in M (of types $\bot \supset \mathcal{P}_1 \dots \bot \supset \mathcal{P}_n$.) In the context of these two lists define $\underline{\bullet}$ as follows:

$$\begin{array}{lcl} \underline{x} & \equiv & x \\ \underline{\lambda x.M} & \equiv & \lambda x.\underline{M} \\ \underline{M\ N} & \equiv & \underline{M}\ \underline{N} \\ \underline{\Delta x_i.M} & \equiv & y_i\ \underline{M} \end{array}$$

□

Lemma **13** *Assume $\vdash M : \mathcal{P}$. Let all the variables in M which are bound by a Δ be contained in $V = [x_1 \dots x_n]$ (of types $\mathcal{P}_1 \dots \mathcal{P}_n$) and let $W = [y_1 \dots y_n]$ be variables not occurring in M (of types $\bot \supset \mathcal{P}_1 \dots \bot \supset \mathcal{P}_n$.) Then*

1. *$\vdash (\underline{M})^+ : \mathcal{P}_1 \supset \dots \mathcal{P}_n \supset (\bot \supset \mathcal{P}_1) \supset \dots (\bot \supset \mathcal{P}_n) \supset \mathcal{P}$ in $\lambda^{\bot,\supset}$.*
2. *If $M \rhd_1 N$ then $(\underline{M})^+ \rhd_1 (\underline{N})^+$.*

Proof: *Ad 1:* Let $\Gamma \equiv \{x_1 : \mathcal{P}_1, \dots, x_n : \mathcal{P}_n, y_1 : \bot \supset \mathcal{P}_1 \dots y_n : \bot \supset \mathcal{P}_n\}$ and prove by induction on M that $\Gamma \vdash \underline{M} : \mathcal{P}$ in $\lambda^{\bot,\supset}$. Then use ($\supset I$) $2n$ times.

Ad 2: Extend $\underline{\bullet}$ to contexts by: $\underline{[]} \equiv []$ and $\underline{C}$ on the same induction as $\underline{M}$ when $C \not\equiv []$. More precisely:

$$\begin{array}{lcl} \underline{[]} & \equiv & [] \\ \underline{\lambda x.C} & \equiv & \lambda x.\underline{C} \\ \underline{C\ N} & \equiv & \underline{C}\ \underline{N} \\ \underline{M\ C} & \equiv & \underline{M}\ \underline{C} \\ \underline{\Delta x_i.C} & \equiv & y_i\ \underline{C} \end{array}$$

We call this the *context extension* of the original translation on constructions. By induction on C it is easily established that $\underline{C}$ is again a general contex and $\underline{C[K]} \equiv \underline{C}[\underline{K}]$.

Now assume $M \rhd_1 N$, ie. $M \equiv C[(\lambda z.J)\ I], N \equiv C[J\{z := I\}]$. Since $\rhd_1$ is compatible it suffices to show that $\underline{M} \rhd_1 \underline{N}$, rather than $(\underline{M})^+ \rhd_1 (\underline{N})^+$. We have $\underline{M} \equiv \underline{C}[(\lambda z.\underline{I})\ \underline{J}]) \rhd_1 \underline{C}[\underline{J}\{z := \underline{I}\}])$ and $\underline{N} \equiv \underline{C}[\underline{J\{z := I\}}]$. So it suffices to show that $\underline{J}\{z := \underline{I}\} \equiv \underline{J\{z := I\}}$. This is easily established by induction J.
□

Corollary **2** *If $\vdash M : \mathcal{P}$ then there is no infinite reduction sequence in $\lambda_{\Delta}^{\perp,\supset}$(1) starting from M.*

Proof: If there were, then the preceding lemma gives a translation into an infinite reduction sequence in $\lambda^{\perp,\supset}(1)$ contradicting SN of $\lambda^{\perp,\supset}$. $\Box$

Note how $y_i\ M$ simulates $\Delta x_i.M$. None of the two ever gets reduced. We conclude that the typing rule $\perp_c$ adds nothing essential (with respect to SN) to what kind of λ-terms we can type.

Strong Normalization for $\lambda_{\Delta}^{\perp,\supset}$(1-2)

The difference between $\lambda_{\Delta}^{\perp,\supset}$(1-2) and $\lambda_{\Delta}^{\perp,\supset}$(1) is that the former can reduce certain applications of Δ-abstractions.

The idea for proving SN (inspired by Prawitz' WN proof for a system like $\lambda_{\Delta}^{\perp,\supset,\forall}$in [Pra65]) is that we can repeatedly unfold all Δx's, with x having non-atomic type, before anything else. The resulting term can simulate (2) reductions with (1) reductions. A Δ can never again end up in a redex. This would require the type of the variable that the Δ binds to have non-atomic type, but all the types of Δx's were reduced to atomic types in the beginning, and this property is preserved under reduction.

Let the size of a type be the number of occurrences of connectives and quantifiers in the type. The following definition of $\underline{M}$ should be understood to be on induction on lexicographically ordered triples (m, n, k) where m is the largest type of a variable bound by a Δ in M, n is the number of variables in M with a type of size m and bound by a Δ, k is the size of M.

Definition **21** Let $\vdash M : \mathcal{P}$. Define $\underline{\bullet}$ as follows:

$$\begin{array}{lcl}
\underline{x} & \equiv & x \\
\underline{\lambda x.M} & \equiv & \lambda x.\underline{M} \\
\underline{M\ N} & \equiv & \underline{M}\ \underline{N} \\
\underline{\Delta x^{P}.M} & \equiv & \Delta x.\underline{M} \\
\underline{\Delta x^{\neg\mathcal{P}\supset\mathcal{Q}}.M} & \equiv & \lambda a^{\mathcal{P}}.\underline{\Delta z^{\neg\mathcal{Q}}.M\{x := \lambda y.z\ (y\ a)\}}
\end{array}$$

where in the last clause $a^{\mathcal{P}}$ is a fresh variable. $\Box$

Lemma **14** *Assume $\vdash M : \mathcal{P}$. Then*

1. $\vdash \underline{M} : \mathcal{P}$.
2. *If $M \rhd_{12} N$ then $\underline{M} \rhd_1 \underline{N}$.*

Proof: *Ad 1:* Induction on M.

Ad 2: Similar to the proof of lemma 13. Let again $\underline{\bullet}$ denote the context extension also, and assume $M \rhd_{12} N$ ie. (i) $M \equiv C[(\lambda x.J)\ I], N \equiv C[J\{x := I\}]$, or (ii) $M \equiv C[(\Delta x.J)\ I], N \equiv C[\Delta z.J\{x := \lambda y.z\ (y\ I)\}]$.

In case (i) we have $\underline{M} \equiv \underline{C}[(\lambda z.\underline{I})\ \underline{J}] \rhd_1 \underline{C}[\underline{J}\{z := \underline{I}\}]$) and $\underline{N} \equiv \underline{C}[\underline{J\{z := I\}}]$. In case (ii), $\underline{M} \equiv \underline{C}[\underline{(\lambda a.\Delta z.J\{x := \lambda y.z\ (y\ a)\})}\ \underline{I}] \rhd_1 \underline{C}[\underline{\Delta z.J\{x := \lambda y.z\ (y\ \underline{I})\}}]$ and $\underline{N} \equiv \underline{C}[\underline{\Delta z.J\{x := \lambda y.z\ (y\ I)\}}]$.

So it suffices to show that $\underline{J}\{z := \underline{I}\} \equiv \underline{J\{z := I\}}$ and $\underline{\Delta z.J\{x := \lambda y.z\ (y\ \underline{I})\}} \equiv \underline{\Delta z.J\{x := \lambda y.z\ (y\ I)\}}$. The first follows by induction on the induction triple mentioned above. The second follows from idempotency of $\underline{\bullet}$. □

COROLLARY **3** *If $\vdash M : \mathcal{P}$ then there is no infinite reduction sequence starting from M in $\lambda_{\Delta}^{\perp,\supset}$ (1-2).*

PROOF: If there were, then the preceding lemma gives a translation into an infinite reduction sequence in $\lambda_{\Delta}^{\perp,\supset}(1)$ contradicting SN of $\lambda_{\Delta}^{\perp,\supset}(1)$. □

Strong Normalization for $\lambda_{\Delta}^{\perp,\supset}$(1-4)

The idea in the proof of SN for all four rules is that we can postpone applications of rule (3-4) as long as we desire, thereby obtaining longer and longer sequences of reductions using only (1-2).

LEMMA **15** *(3,4 Postponement.) If $M_1 \rhd_{3,4} M_2 \rhd_{1,2} M_3$ then there exists an M_4 such that $M_1 \rhd_{1,2}^{+} M_4 \rhd_{3,4}^{*} M_3$.*

PROOF: Given in the appendix. □

COROLLARY **4** *Let $M_1 \rhd M_2 \rhd \ldots$ be an infinite reduction sequence of $\lambda_{\Delta}^{\perp,\supset}$(1-4). Then, for any number n, there is an infinite reduction sequence $M_1 \equiv N_1 \rhd_{1,2} N_2 \rhd_{1,2} \ldots \rhd_{1,2} N_m \rhd \ldots$ with $m \geq n$, using in the first $m-1$ reduction steps only $\rhd_{1,2}$.*

PROOF: By induction on n, and $n = 0$ is trivial. Now suppose we have already constructed an infinite reduction sequence $M_1 \equiv N_1 \rhd_{1,2} \ldots N_k \rhd \ldots L_1 \rhd L_2 \ldots$ using only $\rhd_{1,2}$ in the first $k-1$ steps, with $k \geq n$. Since every reduction by $\rhd_{3,4}$ strictly decreases the size of the term, there can be no infinite sequence of $\rhd_{3,4}$-reductions. Let, accordingly, i be the smallest number such that $L_i \rhd_{1,2} L_{i+1}$. Now i applications of the preceding lemma yields a sequence $M_1 \equiv N_1 \rhd_{1,2} \ldots N_{k'} \rhd \ldots$ using only $\rhd_{1,2}$ in the first $k'-1$ steps with $k' \geq n+1$. □

Now we can prove the main result:

Theorem 4 $\lambda_{\Delta}^{\perp,\supset}$ *has the strong normalization property.*

Proof: Assume that $\lambda_{\Delta}^{\perp,\supset}$ did not have the property. Then there would be an infinite sequence $M_1 \rhd M_2 \rhd \ldots$ starting from M_1. By the preceding corollary there would then be arbitrarily long reduction sequences which use only $\rhd_{1,2}$ and starting from M_1. By König's Lemma there would then be an infinite reduction sequence using only $\rhd_{1,2}$. But this contradicts the fact that reduction is strongly normalizing in $\lambda_{\Delta}^{\perp,\supset}(1\text{-}2)$. $\square$

4 λ_Δ and $\lambda_\Delta^{\perp,\supset}$ as Control Calculi

This section is a preliminary discussion of the expressive power of λ_Δ (or $\lambda_{\Delta\delta}$.) It is largely informal and based on consideration of examples.

The operator Δ has very close connections to Felleisen's control operators. To see this, consider Felleisen's operator $\mathcal{F}$. By a factorization of its operational semantics corresponding to the one given for the operator $\mathcal{C}$ in [Fel87b] (see also [Fel89]) we can express the behaviour of $\mathcal{F}$ by means of a so-called *computation rule* $(\mathcal{F}_T)$, only applicable at the top level (i.e. in the empty context) together with two local (compatible) *reduction rules*, $(\mathcal{F}_L)$ and $(\mathcal{F}_R)$:

$$\begin{array}{llll}
(\mathcal{F}_T) & \mathcal{F}(M) & \rightarrow & M\,(\lambda x.x) \\
(\mathcal{F}_L) & \mathcal{F}(M)\,N & \rightarrow & \mathcal{F}(\lambda\kappa.M(\lambda f.\kappa\,(f\,N))) \\
(\mathcal{F}_R) & M\,\mathcal{F}(N) & \rightarrow & \mathcal{F}(\lambda\kappa.N(\lambda f.\kappa\,(M\,f)))\;, \quad \text{provided } M \text{ is a value}
\end{array}$$

These rules (and analogous ones for the $\mathcal{C}$-operator [7]) are considered together with Call by Value β-reduction by Felleisen.

Now, at the heart of the operational behaviour of the Δ-operator is rule (2) of λ_Δ. By the translation $\underline{\bullet}$ such that $\underline{\Delta x.M} \equiv \mathcal{F}(\lambda x.\underline{M})$, and such that $\underline{\bullet}$ propagates to subterms of M when M is not a Δ-abstraction, we see that

$$\begin{array}{rcl}
\underline{(\Delta x.M)\,N} & \equiv & \mathcal{F}(\lambda x.\underline{M})\,\underline{N} \\
 & \rhd_{\mathcal{F}_L} & \mathcal{F}(\lambda\kappa.(\lambda x.\underline{M})(\lambda f.\kappa\,(f\,\underline{N}))) \\
 & \rhd_\beta & \mathcal{F}(\lambda\kappa.\underline{M}\{x := \lambda f.\kappa\,(f\,\underline{N})\}) \\
 & \equiv & \underline{\Delta\kappa.M\{x := \lambda f.\kappa\,(f\,N)\}}
\end{array}$$

So the second rule of λ_Δ is exactly Felleisen's $(\mathcal{F}_L)$-rule restricted to abstractions and optimized by an on-line β-reduction. If we exchanged our rule (2) with the rule

$$\Delta(\lambda x.M)\,N \rhd \Delta(\lambda\kappa.M\{x := \lambda f.\kappa\,(f\,N)\})$$

so that Δ would not be a variable binding operator, we would obtain a slightly more general system in which we could reduce an expression $\Delta(P)\,N$ where P is any expression (not necessarily an abstraction) which reduces to an abstraction. Using the form $\Delta x.M$ effectively turns $\mathcal{F}$ into a variable binding operator and corresponds to syntactically excluding the possibility of forming expressions $\mathcal{F}(P)$ where P is not an abstraction. This restriction is, however, not significant. It is easy to see that both of the fundamental properties of λ_Δ and $\lambda_\Delta^{\perp,\supset}$ (CR for both and SN for the latter) hold for the slightly more general calculus obtained by translating the Δ-rules as indicated above. Moreover, the difference is of no significance w.r.t. the kind of programming our calculus is intended to model.

[7] $\mathcal{C}$ can be defined in terms of $\mathcal{F}$ by $\mathcal{C}(M) \equiv \mathcal{F}(\lambda\kappa.M(\lambda w.\mathcal{A}(\kappa\,w)))$ where $\mathcal{A}$ is given by $\mathcal{A}(N) \equiv \mathcal{F}(\lambda d.N)$, d a dummy variable.

As for rules (3) and (4) of λ_Δ, we see by the same translation that these rules are restricted forms of the computation rule ($\mathcal{F}_T$) for $\mathcal{F}$, since we have (for $x \notin FV(M)$)

$$\begin{array}{rcl} \underline{\Delta x.x\, M} & \equiv & \mathcal{F}(\lambda x.x\, \underline{M}) \\ & \rhd_{\mathcal{F}_T} & (\lambda x.x\, \underline{M})(\lambda y.y) \\ & \rhd^*_\beta & \underline{M} \end{array}$$

and

$$\begin{array}{rcl} \underline{\Delta x.x\, \nabla(x\, M)} & \equiv & \mathcal{F}(\lambda x.x\, \mathcal{A}(x\, \underline{M})) \\ & \rhd_{\mathcal{F}_T} & (\lambda x.x\, \mathcal{A}(x\, \underline{M}))(\lambda y.y) \\ & \rhd^*_\beta & \mathcal{A}(\underline{M}) \\ & \rhd_{\mathcal{A}} & \underline{M} \end{array}$$

A crucial difference between $\mathcal{F}_T$ and rules (3) and (4) is, of course, that the latter two rules are completely compatible (i.e. not restricted to top level.)

The λ_Δ-calculus can express the `catch/throw`-mechanism with operational semantics given by a rule of the form

$$\texttt{catch}\, x\, E[\texttt{throw}\, x\, M] \quad \rhd \quad M$$

where E ranges over a specified notion of evaluation contexts in which the rule above is parametric.

To tie in with Felleisen's operators, consider that the `catch/throw`-mechanism can be defined in terms of $\mathcal{F}$. Below are shown an implementation using $\mathcal{F}$ and one using $\mathcal{C}$. [8]:

$$\begin{array}{ll|ll} E_1[\mathcal{F}(\lambda x.E_2[\mathcal{A}(x\, M)])] & \rhd & E_1[\mathcal{C}(\lambda x.E_2[x\, M])] & \rhd \\ (\lambda x.E_2[\mathcal{A}(x\, M)])(\lambda z.E_1[z]) & \rhd^* & (\lambda x.E_2[x\, M])(\lambda z.\mathcal{A}(E_1[z])) & \rhd^* \\ E_2[\mathcal{A}(E_1[M])] & \rhd & E_2[\mathcal{A}(E_1[M])] & \rhd \\ E_1[M] & & E_1[M] & \end{array}$$

In the implementation to the left, the form `catch` $x\ N$ is expressed by $\mathcal{F}(\lambda x.N)$ and the form `throw` $x\ L$ is expressed by $\mathcal{A}(x\ L)$. To the right, the `catch` has the same kind of representation whereas the `throw` is so to speak implicit in the application $x\ M$.

In analogy with the impementations above, we can write λ_Δ-expressions in which the form $\Delta x.x\ M$ can be read as `catch` $x\ M$, and the form $\nabla(x\ N)$ can be read as `throw` $x\ N$.

As already stated (and as is clear from the correspondence between Δ and $\mathcal{F}$) the second rule of λ_Δ is the core of the operational meaning of Δ. The other two Δ-rules can typically be seen as "clean-up" operations: From a `catch/throw`-perspective the third rule allows the elimination of empty `catch`'es, thus coping with the situation of unexceptional return. The fourth rule copes with exeptional return, eliminating the `catch/throw`-pair once they have met, after the ∇ has

[8] Given the operational derivability of $\mathcal{C}$ from $\mathcal{F}$ the latter can be regarded as an alternative way of doing it by means of $\mathcal{F}$

"bubbled" out to the Δ by the second rule. The two rules together allow us to write expressions of the form $\Delta k.k\,M$, where M can either return normally with, say, some value V, or M can return exceptionally with, say, $\nabla(k\,V)$. In the case of normal return the leftmost k-application ensures type-correctness, *i.e.* , k of type $\neg\mathcal{P}$ ensures that, whether we return normally or exceptionally, the object returned to the continuation point labelled by the Δ-abstraction will be of type $\mathcal{P}$.

These functions are illustrated by examples below.

The following (perhaps not very useful) function F captures several aspects of how Δ expresses a form of `catch`/`throw`-programming.

$$F \equiv \lambda n.\Delta j.j\,(+\;1\;\Delta k.k\,(+\;1\,(\texttt{if}\;(=\;n\,0)\;\nabla(j\;0)\;(\texttt{if}\;(=\;\texttt{n}\;\texttt{1})\;\nabla(k\;1)\;1))))$$

Reading `catch` and `throw` into the expressions as indicated above, we get the expected behaviour:

EXAMPLE **1**

$$\begin{array}{rcl} F\,2 &=& \Delta j.j\,(+_1\,\Delta k.k\,(+_1\,1)) \\ &=& \Delta j.j\,(+_1\,2) \\ &=& 3 \\ \\ F\,1 &=& \Delta j.j\,(+_1\,\Delta k.k\,(+_1\,\nabla(k\,1))) \\ &=& \Delta j.j\,(+_1\,\Delta k.k\,\nabla(k\,1)) \\ &=& \Delta j.j\,(+_1\,1) \\ &=& 2 \\ \\ F\,0 &=& \Delta j.j\,(+_1\,\Delta k.k\,(+_1\,\nabla(j\,0))) \\ &=& \Delta j.j\,(+_1\,\Delta k.k\,\nabla(j\,0)) \\ &=& \Delta j.j\,\Delta\kappa.(k\,\nabla(j\,0))\{k := \lambda f.\kappa\,(+_1\,f)\} \\ &=& \Delta j.j\,\Delta\kappa.(\lambda f.\kappa\,(+_1\,f))\,\nabla(j\,0) \\ &=& \Delta j.j\,\Delta\kappa.\kappa\,(+_1\,\nabla(j\,0)) \\ &=& \Delta j.j\,\Delta\kappa.\kappa\,\nabla(j\,0) \\ &=& \Delta j.j\,\nabla(j\,0) \\ &=& 0 \end{array}$$

□

Assuming a typed variant of $\lambda_{\Delta\delta}$including base types, such as **Int** and **Bool**, we have

$$\lambda n.^{\mathbf{Int}}\Delta j^{\neg\mathbf{Int}}.j\,(+\;1\,\Delta k^{\neg\mathbf{Int}}.k\,(+\;1\,(\texttt{if}\;(=\;n\,0)\;\nabla(j\,0)\;(\texttt{if}\;(=\;\texttt{n}\;\texttt{1})\;\nabla(k\;1)\;1))))$$

of type **Int** $\rightarrow$ **Int**, using the intuitionistic rule to get the typings $\nabla(j\,0)$: **Int** and $\nabla(k\;1)$: **Int** for those subexpressions. Compare the second last line of the trace of $F\,2$ (normal return) with the second last line of the trace of $F\,0$ (exceptional return).

Finally, we shall consider the by now classical problem of writing a function `M` which takes a binary tree of integer nodes and returns the result of multiplying all the node values. The idea is to stop multiplying as soon as a node value of 0 is encountered, by `throw`'ing 0 to the top level. Suppose we have a representation of binary trees `nil`, `<root,leftson,rightson>` together with a test function for `nil`, `mt?`, a test for zero on integers, `zero?`, and selectors `num`, `lson`, `rson` selecting respectively the node value of the root node, the left subtree and the right subtree. Let $\mathbf{Y}$ denote Church's fixpoint combinator.

EXAMPLE **2** (tree multiplier)

```
M = λt.Δ j.j
        (Y (λf.λt'. (if (mt? t')
                   1
                   (if (zero? (num t'))
                         ∇(j 0)
                         (* (num t')
                         (* (f (lson t'))(f (rson t')))))))) t)  □
```

To see how this works we trace a simple execution. Let $\mathtt{T} \equiv$ `<2,<0,nil,nil>,nil>`. Let F_j denote the subexpression

```
(λf.λt'. (if (mt? t')
             1
             (if (zero? (num t'))
                   ∇(j 0)
                   (* (num t')
                   (* (f (lson t'))(f (rson t')))))))
```

Then we have

$$\begin{aligned}
\mathtt{MT} &= \Delta j.j\,((\mathbf{Y}F_j)\,\mathtt{T})\\
&= \Delta j.j\,((F_j\,(\mathbf{Y}\,F_j))\,\mathtt{T})\\
&= \Delta j.j\,(*\,2\,(*\,((\mathbf{Y}\,F_j) < \mathtt{0},\mathtt{nil},\mathtt{nil} >)\,((\mathbf{Y}\,F_j)\,\mathtt{nil})))\\
&= \Delta j.j\,(*_2\,(*\,(F_j\,(\mathbf{Y}\,F_j) < \mathtt{0},\mathtt{nil},\mathtt{nil} >)\,((\mathbf{Y}\,F_j)\,\mathtt{nil})))\\
&= \Delta j.j\,(*_2\,(*\,\nabla(j\;0)\,((\mathbf{Y}\,F_j)\,\mathtt{nil})))\\
&= \Delta j.j\,(*_2\,(\nabla(j\;0)\,((\mathbf{Y}\,F_j)\,\mathtt{nil})))\\
&= \Delta j.j\,(*_2\,\nabla(j\;0))\\
&= \Delta j.j\,\nabla(j\;0)\\
&= 0
\end{aligned}$$

where we used that $F_j\,(\mathbf{Y}\,F_j) < \mathtt{0},\mathtt{nil},\mathtt{nil} > = \nabla(j\;0)$ at the fifth equality.

Since the `catch/throw`-mechanism consists only in the potential to discard a continuation, rules (3), (4) together with the derived ∇-rules are sufficient for the `catch/throw`-mechanism. Since this calculus is interesting in its own right, let us emphasize it by formally defining the subcalculus $\lambda_{\mathbf{ct}}$ generated by the following notion of reduction:

DEFINITION **22** (`catch`/`throw`-subcalculus $\lambda_{\mathbf{ct}}$)

$$\begin{array}{lll}
(ct1) & (\lambda x.M)\,N & \rightarrow\ M\{x := N\} \\
(ct2) & \nabla(M)\,N & \rightarrow\ \nabla(M) \\
(ct3) & \phi\,\nabla(N) & \rightarrow\ \nabla(N) \\
(ct4) & \phi\,b & \rightarrow\ \delta(\phi, b) \\
(ct5) & \Delta x.x\,M & \rightarrow\ M\ ,\ \text{provided } x \notin FV(M) \\
(ct6) & \Delta x.x\,\nabla(x\,M) & \rightarrow\ M\ ,\ \text{provided } x \notin FV(M)
\end{array}$$

□

THEOREM **5** *The reduction relation of* $\lambda_{\mathbf{ct}}$ *is* CR.

PROOF: By Lemma 2 we see that rules ($ct1$), ($ct2$), ($ct5$) and ($ct6$) constitute a Church-Rosser subcalculus. Now, rule ($ct3$) is easily seen to be strongly normalizing and locally confluent, and the rule commutes (analogously to Lemma 12) with the subcalculus just mentioned. Therefore rule ($ct3$) can be added by the Hindley-Rosen Lemma. Also, rule ($ct4$) can be added, analogously to Lemma 12. This accounts for the CR property. □

As an example of a control construct which goes beyond this restricted scheme consider the `call/cc`-construct. Using Felleisen's $\mathcal{A}$ we can express its meaning as:

$$E[\texttt{call/cc}(\lambda k.M)] \ \ \triangleright\ \ E[M\{k := \lambda z.\mathcal{A}(E[z])\}]$$

As is well known, we could use $\mathcal{C}$ to implement this, writing $\mathcal{C}(\lambda k.k\ M)$ for `call/cc`$(\lambda k.M)$. Now, we can do something *similar* with Δ, writing $\Delta k.(kM\{k := \lambda w.\nabla(k\,w)\})$ for `call/cc`$(\lambda k.M)$. To see how this works, consider the program

$$(+\ 2\ \texttt{call/cc}(\lambda k.(*\ 5\ (k\ 4))))$$

which we expect to evaluate to `6`. And in fact we have

$$\begin{array}{ll}
(+2\Delta k.(k\ (*\ 5\ (k\ 4))\{k := \lambda w.\nabla(k\ w)\})) & \equiv \\
(+\ 2\ \Delta k.(k\ (*\ 5((\lambda w.\nabla(k\ w))\ 4)))) & = \\
(+_2\ \Delta k.(k\ (*_5\ ((\lambda w.\nabla(k\ w))\ 4)))) & = \\
(+_2\ \Delta k.(k\ (*_5\ \nabla(k\ 4)))) & = \\
\Delta\kappa.(k\ (*_5\ \nabla(k\ 4)))\{k := \lambda f.\kappa\ (+_2\ f)\} & \equiv \\
\Delta\kappa.(\lambda f.\kappa\ (+_2\ f))\ (*_5\ \nabla((\lambda f.\kappa\ (+_2\ f))\ 4)) & = \\
\Delta\kappa.(\lambda f.\kappa\ (+_2\ f))\ (*_5\ \nabla(\kappa\ (+_2\ 4))) & = \\
\Delta\kappa.(\lambda f.\kappa\ (+_2\ f))\ \nabla(\kappa\ (+_2\ 4)) & = \\
\Delta\kappa.\kappa\ (+_2\ \nabla(\kappa\ (+_2\ 4))) & = \\
\Delta\kappa.\kappa\ \nabla(\kappa\ (+_2\ 4)) & = \\
(+_2\ 4) & = \\
6 &
\end{array}$$

As an example of normal return we consider the program

$$(+\,2\;\texttt{call/cc}(\lambda k.(*\,5\,4)))$$

This is turned into

$$\begin{array}{ll}(+\,2\,\Delta k.(k\,(*\,5\,4)\{k := \lambda w.\nabla(k\,w)\}) & \equiv\\ (+\,2\,\Delta k.(k\,(*5\,4))) & =\\ (+\,2\,\Delta k.k\,20) & =\\ (+\,2\,20) & =\\ 22 & \end{array}$$

This scheme is obviously not perfect, since it cannot *express* the full `call/cc`-mechanism. It only works in the examples above because they make restricted use of the `call/cc` construct. In fact, the uses of `call/cc` in those examples are exactly expressible by the `catch/throw`-mechanism, using `catch` k $M\{k := \lambda w.$`throw` k $w\}$ for `call/cc`$(\lambda k.M)$.

By expressibility we have in mind here an informal notion of *syntactically local* translatability ("syntactic sugaring" in a strong, local sense), such that "a facility is expressible if every usage instance is replacable by a behaviourally equivalent instantiation of an expression schema". [9] For instance, we know that control mechanisms can be *simulated* in the pure λ-calculus by CPS translation. This does *not* mean that the pure λ-calculus can *express* these mechanisms, since the CPS translation is inherently *global*, restructuring the layout of the entire program. Expressibility is a much closer relationship where the expressed construct should be *locally* eliminable by the expressing construct, independently of context.

Now, `call/cc` is an essentially *context sensitive* or *non-transparent* operator, in contradistinction to the `catch/throw`-mechanism. Consider as a simple example the expression $F \equiv$ `call/cc`$(\lambda k.k)$; since this expression evaluates to $E[\lambda z.\mathcal{A}(E[z])]$ where E is the context in which the expression is evaluated, there exists no value V such that F is exchangible with V in all contexts. Therefore, we cannot expect any transparent construct to express it. Since our calculus is fully compatible (and therefore transparent) we cannot hope to express the full `call/cc` construct in a strong sense (to repeat : it is another question whether we might be able to *simulate* it by some kind of translation; to be sure, such a translation could not be local.)

It is characteristic that all the example programs above could in fact be reduced by means of the $\lambda_{\mathbf{ct}}$-rules only. This raises the question whether λ_Δ is a strict extension of $\lambda_{\mathbf{ct}}$ or, in other words, whether the full "bubble-rules" (2) and (7) are superfluous in the presence of just the special cases, (*ct*2) and (*ct*3).

[9] Citation from [Fel90], referring to the common informal notion of expressibility. As shown by Felleisen, this informal notion can be made precise. Also, the present discussion should be carried out using a technical notion of expressibility, such as Felleisen's. For lack of time this has not been done yet.

In order to get a fully compatible calculus, we had to impose a restriction on the computation rule for $\mathcal{F}$ strong enough to ensure that the Church-Rosser property is preserved when the restricted computation rule is turned into a compatible rule. But this means that only very specific modes of "return" within a Δ-abstraction allow the abstraction to be eliminated and reduce to a value. As can be seen from the Δ-elimination rules, (3) and (4), only `catch`/`throw`-like modes of return are allowed.

However, although this is so, the rules (2) and (7) are, in a technical sense, not redundant, since we can find expressions that reduce to a value in λ_Δ but not in $\lambda_{\mathbf{ct}}$. Consider $G \equiv (\Delta k.\ k\ \lambda t.t\ k)\ (\lambda x.\nabla(x\ \lambda d.V))$. The left hand side of this application cannot be reduced in $\lambda_{\mathbf{ct}}$, so G is in normal form w.r.t. that calculus (in so far as V is.) But in λ_Δ we can reduce $G \rhd^* \Delta\kappa.\kappa\ \nabla(\kappa\ V) \rhd V$, using rule (2). G exemplifies that an abstracted `throw` can be captured by a Δ-abstraction, `catch`'ing the `throw` once it is executed inside the scope of the Δ.

It remains to be seen whether there are *interesting* uses of the surplus power of λ_Δ over $\lambda_{\mathbf{ct}}$ from a programming point of view. All we can do here is pose the question: *Are there computationally interesting, strict transparent generalizations of the* `catch`/`throw`*-mechanism ?* Here it is a possibility that λ_Δ is not powerful enough. Unfortunately, we have not yet found the time to answer these questions, but we propose for further investigation to study λ_Δ as a platform for seeking transparent generalizations of the `catch`/`throw`-mechanism (or $\lambda_{\mathbf{ct}}$.) In view of what was said above, a natural way to go about this would be to add more Δ-elimination rules. And such rules, in turn, could naturally be sought as new restrictions of the $\mathcal{F}$-computation rule. For this idea to be interesting we must at the very least make sure that λ_Δ is *incomplete* in the sense that new reductions can be added under preservation of the Churh-Rosser property. [10]

To show that λ_Δ is in fact incomplete, consider the notion of reduction

$$(8)\quad \Delta x.\nabla(M) \quad\rightarrow\quad \Delta x.M$$

Let $\unrhd$ be the one step reduction induced by the union of all the $\rightarrow_i$, $i = 1 \ldots 8$. Clearly, $\rhd$ is strictly contained in $\unrhd$.

THEOREM **6** *$\unrhd$ is CR. In particular, λ_Δ is incomplete.*

PROOF: Given in Appendix D □

Rule (8) is computationally interesting in its own right. As a special case of it we have $\nabla(\nabla(M)) \unrhd \nabla(M)$, expressing idempotency of the local abort operator ∇. Another property of this rule is the following. Remembering that $\mathcal{C}$ is definable in terms of $\mathcal{F}$, it is natural to ask whether the operator Δ', given

[10]The notion of completeness here is related to the notion of Hilbert-Post compleness of equational theories, see [Bar84].

by having the same relationship to $\mathcal{C}$ as Δ has to $\mathcal{F}$, is definable in terms of Δ. One can verify that this is in fact not so in λ_Δ, but it is the case when Δ is extended with rule (8) and Δ' is defined accordingly.

5 Classical proof theory

The preceding section showed that the reduction rules for Δ (*i*) are closely related to Fellisen's $\mathcal{F}$-operator and (*ii*) can express the catch-throw paradigm. In this section we show that the reduction rules for Δ (*i*) are closely related to classical proof normalization rules as found in the proof theory literature, and (*ii*) can be used to develop classical proof theory. This shows, indirectly, that there is a close connection between Felleisen's control operators and classical proof normalization rules.

Subsection 1 describes the inversion principle in minimal and classical logics. Subsection 2 briefly reviews the standard classical proof normalization procedures and compares them to the reduction rules in the classically typed λ_Δ-calculi. Subsection 3 develops proof theory for $\lambda_\Delta^{\perp,\supset}$ and its classical relatives in the style of [Pra65] ch. III.

5.1 The inversion principle

Recall that derivations are certain trees, *e.g.*

$$\dfrac{\dfrac{[x:P]}{\lambda x.x : P\supset P} \quad \dfrac{[y:P]}{\lambda y.y : P\supset P}}{<\lambda x.x, \lambda y.y>: P\supset P\wedge P\supset P}$$

A derivation with no open assumptions will henceforth be called a *proof.*[11] A proof is represented by the construction at the root of the proof. To be precise, there corresponds to every proof one construction, viz. the one at the root of the proof, but the same construction may correspond to different proofs, *e.g.* $\lambda x.x$ corresponds to the simplest proof of both $P\supset P$ and $(P\supset P)\supset(P\supset P)$. This is a consequence of the fact that we are considering types à là Curry as opposed to types à là Church, see [Bar91]. (In a sense, the construction represents the *structure* of the proof rather than the proof itself.) We take the notion of a formula *occurrence* in a derivation (*e.g.* $P\supset P$ at some position in a derivation) and the notion of one formula occurrence standing immediately below or above another for granted.

For convenience the discussion below is carried out in the systems $\lambda^{\perp,\supset,\wedge,\vee}$ and $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$, but it applies to other minimal and classical logics as well. In this subsection we would like the reader to temporarily forget that we have already defined certain reductions on constructions in $\lambda^{\perp,\supset,\wedge,\vee}$ and $\lambda_\Delta^{\perp,\supset,\wedge,\vee}$, although the notion of a redex will still be used.

In $\lambda^{\perp,\supset,\wedge,\vee}$ the inference rules can be divided into I rules and E rules. The I rule for a connective γ provides a sufficient condition for an inference *to* a formula with γ as its principal connective. The E rule for γ allows an inference

[11]We also occassionally use "proof" to mean a proof in the metareasoning. No other meanings will be assigned to "proof."

from a formula with γ as its principal connective. In this sense E rules and I rules are each others inverses. Prawitz' *Inversion Principle* [Pra65] ch. II (originating from Gentzen) states that in a proof of $\mathcal{P}$ which applies an I rule immediately follwed by an application of the corresponding E rule, the proof of $\mathcal{P}$ is already contained (in some sense) in the fragment of the inference which ends with the application of the I rule. For instance, in a proof which infers $\mathcal{P}_1 \wedge \mathcal{P}_2$ by $I\wedge$ and then $\mathcal{P}_1$ by $\wedge E$ the proof of $\mathcal{P}_1 \wedge \mathcal{P}_2$ already contains a proof of $\mathcal{P}_1$, because $I\wedge$ requires a proof of $\mathcal{P}_1$ and a proof of $\mathcal{P}_2$ to conclude $\mathcal{P}_1 \wedge \mathcal{P}_2$.

A formula occurrence in a derivation in $\lambda^{\perp,\supset,\wedge,\vee}$which is both the conclusion of an I rule and the major premise of an E rule is called a maximum formula, and in analogy with the terminology for constructions let us call a derivation with no maximum formulae normal.

The inversion principle suggests the *Inversion Theorem* which states that for every derivation there is a normal derivation. The Inversion Theorem is proved by providing a set of reduction rules from derivations to derivations which remove maximum formulae such that for every derivation there is a finite sequence of reductions ending in a normal derivation.

Now, every introduction rule is represented by a constructor construction, and every elimination rule is represented by a destructor construction. A maximum formula in a derivation is therefore the type of a certain part of a redex. In fact, there is a one-one correspondence between maximum formulae in a derivation and redexes in the construction representing the derivation. In terms of constructions the Inversion Theorem states that for every construction of type $\mathcal{P}$ there is another construction of type $\mathcal{P}$ with no redexes, and the proof proceeds by defining a set of reduction rules which eliminate redexes and which have the WN property.

As Prawitz notes [Pra65] p35 the Inversion Principle does not have the same appealing character for $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee}$as for $\lambda^{\perp,\supset,\wedge,\vee}$. Specifically, the symmetry in introduction/elimination rules dos not hold; rather, $\Delta x.M$ plays the role of *all* the constructors, the particular choice depending on the type of x.

Let us call a formula occurrence in a derivation or proof in $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee}$which is both the conclusion of an I rule or $\perp_c$ and the major premise of an E rule a maximum formula, and extend the notion of normal derivation accordingly. It then still holds that a maximum formula is the type of a certain part of a redex. It also still holds that for any derivation there is a normal derivation of the same type, and the proof proceeds in the same manner as in the case of $\lambda^{\perp,\supset,\wedge,\vee}$by defining a certain set of reduction rules. The next subsection shows how these reduction rules have been defined in the literature.

5.2 Standard classical normalization procedures

We shall mainly consider two systems for classical first-order proof normalization: Prawitz' system from [Pra65] and the system for which Stålmarck proves SN in [Sta91].

In the original works, the reductions are stated from derivations to derivations. We find it more convenient to state them as reductions on constructions.

In side conditions of some of the following rules we say that a term M is *free for* a general context C if no variable free in M is bound in C. If E_1 and E_2 are either constructions or contexts or both we let $FV(E_1)$ denote the free variables of E_1 and we write $FV(E_1, E_2)$ for $FV(E_1) \cup FV(E_2)$. Sometimes a reduction is conditonal upon the typing of the construction (i.e. the kind of formula of which the construction represents a proof) or on the types of certain assumptions. In such cases type information is given in the construction, as in e.g. $\lambda x : \mathcal{P}.M : \mathcal{Q}$ which means that $\lambda x.M$ is a proof of $\mathcal{Q}$ with x marking an assumption of $\mathcal{P}$.

Prawitz' system

We first consider Prawitz's system which is as follows. The construction and type languages is that of $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$except that formulae containing $\vee, \exists$ and constructions containing injection, case, brackets, let are excluded. The type inference system is obtained from $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$by excluding $\vee I$, $\vee E$, $\exists I$, $\exists E$. The set of reductions is (1a), (1b), (1d) from $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$and in addition the following for $\Delta x.M : \mathcal{P}$ defined by cases over the structure of $\mathcal{P}$.

$$\begin{array}{llcl}
(Pa)\, \mathcal{P} \equiv \mathcal{Q} \wedge \mathcal{R} : & \Delta x.(M) & \rightarrow & < \Delta y.M\{x := \lambda u.y\; \pi_1(u)\}, \Delta z.M\{x := \lambda v.z\; \pi_2(v)\} > \\
(Pb)\, \mathcal{P} \equiv \mathcal{Q} \supset \mathcal{R} : & \Delta x.(M) & \rightarrow & \lambda u.\Delta y.M\{x := \lambda v.y\; (v\; u)\} \\
(Pd)\, \mathcal{P} \equiv \forall \alpha.\mathcal{Q} : & \Delta x.(M) & \rightarrow & \lambda^{\forall} \alpha.\Delta y.M\{x := \lambda v.y\; (v\; \alpha)\}
\end{array}$$

These rules are very similar to our (2a), (2b), (2d). The difference is that our Δ waits for a one-level destructor context before reduction can proceed while Prawitz' Δ does not; but if Prawitz' Δ is surrounded by a one-level destructor context, it can reduce to the same as our Δ, that is $D^1[\Delta x.M] \rhd^* \Delta z.M\{x := \lambda y.z\; D^1[y]\}$ holds in Prawitz' system.

Reduction rules corresponding to (2c), (2e) in the style of Prawitz' rules do not seem to exist; this is another peculiarity of $\vee$ and $\exists$.

The advantage of Prawitz' formulation is that given a construction M one can apply (Pa), (Pb), (Pd) to arrive at a construction M' such that all conclusions of $\perp_c$ are atomic, and this property is not disturbed by the other reductions (1a), (1b), (1d). This means that subsequent application of the rules (1a),(1b),(1d) does not yield new redexes of the former kind. As a consequence, Prawitz' WN proof [Pra65] III Theorem 2 p40 is simple and illuminating. A WN proof for (1a)-(1e), (2a)-(2e) is much more complicated (see [Sta91].) Also, taking $\vee$ and $\exists$ as derived symbols greatly simplifies the SN proof, even for the minimal and intuitionistic subsystems—compare Prawitz' proof in [Pra71] to Stålmarck's proof in [Sta91].

In closing the discussion of Prawitz' rule we mention that Prawitz in [Pra71] (p249, p254) also considers the rule

$$\Delta x.C[x\; M] \rhd M, \text{ if } M \text{ is free for } C,\; x \notin FV(M, C)$$

which has an obvious interpretation in the catch-throw paradigm.

Stålmarck's system

We now turn to Stålmarck' system. The construction and type language, and the type inference system is that of $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee,\forall,\exists}$. The set of reductions is (1a)-(1e) from $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee,\forall,\exists}$and in addition the rules described below.

The following notion will be convenient.

Definition **23** (Eliminative Contexts)

$$\begin{array}{lll} E & ::= & \pi_1([\,]) \mid \pi_2([\,]) \\ & & \mid case([\,]; x_1.M; x_2.N) \mid case(L; x_1.[\,]; x_2.N) \mid case(L; x_1.M; x_2.[\,]) \\ & & \mid M\ [\,] \mid [\,]\ N \\ & & [\,]\tau \\ & & \mid let\ [<\alpha, y>] =\ [\,]\ in\ N \mid let\ [<\alpha, y>] =\ M\ in\ [\,] \end{array}$$

□

The following reduction rules are added to (1a)-(1e).

$$\begin{array}{lll} (Sa) & (\Delta x.M)\, N & \to\ \Delta u.M\{x := \lambda v.u\ (v\ N)\} \\ (Sb) & \pi_i(\Delta x.M) & \to\ \Delta u.M\{x := \lambda v.u\ \pi_i(v)\} \\ (Sc) & case(\Delta x.L; y_1.M_1; y_2.M_2) & \to\ \Delta u.L\{x := \lambda v.case(v; y_1.u\ M_1; y_2.u\ M_2)\} \\ (Sd) & (\Delta x.M)\ \tau & \to\ \Delta u.M\{x := \lambda v.u\ (v\ \tau)\} \\ (Se) & let\ [<\alpha, y>] =\ \Delta x.M\ in\ N & \to\ \Delta u.M\{x := \lambda v.let\ [<\alpha, y>] =\ v\ in\ u\ N\} \\ (R1) & \Delta x : \perp \supset \perp .M & \to\ M,\ \text{if } x \notin FV(M) \\ (R2) & x^{\perp\supset\perp}\ M^{\perp} & \to\ M \\ (P1) & E[case(L; x_1.M_1; x_2.M_2)] & \to\ case(L; x_1.EL[M_1]; x_2.E[M_2]) \\ (P2) & E[let\ [<\alpha, y>] =\ L\ in\ M] & \to\ let\ [<\alpha, y>] =\ L\ in\ E[M] \end{array}$$

The rules (Sa)-(Se) are almost precisely our (2a)-(2e). The only difference is in (Sc) and (Se) where the application of u is moved into the branches of the case-construction and into the second expression of the let-construction. The motivation for this difference as well as for adopting the rules (R1)-(R2) pertains to the method that Stålmarck applies for the SN proof.

The rules (P1) and (P2) are under the restriction that the subterms $case(L; x_1.M_1; x_2.M_2)$ and $let\ [<\alpha, y>] =\ L\ in\ M$ of the left hand sides respectively are normal forms w.r.t. the other rules they are grouped with above. The purpose of these rules is to establish the subformula property, see [Gir89] ch. 10. They are only necessary when $\vee, \exists$ are taken as primitive.

We conclude that the core of $\lambda^{\perp,\supset,\wedge,\vee,\forall,\exists}$is very similar to the core of Stålmarck's system.

Stålmarck considers a few more rules in a proof of WN for (1a)-(1e), (Sa)-(Se). The extra rules are adopted for the purpose of making an induction proof go through. These rules look peculiar at a first glance, *e.g.*

$$(*)\ \ \pi_i(\Delta x.C[x\ M]) \rhd \Delta u.u\ \pi_i(\Delta x.C[u\ \pi_i(M)])$$

where it is to be understood that *all* subterms of form $x\ M$ should be replaced by $u\ \pi_i(M)$. Such rules can, however, be perceived as on-line program transformations that "make sense" operationally.

For instance, the above rule can be explained in terms of the catch-throw paradigm as follows. Suppose that we have a projection of a $\Delta x.M$, which apparently has an outermost catch and where there are throw's, *e.g.* subexpressions of form $x\ N$ to the catch deep inside M. It is then tempting to say that the left hand side reduces to $\Delta u.(\pi_i(C[u\ \pi_i(N)]))$. Here we have simply moved the projection to the term which is thrown, and we have put a projection around the entire term $C[\bullet]$ in case no value is ever thrown; if there *is* a throw it simply jumps over this projection. However, it may be the case that x occurs in M, just not yet in the throw form $x\ N$. What the rule (*) above does is that if such a throw should pop up and be activated, then it is caught by $\Delta x.$, the projection of it is calculated, and the entire result is thrown to u.

So the rule makes sense, but what exactly does it mean to make sense? The intuition is that it *preserves semantics.* In the framework of an equational theory with an axiomatization of observational congruence, we would hope to be able to *prove* rules as the above as equalities. This is another reason why one should study $\lambda_\Delta^{\perp,\supset}$ and relatives as typed equational theories.

Seldin's system

We finally mention our last source for classical proof normalization procedures, viz. Seldin's systems [Sel86], [Sel89]. These are based not on the $\perp_c$ rule, but on Peirce's axiom and the intuitionistic rule $\perp_i$. These rules are related to Stålmarck's and Prawitz' rules in a certain sense that can be made more precise. In fact we would claim that the core of all three systems are variations of the same theme. In stead of our λ_Δ-calculus we could have taken both a Δ and ∇ as primitive and typed them by Peirce's axiom and $\perp_i$, repectively.

5.3 Proof theory of classically typed λ_Δ-calculi

Chapter III and IV of [Pra65], and with them other texts on proof theory, proceeds as follows. (*i*) One considers a set of reduction rules (with the property that when no reduction rule applies, the derivation is normal, see subsection 1.) (*ii*) One shows that this system has WN or SN. (*iii*) One gives a syntactic characterization of the normal forms. (*iv*) One deduces corollaries from the preceding two results. These corollaries state assertions of the form: "whenever $\mathcal{P}$ is provable then ..." and the metaproof starts by saying "Let M be a normal proof," and then proceeds using the syntactic characterization. This explains one significant aspect of the Inversion Principle and Theorem: proofs without maximum formulae have a form which is particularly convenient in metaproofs.

Below we carry out this program for our λ_Δ-calculi. That this can be done shows that not only are the reduction rules of *e.g.* $\lambda_\Delta^{\perp,\supset,\forall}$similar (in some informal sense) to those of Prawitz' system, the former system also passes the important test of being suitable for significant proof theoretical applications.

For step (*ii*) in the program we refer back to SN of $\lambda_{\Delta}^{\perp,\supset}$.[12]

Recall that we have defined a normal construction as one with no redexes. Prawitz' FND theorem in [Pra65] III §2 syntactically characterizes *derivations*, while we find it more convenient to give a characterization of the corresponding normal constructions. Such characterizations are standard in λ-calculus, see *e.g.* [Hin86] p14 for a result from the untyped world.

DEFINITION **24** Define the mutually recursive syntactic classes E, C, I:

$$\begin{array}{lcl} E & ::= & x \mid E\ I \mid \pi_i(E) \mid \mathrm{case}(E; x.I; y.I) \mid E\ \tau \mid \mathrm{let}\ [<\alpha, y>] = E\ \mathrm{in}\ I \\ C & ::= & E \mid \Delta x.C \\ I & ::= & C \mid \lambda x.I \mid\ < I, I >\ \mid \mathrm{in}_i(I) \mid \lambda^{\forall}\alpha.I \mid [<\tau, I>] \end{array}$$

□

THEOREM **7** *(Form of Normal Deductions.) Suppose that* $\Gamma \vdash M : \mathcal{P}$ *in* $\lambda_{\Delta}^{\perp,\supset,\wedge,\vee,\forall,\exists}$. *If* M *is normal then* M *conforms to* I.

PROOF: By the typing rules, a destructor surrounding a constructor would be a redex. A destructor surrounding a control operator is also a redex. So, by the set of reduction rules, there cannot be a destructor surrounding a constructor or control operator, *i.e.* destructors can only surround destructors and variables.

By the $\perp_c$ rule, the argument of a control operator must have type $\perp$. By the type system, constructors never have type $\perp$, so a control operator cannot surround a constructor. □

Similar theorems for the three other classical theories are easily obtained.

So every normal construction can be factored into three parts: an E part, a C part and an I part. The reader may like to visualize this situation by the following "picture:"

$$\lambda x_1.\ldots\lambda x_n.\Delta k_1.\ldots\Delta k_m.\pi_1(\ldots\pi_1(x_1)\ldots)$$

In fact, a normal construction M will contain several such pictures. For instance, suppose that N_1 and N_2 are such pictures. Then so is the normal construction

$$\lambda x.\Delta k.\mathrm{case}(x; y.N_1; z.N_2)$$

[12]We should also extend the SN result to the other systems. There are two practical difficulties in this. (1) The Postponement lemma must be extended to the larger systems. This is a straight-forward but excessively laborious task which we have not undertaken. (2) The proof of SN for $\lambda_{\Delta}^{\perp,\supset}$(1-2) from $\lambda_{\Delta}^{\perp,\supset}$(1) does not work when $\vee, \exists$ are present. We would therefore have to use a more complicated method such as that of [Sta91]. However, for reasons pertaining to technicalities in his proof Stålmarck's rules corresponding to our (2c) and (2e) are slightly different from ours and unnatural in a certain sense, and it is not clear whether one can change the idea in the proof to cope with our rules (2c) and (2e.)

Notice how the grammar above for I corresponds to the characterization of derivations in [Pra65] III §2 Theorem 3. The *only* difference is that we can have a number (in stead of just one) Δ's in sequence. This is because we have omitted Prawitz' restriction on the ($\perp_c$) rule that the conclusion be different from $\perp$, see [Pra65] p20. Alternatively, we could have adopted a rule to get rid of such sequences, *e.g.* $\Delta x.\Delta y.M \to \Delta x.M\{y := \lambda z.z\}$.

A more abstract form of normal deductions is also possible. The constructions can be factored into three groups: (1) A number of destructors indexed by j. The form of destructors is $E^j(x, y_1, \ldots, y_n)$ where x is the major premise and $y_1 \ldots y_n, n \geq 0$ are minor premises. (2) A number of control operators, C^j. (3) A number of constructors. The form of constructors is $I^j(z_1, \ldots z_n)$ (we ignore eigenterms.) (4) Variables.

Now Definition 24 and Theorem 7 reads:

DEFINITION **25** Define the mutually recursive syntactic classes E', C', I':

$$\begin{array}{lcl} E' & ::= & x \mid E^j(E', I'_1, \ldots, I'_n) \\ C' & ::= & E' \mid C^j(C') \\ I' & ::= & C' \mid I^j(I'_1, \ldots, I'_n) \end{array}$$

□

COROLLARY **5** *(Form of Normal Deductions.) Suppose that* $\Gamma \vdash M : \mathcal{P}$ *in* $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. *If* M *is normal then* M *conforms to* I'.

PROOF: By theorem 7. □

We now turn to the Corollaries of FND. Specifically, we consider (1) *Consistency*; (2) *Craig-Lyndon's Interpolation Theorem*; (3) *The Subformula Property.* As far as we know, these are the most common Corollaries, see [Pra65] III §2-3, [Sel86], [Sel89], [Gal87] sec. 6.5, [Tak75] ch. I §6.

The reader familiar with [Pra65] III §2-3 will have no trouble proving the first two results using our FND. In fact, a careful analysis of the proofs in [Pra65] shows that the critical property of the FND is exactly that there are never constructors or control operators under a destructor.

COROLLARY **6** *(Consistency) There is no proof of* $\perp$ *in* $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$.

COROLLARY **7** *(Craig-Lyndon's Interpolation Theorem) Suppose that* $\Gamma \vdash M : \mathcal{P}$ *in* $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. *Then there is an interpolation formula* $\mathcal{Q}$ *such that* $\Gamma \vdash L : \mathcal{Q}$ *and* $x : \mathcal{Q} \vdash K : \mathcal{P}$ *in* $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$, *and such that every free eigen variable or function symbol that occurs positively [negatively] in* $\mathcal{Q}$ *occurs positively [negatively] in both* $\mathcal{P}$ *and some formula of* Γ.

The preceding two results can also be proved for $\lambda_\Delta^{\perp,\supset,\forall}$by the same abstract FND. For well-known problems caused by disjunction and existential quantifier (see [Gir89] ch. 10) the following result holds in $\lambda_\Delta^{\perp,\supset,\forall}$, but not in $\lambda_\Delta^{\perp,\supset,\wedge,\vee,\forall,\exists}$. The proof can be adapted from [Pra65] right away.

COROLLARY 8 *(Subformula Property) Suppose that* $\Gamma \vdash M : \mathcal{P}$ *in* $\lambda_\Delta^{\perp,\supset,\forall}$. *If* M *is normal, then every formula occurrence in the derivation that* M *represents is a subformula of* $\mathcal{P}$ *or of some formula in* Γ*, except for assumptions discharged by the* $\perp_c$ *rule and for sequences of occurrences of* $\perp$ *such that the first occurrence stands immedieately below such an asssumptions and such that the* $i+1$*'th occurrence stands immedieately below the* i*'th occurrence.*

We did not apply rule (3) or (4) above, so it is perhaps suitable that this section end with a proof theoretical explanation of these two rules. Consider rule (3). The left hand side represents:

$$\dfrac{\dfrac{[k : \mathcal{P} \supset \perp] \quad M : \mathcal{P}}{k\ M : \perp}}{\Delta k.k\ M : \mathcal{P}}$$

which reduces to $M : \mathcal{P}$. Let us for a moment imagine that we had taken the double-negation elimination rule

$$\dfrac{M : (\mathcal{P} \supset \perp) \supset \perp}{\Delta'(M) : \mathcal{P}}$$

in stead of our principle for indirect inference. Then rule (3) reads

$$\Delta'(\lambda k.k\ M) \rhd M \text{ if } k \notin FV(M)$$

where the left hand side now represents

$$\dfrac{\dfrac{\dfrac{[k : \mathcal{P} \supset \perp] \quad M : \mathcal{P}}{k\ M : \perp}}{\lambda k.k\ M : (\mathcal{P} \supset \perp) \supset \perp}}{\Delta'(\lambda k.k\ M) : \mathcal{P}}$$

which reduces to $M : \mathcal{P}$. This rule says that if we have a proof M of $\mathcal{P}$ then nothing is gained by first turning M into a new proof M' of the double-negated formulae $(\mathcal{P} \supset \perp) \supset \perp$ and then turning that proof into a proof M'' of the original formula $\mathcal{P}$. This is a kind of inversion priciple: double-negation introduction and -elimination are inverses. Just as the inversion priciple for the other introduction and elimination rules is reflected by a similar inversion priciple in constructions, we have here that the double-negation inversion principle is reflected by the fact that the constructions $\Delta'(\bullet)$ and $\lambda k.k\ M$ are inverses. The symmetry is of course not as beatiful as in the minimal constructions.

Rule (4) with left hand side

$$\cfrac{\cfrac{k : \mathcal{P} \supset \bot \qquad \cfrac{\cfrac{k : \mathcal{P} \supset \bot \qquad \begin{array}{c}[d : \bot \supset \bot]\\ \vdots \\ M : \mathcal{P}\end{array}}{k\ M : \bot}}{\Delta d.k\ M : \mathcal{P}}}{k\ \Delta d.k\ M : \bot}}{\Delta k.k\ \Delta d.k\ M : \mathcal{P}}$$

which reduces to M, can be justified similarly.

6 Definability

This section brings together the notion of definability of connectives, *e.g.* conjunction, in logic with the notion of definability of constructions, *e.g.* pairs, in typed λ-calculi. We have not seen any general definition of definability of constructions in the literature; therefore Subsection 1 develops a notion of definability from the notions already known in logic and λ-calculus. Subsection 2 shows that in $\lambda_{\Delta}^{\bot,\supset}$ $[\lambda_{\Delta}^{\bot,\supset,\forall}]$ pairs and sums [and brackets] are definable according to the definition developed. Subsection 3 relates the definitions to the similar definitions in F_2. This leads to reflections on the Inversion Principle in a number of areas.

6.1 Defining definability

In this and the following subsections $\vdash \mathcal{P}$ means: there exists an M such that $\vdash M : \mathcal{P}$. We use the following abbreviations: $M = \lambda^{\bot,\supset,\wedge,\vee}$, $C = \lambda_{\Delta}^{\bot,\supset,\wedge,\vee}$, $MQ = \lambda^{\bot,\supset,\wedge,\vee,\forall,\exists}$, $CQ = \lambda_{\Delta}^{\bot,\supset,\wedge,\vee,\forall,\exists}$, $M' = \lambda^{\bot,\supset}$, $C' = \lambda_{\Delta}^{\bot,\supset}$, $MQ' = \lambda^{\bot,\supset,\forall}$, $CQ' = \lambda_{\Delta}^{\bot,\supset,\forall}$.

The two following notions of definability of connectives are from [Pra65] (pp58-59.)

Definition **26** (Strong definability) For $K = M, C$ a connective γ [or quantifier γ] is *strongly definable* in K $[KQ]$ if for every pair of formulae $\mathcal{P}, \mathcal{Q}$ [every formula $\mathcal{P}$] there is a formula $\mathcal{R}$ not containing γ such that $\vdash^{K} ((\mathcal{P}\gamma\mathcal{Q}) \supset \mathcal{R}) \wedge (\mathcal{R} \supset (\mathcal{P}\gamma\mathcal{Q}))$ $[\vdash^{KQ} (\gamma x\mathcal{P} \supset \mathcal{R}) \wedge (\mathcal{R} \supset \gamma x\mathcal{P})]$. The constant $\bot$ is strongly definable in K $[KQ]$ if there is a formula $\mathcal{R}$ not containing γ such that $\vdash^{K} (\bot \supset \mathcal{R}) \wedge (\mathcal{R} \supset \bot)$ $[\vdash^{KQ} (\bot \supset \mathcal{R}) \wedge (\mathcal{R} \supset \bot)]$. □

For a connective γ [a quantifier γ] a γ-compositional translation $\underline{\bullet}$ means a translation from formulae to formulae such that $\underline{\mathcal{P}\delta\mathcal{Q}} \equiv \underline{\mathcal{P}}\delta\underline{\mathcal{Q}}$ $[\underline{\delta x\mathcal{Q}} \equiv \delta x\underline{\mathcal{Q}}]$ when δ is not γ and such that $\underline{\mathcal{R}}$ does not contain γ. $\underline{\bullet}$ is a $\bot$-compositional translation if $\underline{\mathcal{P}} \equiv \mathcal{P}[\mathcal{R}/\bot]$ for some $\mathcal{R}$ not containing $\bot$.

Definition **27** (Weak definability) For $K = M, C$ a connective γ [or quantifier γ] is *weakly definable* in K $[KQ]$ if there exists a γ-compositional translation such that $\vdash^{K} \mathcal{P}$ iff $\vdash^{K} \underline{\mathcal{P}}$ $[\vdash^{KQ} \mathcal{P}$ iff $\vdash^{KQ} \underline{\mathcal{P}}]$. The constant $\bot$ is weakly definable in K $[KQ]$ if there is a $\bot$-compositional translation such that $\vdash^{K} \mathcal{P}$ iff $\vdash^{K} \underline{\mathcal{P}}$ $[\vdash^{KQ} \mathcal{P}$ iff $\vdash^{KQ} \underline{\mathcal{P}}]$. □

Strong definability in K $[KQ]$ implies weak definability in K $[KQ]$. Weak definability of $\wedge, \vee$ [or $\exists$], respectively, in M $[MQ]$ implies strong definability of $\wedge, \vee$ [or $\exists$], respectively, in M $[MQ]$, see [Pra65] p59.

It is well-known (see *e.g.* [Pra65] p59) that in M $[MQ]$ no connective [and no quantifier] is weakly definable (and therefore not strongly definable either), while $\bot$ is weakly but not strongly definable. On the other hand it is well-known that *e.g.* conjuction and disjunction [and existential quantifier] are strongly definable in C $[CQ]$ (see *e.g.* [Men87].)

There are two problems with the definition of weak definability as a definition of definability of *e.g.* pairs in typed λ-calculi. First, it does not mention constructions at all. To see the second problem note that in the definition of weak definability both sides of the required biimplication concerns provability in the same (full) system K. What we would like in stead is some definition relating provability in a system *with* γ in the type language to provability in another system *without* γ in the type language, *e.g.* a connection between provability in CQ' and CQ. (The problems are even worse in the definition of strong definability.)

A natural change in the definition of weak definability is to require a γ-compositional translation on both constructions and formulae from one system K to another $\underline{K}$ such that (1) $\underline{K}$ is the same system as K except that everything concerning γ has been erased from the type language, construction language, type inference system and reduction mechanism; (2) $\Gamma \vdash^K M : \mathcal{P}$ iff $\underline{\Gamma} \vdash^{\underline{K}} \underline{M} : \underline{\mathcal{P}}$; (3) $M \rhd_K N$ iff $\underline{M} \rhd_{\underline{K}}{}^* \underline{N}$.

However it is easy to see that in both (2) and (3) the right to left implication will fail for all standard translations used for defining connectives in C and CQ. For instance, suppose that $\vdash^K M : \mathcal{P}$ where M contains pairs and $\mathcal{P}$ contains conjunction, and let $\underline{\bullet}$ be a $\wedge$-compositional translation. So $\vdash^K \underline{M} : \underline{\mathcal{P}}$, and since $\underline{\bullet}$ is idempotent $\vdash^{\underline{K}} \underline{M} : \underline{(\underline{\mathcal{P}})}$. However it does not hold that $\vdash^K M : \underline{\mathcal{P}}$, since M contains pairs and $\mathcal{P}$ does not contain $\wedge$.

So we may only require the left to right implications in (2) and (3), which we might call *representation of inference trees* and *representation of reduction sequences.* The following definition from [Bar84] p567 does essentially this (for pairs in M'), but in stead of requiring the above "big step" representations it requires representation of each inference *rule* and each reduction *rule.*

DEFINITION **28** (Pairing, Definability of pairs) A *pairing* in M' for types $\mathcal{P}_1, \mathcal{P}_2$ is a triple of constructions D, D_1, D_2 and a type $\mathcal{R}$ such that (1) $\vdash^{MS} D : \mathcal{P}_1 \supset \mathcal{P}_2 \supset \mathcal{R}$, $\vdash^{MS} D_i : \mathcal{R} \supset \mathcal{P}_i$ and (2) $D_i(DM_1M_2) =_{MS} M_i$ for all M_1, M_2 with $\vdash^{MS} M_i : \mathcal{P}_i$. $\Box$

It is well known that there are types $\mathcal{P}_1, \mathcal{P}_2$ for which no pairing exists in M', see [Bar84] p567. As a quick way of checking that the result holds, simply note that if pairings existed for all types in M' then $\wedge$ would be strongly definable in M—which $\wedge$ is not, by Prawitz' result mentioned above.

The definition of pairing uses implication; a step from M to, say, $\pi_1(M)$ is represented be the application of D_1 to M. If we were after a general notion of definability this would be unfortunate, but we are mainly interested in definability of pairs, sums and brackets. However, the definition using implication becomes rather clumsy in the case of sums and brackets, so we resort to a formulation using translations.

By a pair-compositional translation we mean a translation $\underline{\bullet}$ from constructions to constructions such that $\underline{M}$ does not contain any pairs or projections and such that $\underline{\bullet}$ propagates to the subterms of M when M is not a pair or a projection. Similarly for sums and brackets.

We then arrive at the final definition of definability of pairs, sums and brackets, which we like to think of as a synthesis of the two preceding definitions. In each of the three cases the definition requires that the relevant inference rules and reduction rule hold derived. The definition is straight-forward to extend to λ- and $\lambda^{\forall}$- abstraction. In the definition a horisontal line denotes an implication from the assertion above the line to the assertion below the line.

Definition **29** (Definability of pairs, sums, brackets) In the following K is one of M', C', MQ', CQ'.

A *pairing* in K is a $\wedge$-compositional translation on formulae and pair-compositional translation on constructions such that

$$\frac{\Gamma \vdash^{K} \underline{M_1} : \underline{\mathcal{P}_1} \quad \Gamma \vdash^{KS} \underline{M_2} : \underline{\mathcal{P}_2}}{\Gamma \vdash^{K} \underline{< M_1, M_2 >} : \underline{\mathcal{P}_1 \wedge \mathcal{P}_2}} \frac{\Gamma \vdash^{K} \underline{M} : \underline{\mathcal{P}_1 \wedge \mathcal{P}_2}}{\Gamma \vdash^{K} \underline{\pi_i(M)} : \underline{\mathcal{P}_i}} \underline{C[\pi_i(< M_1, M_2 >)]} \rhd^{*}_{K} \underline{C[M_i]}$$

A *summing* in K is a $\vee$-compositional translation on formulae and sum-compositional translation on constructions such that

$$\frac{\Gamma \vdash^{K} \underline{M_1} : \underline{\mathcal{P}_1}}{\Gamma \vdash^{K} \underline{\mathrm{in}_1(M_1)} : \underline{\mathcal{P}_1 \vee \mathcal{P}_2}} \qquad \frac{\Gamma \vdash^{K} \underline{M_2} : \underline{\mathcal{P}_2}}{\Gamma \vdash^{K} \underline{\mathrm{in}_2(M_2)} : \underline{\mathcal{P}_1 \vee \mathcal{P}_2}}$$

$$\frac{\Gamma \vdash^{K} \underline{L} : \underline{\mathcal{P}_1 \vee \mathcal{P}_2} \quad \Gamma, y_1 : \underline{\mathcal{P}_1} \vdash^{K} \underline{N_1} : \underline{\mathcal{R}} \quad \Gamma, y_2 : \underline{\mathcal{P}_2} \vdash^{K} N_2 : \underline{\mathcal{R}}}{\Gamma \vdash^{K} \underline{\mathrm{case}(L; y_1.N_1; y_2.N_2)} : \underline{\mathcal{R}}}$$

$$\underline{C[\mathrm{case}(\mathrm{in}_i(M_i); y_1.N_1; y_2.N_2)]} \rhd^{*}_{K} \underline{C[N_i[M_i/y_i]]}$$

A *bracketing* in K is an $\exists$-compositional translation on formulae and bracket-compositional translation on constructions such that

$$\frac{\Gamma \vdash^{K} \underline{L} : \underline{\mathcal{Q}[\tau/\alpha]}}{\Gamma \vdash^{K} \underline{[< \tau, L >]} : \underline{\exists \alpha.\mathcal{Q}}} \qquad \frac{\Gamma \vdash^{K} \underline{M} : \underline{\exists \alpha.\mathcal{Q}} \quad \Gamma, y : \underline{\mathcal{Q}} \vdash^{K} \underline{N} : \underline{\mathcal{R}}}{\Gamma \vdash^{K} \underline{\mathrm{let}\ [< \alpha, y >] = M\ \mathrm{in}\ N} : \underline{\mathcal{R}}}$$

$$\underline{C[\mathrm{let}\ [< \alpha, y >] = [< \tau, L >]\ \mathrm{in}\ N]} \rhd^{*}_{K} \underline{C[N[\tau/\alpha, M/y]]}$$

□

6.2 Definability of pairs, sums and brackets in $\lambda_{\Delta}^{\perp,\supset,\forall}$

Definition **30** Let $\underline{\bullet}$ be the $\wedge$-, $\vee$- and $\exists$-compositional translation on formulae, and pair-, sum- and bracket-compositional translation on constructions defined below.

$$\begin{array}{lcl}
\underline{\mathcal{P}_1 \wedge \mathcal{P}_2} & \equiv & \neg(\underline{\mathcal{P}_1} \supset \neg\underline{\mathcal{P}_2}) \\
\underline{\mathcal{P}_1 \vee \mathcal{P}_2} & \equiv & \neg\underline{\mathcal{P}_1} \supset \neg\neg\underline{\mathcal{P}_2} \\
\underline{\exists\alpha.\mathcal{P}} & \equiv & \neg\forall\alpha.\neg\underline{\mathcal{P}}
\end{array}$$

$$\begin{array}{lcl}
\underline{< M_1, M_2 >} & \equiv & \lambda f.f\ \underline{M_1}\ \underline{M_2} \\
\underline{\pi_i(M)} & \equiv & \Delta k.\underline{M}\ \lambda x_1.\lambda x_2.k\ x_i \\
 & & \\
\underline{\mathrm{in}_i(M)} & \equiv & \lambda y_1.\lambda y_2.y_i\ \underline{M} \\
\underline{\mathrm{case}(M; x_1.N_1; x_2.N_2)} & \equiv & \Delta k.\underline{M}\ \lambda x_1.k\ \underline{N_1}\ \lambda x_2.k\ \underline{N_2} \\
 & & \\
\underline{[< \tau, M >]} & \equiv & \lambda f.f\ \tau\ \underline{M} \\
\underline{\mathrm{let}\ [< \alpha, y >] = N\ \mathrm{in}\ M} & \equiv & \Delta k.\underline{M}\ \lambda^{\forall}\alpha.\lambda y.k\ \underline{N}
\end{array}$$

□

We motivate each of the chosen translations of connectives and constructors in turn.

Conjunctive formulae: The definition for conjunctive formulae is standard in logic (see *e.g.* [Men87].) The corresponding definitions for pairing and projection are obtained by deriving the conjunction introduction and elimination rules in C; if one substitutes Griffin's $\mathcal{C}(\lambda x.M)$ for our $\Delta x.M$ then the defined constructions for pairing and projections above are exactly as in [Gri90].

The pairing construction is standard in untyped λ-calculus (see [Bar84]) while the projection construction is different from that normally employed in untyped λ-calculus, viz. $M\ \lambda x_1.\lambda x_2.x_i$. This latter definition does not work because $\lambda x_1.\lambda x_2.x_i$ has type $\mathcal{P}_1 \supset \mathcal{P}_2 \supset \mathcal{P}_i$ instead of the type $\mathcal{P}_1 \supset (\mathcal{P}_2 \supset \bot)$, which M expects. Changing the definition of conjunctive types to solve the problem is not possible; it leads to the type of a pair being dependent on which component a surrounding projection picks.[13] The Δ operator solves the problem by means of an application which turns the type of x_i into $\bot$ regardless of i. When the projection is calculated, the k reaches its Δ and can be removed by reduction rule (3) for Δ:

$$\begin{array}{lcl}
\underline{\pi_1(< M_1, M_2 >)} & \equiv & \Delta k.(\lambda f.f\ \underline{M_1}\ \underline{M_2})\ \lambda x_1.\lambda x_2.k\ x_1 \\
 & \triangleright & \Delta k.(\lambda x_1.\lambda x_2.k\ x_1)\ \underline{M_1}\ \underline{M_2} \\
 & \triangleright & \Delta k.k\ \underline{M_1} \\
 & \triangleright & \underline{M_1}
\end{array}$$

This shows how Δ works as a type coercion operator.

[13]If one is willing to settle for a weaker notion of pairs where both component must have the same type, then this problem vanishes. This shows that pairs with components of the same type can be represented in M (the simply typed λ-calculus.)

Existentially quantified formulae: The definition for existentially quantified formulae is also standard in logic (see *e.g.* [Men87].) The corresponding constructions are obtained by deriving the corresponding introduction and elimination rule in CQ. These constructions differ from those in [Gri90], which define *weak existence*, see [Gri90] [How80]. The present constructions represent bracketed pairs in the same way as ordinary pairs, and *let* in the same ways as projections.

Disjunctive formulae: The definition for disjunctive formulae is not standard in logic. The standard definition is $(\neg\mathcal{P}_1) \supset \mathcal{P}_2$. However, when one derives the disjunction introduction and elimination rules in CQ using this definition, the corresponding constructions for injection and case analysis are very different from those defining pairs, projections, bracketed pairs and let. Specifically, to get item 3 in the definitions of definability, one would need to add extra power to Δ.[14]

The present definition and corresponding defined constructions can be motivated via de Morgan's law $\mathcal{P}_1 \vee \mathcal{P}_2 \Leftrightarrow \neg(\neg\mathcal{P}_1 \wedge \neg\mathcal{P}_2)$, but it is much more suggestive to compare the definition to the corresponding definition in F_2 (see the next subsection.)

In concluding this small tour, note that injections are represented roughly as pairing, and case analysis is represented roughly as projections; or more generally that all constructors are represented in much the same way, and all destructors are represented in much the same way. This explains why the power required by each of the representations from Δ in order for the derived reduction rules to hold is also "much the same."

THEOREM **8** $\underline{\bullet}$ *defined above is a pairing, summing and bracketing in* CQ'. $\underline{\bullet}$ *restricted to propositional formulae and constructions is a pairing and summing in* C'.

PROOF: That the inference rules hold derived is to check. To show that the reduction rules hold derived, define a translation on contexts also denoted $\underline{\bullet}$ as follows. $\underline{[]} \equiv []$, and as for constructions otherwise (*e.g.* $\underline{\pi_i(C)} \equiv \Delta k.\underline{C}\ \lambda x_1.\lambda x_2.k\ x_i$ and $\underline{< C, M >} \equiv \lambda f.f\ \underline{C}\ \underline{M}$) It is then easy to prove by induction on C that $\underline{C}$ is again a general context and $\underline{C[M]} \equiv \underline{C}[\underline{M}]$.

Since $\rhd_{CQ}$ is compatible it therefore suffices to show that the derived reduction rules in the definition hold with empty context. This is easy to check in each of the three cases. □

We have not encountered anything similar to this theorem in the literature on control operators except in [Gri90] where similar results are considered informally for Felleisen's global (call-by-value) $\mathcal{C}$-operator. As pointed out [Gri90]

[14]This was not a problem in [Gri90] because Griffin's $\mathcal{C}$ is so strong that it contains both the power needed for the previous definitions and the power needed for his definition of disjunction and case analysis as well. But as already noted, $\mathcal{C}$ contains in fact too much power.

p52 right column the reduction rules does not generally hold derived in [Gri90]. It is not hard to see that under a call-by-name semantics the desired reduction rules would in fact hold in [Gri90], and under a call-by-value semantics the desired reduction rules would *not* hold derived in $\lambda_{\Delta}^{\perp,\supset,\forall}$. This suggests that call-by-value versions of control operators are inferior to corresponding call-by-name operators in defining constructions.

6.3 Relation to F_2; Inversion Principle revisited

We have already compared the definitions of pairing, *etc.* to the standard definitions in logic and untyped λ-calculus. We now consider the corresponding definitions in F_2 (minimal or intuitionistic second-order propositional calulus with constructions.) This sytem is obtained from M' by adding the following: (1) to the construction language the clauses[15] $\Lambda P.M$ and $M\ \{\mathcal{Q}\}$; (2) to the formula language $\forall^2 P.\mathcal{Q}$;[16] (3) to the set of reduction rules $(\Lambda P.M)\ \{\mathcal{Q}\} \rhd M$; (4) to the set of inference rules

$$\frac{M : \mathcal{P}}{\Lambda P.M : \forall^2 P.\mathcal{P}}$$

$$\frac{M : \forall^2 P.\mathcal{P}}{M\ \{\mathcal{Q}\} : \mathcal{P}[\mathcal{Q}/P]}$$

where in the first rule P must not be free in any open assumption.

In F_2, pairs and sums are defined by the following formulae and constructions (see *e.g.* [Gir89])

$$\begin{array}{rcl} \mathcal{P}_1 \wedge \mathcal{P}_2 & \equiv & \forall^2 R.(\mathcal{P}_1 \supset \mathcal{P}_2 \supset R) \supset R \\ \mathcal{P}_1 \vee \mathcal{P}_2 & \equiv & \forall^2 R.(\mathcal{P}_1 \supset R) \supset (\mathcal{P}_2 \supset R) \supset R \end{array}$$

$$\begin{array}{lcl} < M_1, M_2 > & \equiv & \Lambda R.\lambda f.f\ \underline{M_1}\ \underline{M_2} \\ \pi_i(M) & \equiv & \underline{M}\ \{\mathcal{P}_i\}\ \lambda x_1.\lambda x_2.x_1 \\ & & \\ \mathrm{in}_i(M) & \equiv & \Lambda R.\lambda y_1.\lambda y_2.y_i\ \underline{M} \\ \mathrm{case}(K; x_1.N_1; x_2.N_2) & \equiv & \underline{M}\ \{\mathcal{R}\}\ \lambda x_1.N_1\ \lambda x_2.N_2 \end{array}$$

Note how the problem concerning pairs from the simply typed calculus is solved by a universal type abstraction in the pairing constructor and a type application in the projection destructor. Contrast this with the technique employed in C where instead of blowing up the result type of pairing to a universal

[15]This is the explicitly typed version of F_2. We should also modify the previous components to be explicit *e.g.* λ-abstractions. That will not be necessary, however, to illustrate the points we shall make.

[16]One does not need to take $\perp$ as primitive in the type language, but that shall not matter here.

type we collapsed part of the type of the arguments to the projection destructor into $\perp$ by means of applications of k. Notice also that the same relationship holds between the solution in F_2 and C in the case of sums.

Now, many more types than sums and pairs are definable in F_2 *e.g.* numerals, lists, trees, *etc.* see [Gir89], [Pie89]. In fact, all these types are special instances of a general notion, viz. *inductive types* ([Pie89], [Gir89]) or *data systems* ([Boh85], [Lei83]), and all these can be represented in F_2 as shown by Böhm and Beraducci [Boh85] and Leivant [Lei83].

The question then naturally arises whether lists, trees, *etc.* or in general all data systems or inductive types can be defined in C too. It turns out that this is not possible with the rules (2-4) for Δ. We are currently investigating the possibility of adding rules to allow such definitions while at the same time retaining the desirable properties of our system.

The basic problem is that our rules are taylored to model control operators where a ∇ bubbles out of it's context, while in the case of type coercion operators one would like $\nabla(j\ M)$ to reduce ti M. We might say that in both the control operator and type coercion case there is an inversion principle present, but the two cases differ in how one gets to the situation where the inversion principle applies. In the control operator case the ∇ bubbles up the its Δ skipping the context inbetween. In the type coercion case, a ∇ around an application of a variable which is bound by a Δ is cancelled.

The Inversion Principle also occurs in other areas of which we briefly review below.

Program transformation. The *Listless transformer* [Wad84], [Wad85] and the *Deforestation* algorithm [Wad88], [Fer88] transform functional programs which use intermediate data structures into semantically equivalent programs which do not use intermediate data structures. An intermediate structure is something which is constructed and subsequently destructed. For instance, an implementation of the factorial function may construct the list $[1 \ldots n]$ and then use another function to compute the product of the elements of the list; in this case we have an intermediate list. Examples more familiar to the proof theorist include *e.g.* projection of a pair and case analysis of an injection.

Imposing *e.g.* a Milner type discipline [Mil78] on the functional programming language one can think of the terms as constructions in a certain logic, and the Deforestation algorithm can be viewed as a proof normalizer. The inversion principle states that constructing and destructing intermediate structures are in a sense inverse operations.

As is often the case in program transformations, the Deforestation algorithm does not terminate for all input programs. This is essentially due to the fact that the construction language contains a fix-point operator. Current research [Wad93] apparently attempts to exploit the connection between Deforestation and proof normalization to obtain a new characterization of the set of functions for which the deforestation algorithm terminates.

Dynamic typing. When one tries to take the head of an atom in an untyped functional programming language such as Scheme, one does not get a data bus error; the interpreter returns an error message. This is because all objects are supplied with a tag saying which kind of object they are. When the interpreter encounters an application it sees whether the operand is tagged as a function, and if not returns an error message.

This results in some computational overhead, but many of the tagging and subsequent untagging operations can be carried out before actually applying the interepreter to the program. Doing this is the job of a dynamic typing discipline. There seems to be some relations betweem the calculi studied in this paper with systems for dynamic typing as in [Hen93].

It would appear that the inversion principle encapsulates a common core in proof normalization, program evaluation, deforestation and dynamic typing, and this core is more than an informal principle: it is a syntactic correspondence.

A Proofs for section 3.1 and Postponement (section 3.4)

Proofs of lemmas 2, 3, 4 were left out in section 3.1. Also, lemma 15 (postponement) of section 3.4 was left unproven. These lemmas are given full proofs below.

Recall the parallel reductions :

$$\begin{array}{lll}
(1) & M \gg_\gamma M & \\
(2) & M \gg_\gamma M', N \gg_\gamma N' & \Rightarrow \; (\lambda x.M)\, N \gg_\gamma M'\{x := N'\} \\
(3) & M \gg_\gamma M' & \Rightarrow \; \Delta x.x\, M \gg_\gamma M' \, , \quad x \notin FV(M) \\
(4) & M \gg_\gamma M' & \Rightarrow \; \Delta x.x\, \nabla(x\, M) \gg_\gamma M' \, , \quad x \notin FV(M) \\
(5) & M \gg_\gamma M' & \Rightarrow \; \nabla(M)\, N \gg_\gamma \nabla(M') \\
(6) & M \gg_\gamma M', N \gg_\gamma N' & \Rightarrow \; M\, N \gg_\gamma M'\, N' \\
(7) & M \gg_\gamma M' & \Rightarrow \; \lambda x.M \gg_\gamma \lambda x.M' \\
(8) & M \gg_\gamma M' & \Rightarrow \; \Delta x.M \gg_\gamma \Delta x.M'
\end{array}$$

$$\begin{array}{lll}
(1) & M \gg_\delta M & \\
(2) & M \gg_\delta M', N \gg_\delta N' & \Rightarrow \; (\Delta x.M)\, N \gg_\delta \Delta \kappa.M'\{x := \lambda f.\kappa\,(f\, N')\} \\
(3) & M \gg_\delta M', N \gg_\delta N' & \Rightarrow \; M\, N \gg_\delta M'\, N' \\
(4) & M \gg_\delta M' & \Rightarrow \; \lambda x.M \gg_\delta \lambda x.M' \\
(5) & M \gg_\delta M' & \Rightarrow \; \Delta x.M \gg_\delta \Delta x.M'
\end{array}$$

Proof of Lemma 2

To prove Lemma 2 we must first prove some substitution substitution lemmas. First, we remind of the standard substitution lemma.

LEMMA **16** *If $x \not\equiv y$ and $x \notin FV(L)$ then*

$$M\{x := N\}\{y := L\} \equiv M\{y := L\}\{x := N\{y := L\}\}.$$

PROOF: Structural induction on M (See [Bar84], 2.1.16.) □

β-reduction is substitutive:

LEMMA **17** *For any terms M, N, L of λ_Δ, if $M \rhd_\beta N$ then $M\{x := L\} \rhd_\beta N\{x := L\}$.*

PROOF: Since β-reduction of the pure λ-calculus is substitutive (cf. [Bar84], 3.1.15 and 3.1.16) it is easy to see that this is also the case for $\rhd_\beta$ of λ_Δ. □

Now substitution lemma for for $\gg_\gamma$:

LEMMA **18** *$M \gg_\gamma M'$ and $N \gg_\gamma N'$ imply $M\{x := N\} \gg_\gamma M'\{x := N'\}$.*

PROOF: By induction on the definition of $M \gg_\gamma M'$.

Case 1 : $M \gg_\gamma M'$ is $M \gg_\gamma M \equiv M'$. The claim is proven by structural induction on M:

$\boxed{M \equiv x}$ The claim is obvious.

$\boxed{M \equiv y \not\equiv x}$ The claim is obvious.

$\boxed{M \equiv \lambda y.P, y \not\equiv x}$ We have $M\{x := N\} \equiv \lambda y.P\{x := N\}$ and by induction hypothesis we get $P\{x := N\} \gg_\gamma P\{x := N'\}$ and from this the claim follows by rule (7) of $\gg_\gamma$.

$\boxed{M \equiv \lambda x.P}$ The claim is obvious.

$\boxed{M \equiv P\,Q}$ $M\{x := N\} \equiv P\{x := N\}\,Q\{x := N\}$, and the claim follows by induction hypothesis and rule (6) of $\gg_\gamma$.

$\boxed{M \equiv \Delta y.P, y \not\equiv x}$ $M\{x := N\} \equiv \Delta x.P\{x := N\}$ and by induction hypothesis and rule (8) of $\gg_\gamma$ the claim follows.

$\boxed{M \equiv \Delta x.P}$ The claim is obvious.

Case 2 : $M \gg_\gamma M'$ is $M \equiv (\lambda y.P)\,Q \gg_\gamma P'\{y := Q'\} \equiv M'$, $y \not\equiv x$, with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$. We have $M\{x := N\} \equiv (\lambda y.P\{x := N\})Q\{x := N\}$. By induction hypothesis, $P\{x := N\} \gg_\gamma P'\{x := N'\}$ and $Q\{x := N\} \gg_\gamma Q'\{x := N'\}$, so by rule (2) of $\gg_\gamma$ and the standard substitution lemma we get

$$\begin{array}{rcl} M\{x := N\} & \gg_\gamma & P'\{x := N'\}\{y := Q'\{x := N'\}\} \\ & \equiv & P'\{y := Q'\}\{x := N'\} \\ & \equiv & M'\{x := N'\} \end{array}$$

where the conditions of the standard substitution lemma are satisfied by our variable conventions. The case where $y \equiv x$ is trivial and left out.

Case 3 : $M \gg_\gamma M'$ is $M \equiv \Delta y.y\,P \gg_\gamma M'$, $y \not\equiv x$, with $P \gg_\gamma M'$ and $y \notin FV(P)$. We have $M\{x := N\} \equiv \Delta y.y\,P\{x := N\}$ and $y \notin FV(P\{x := N\})$ follows from $y \notin FV(P)$ by the variable conventions. By induction hypothesis we get $P\{x := N\} \gg_\gamma M'\{x := N'\}$ from which we get by rule (3) of $\gg_\gamma$ that$M\{x := N\} \equiv \Delta y.y\,P\{x := N\} \gg_\gamma M'\{x := N'\}$. The case where $y \equiv x$ is trivial and left out.

Case 4 : $M \gg_\gamma M'$ is $M \equiv \Delta y.y\,\nabla(y\,P) \gg_\gamma M'$, $y \not\equiv x$, with $P \gg_\gamma M'$ and $y \notin FV(P)$. We have $M\{x := N\} \equiv \Delta y.y\,\nabla(y\,P\{x := N\})$. By induction hypothesis, $P\{x := N\} \gg_\gamma M'\{x := N'\}$, and $y \notin FV(P\{x := N\})$ follows from $y \notin FV(P)$ by variable conventions. From rule (4) of $\gg_\gamma$ we get, therefore, that $M\{x := N\} \equiv \Delta y.y\,\nabla(y\,P\{x := N\}) \gg_\gamma M'\{x := N'\}$. The case where $y \equiv x$ is trivial and left out.

Case 5 : $M \gg_\gamma M'$ is $M \equiv \nabla(P)\, Q \gg_\gamma \nabla(P') \equiv M'$ with $P \gg_\gamma P'$. We have $M\{x := N\} \equiv \nabla(P\{x := N\})\, Q\{x := N\}$, and by induction hypothesis we get $P\{x := N\} \gg_\gamma P'\{x := N\}$ from which by rule (5) of $\gg_\gamma$ $M\{x := N\} \equiv \nabla(P\{x := N\})\, Q\{x := N\} \gg_\gamma \nabla(P'\{x := N'\}) \equiv M'\{x := N'\}$.

Case 6 : $M \gg_\gamma M'$ is $M \equiv P\, Q \gg_\gamma P'\, Q' \equiv M'$ with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$. Then $M\{x := N\} \equiv P\{x := N\}\, Q\{x := N\}$ and the claim follows by induction hypothesis together with rule (6) of $\gg_\gamma$.

Case 7 : $M \gg_\gamma M'$ is $M \equiv \lambda y.P \gg_\gamma \lambda y.P' \equiv M'$ with $P \gg_\gamma P'$. By induction hypothesis together with rule (7) of $\gg_\gamma$ the claim follows.

Case 8 : $M \gg_\gamma M'$ is $M \equiv \Delta y.P \gg_\gamma \Delta y.P' \equiv M'$, $y \not\equiv x$, with $P \gg_\gamma P'$. We have $M\{x := N\} \equiv \Delta y.P\{x := N\}$, so by induction hypothesis and rule (8) of $\gg_\gamma$ the claim follows. The case where $y \equiv x$ is trivial and left out. □

Now we can prove

Lemma 2 $\gg_\gamma \models \diamond$.
Proof: Suppose $M \gg_\gamma M_1$ and $M \gg_\gamma M_2$. By induction on the definition of $M \gg_\gamma M_1$ we show $\exists M_3 : M_1 \gg_\gamma M_3, M_2 \gg_\gamma M_3$.

Case 1 : $M \gg_\gamma M_1$ is $M \gg_\gamma M \equiv M_1$. Choose $M_3 \equiv M_2$.

Case 2 : $M \gg_\gamma M_1$ is $M \equiv (\lambda x.P)\, Q \gg_\gamma P'\{x := Q'\} \equiv M_1$ with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$. We distinguish two subcases:

Subcase 2.1 : $M \gg_\gamma M_2$ is $(\lambda x.P)\, Q \gg_\gamma (\lambda x.P'')\, Q'' \equiv M_2$ with $P \gg_\gamma P''$ and $Q \gg_\gamma Q''$. By induction hypothesis there are terms P''', Q''' such that $P' \gg_\gamma P'''$, $P'' \gg_\gamma P'''$, $Q' \gg_\gamma Q'''$ and $Q'' \gg_\gamma Q'''$. Choose $M_3 \equiv P'''\{x := Q'''\}$, then the claim follows by Lemma 18 and rule (2) of $\gg_\gamma$.

Subcase 2.2 : $M \gg_\gamma M_2$ is $(\lambda x.P)\, Q \gg_\gamma P''\{x := Q''\} \equiv M_2$ with $P \gg_\gamma P''$ and $Q \gg_\gamma Q''$. By induction hypothesis there are terms P''', Q''' such that $P' \gg_\gamma P'''$, $P'' \gg_\gamma P'''$, $Q' \gg_\gamma Q'''$ and $Q'' \gg_\gamma Q'''$. Choose $M_3 \equiv P'''\{x := Q'''\}$, then the claim follows by Lemma 18.

Case 3 : $M \gg_\gamma M_1$ is $M \equiv \Delta x.x\, P \gg_\gamma P' \equiv M_1$ with $P \gg_\gamma P'$ and $x \notin FV(P)$. We distinguish two subcases:

Subcase 3.1 : $M \gg_\gamma M_2$ is $\Delta x.x\, P \gg_\gamma P'' \equiv M_2$ with $P \gg_\gamma P''$ and $x \notin FV(P)$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$.

Subcase 3.2 : $M \gg_\gamma M_2$ is $\Delta x.x\, P \gg_\gamma \Delta x.x\, P''$ with $P \gg_\gamma P''$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$. Then we have $M_1 \gg_\gamma M_3$ already, and $M_2 \gg_\gamma M_3$ follows by rule (3) of $\gg_\gamma$, since $x \notin FV(P)$ and $P \gg_\gamma P''$ imply $x \notin FV(P'')$.

Case 4 : $M \gg_\gamma M_1$ is $M \equiv \Delta x.x\, \nabla(x\, P) \gg_\gamma P' \equiv M_1$ with $P \gg_\gamma P'$ and $x \notin FV(P)$. We distinguish two subcases:

Subcase 4.1 : $M \gg_\gamma M_2$ is $\Delta x.x\, \nabla(x\, P) \gg_\gamma P'' \equiv M_2$ with $P \gg_\gamma P''$ and $x \notin FV(P)$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$.

Subcase 4.2 : $M \gg_\gamma M_2$ is $\Delta x.x\,\nabla(x\ P) \gg_\gamma \Delta x.x\,\nabla(x\ P'') \equiv M_2$ with $P \gg_\gamma P''$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$. Then we already have $M_1 \gg_\gamma M_3$, and $M_2 \gg_\gamma M_3$ follows by rule (4) of $\gg_\gamma$, because $x \notin FV(P)$ implies $x \notin FV(P'')$ by $P \gg_\gamma P''$.

Case 5 : $M \gg_\gamma M_1$ is $M \equiv \nabla(P)\,Q \gg_\gamma \nabla(P') \equiv M_1$ with $P \gg_\gamma P'$. We distinguish two subcases:

Subcase 5.1 : $M \gg_\gamma M_2$ is $\nabla(P)\,Q \gg_\gamma \nabla(P'') \equiv M_2$ with $P \gg_\gamma P''$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv \nabla(P''')$. Then $M_1 \gg_\gamma M_3$ and $M_2 \gg_\gamma M_3$ by rule (8) of $\gg_\gamma$.

Subcase 5.2 : $M \gg_\gamma M_2$ is $\nabla(P)\,Q \gg_\gamma \nabla(P'')\,Q'' \equiv M_2$ with $P \gg_\gamma P''$ and $Q \gg_\gamma Q''$. By induction hypothesis there is a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv \nabla(P''')$. Then $M_1 \gg_\gamma M_3$ by rule (8) of $\gg_\gamma$, and $M_2 \gg_\gamma M_3$ by rule (5) of $\gg_\gamma$.

Case 6 : $M \gg_\gamma M_1$ is $M \equiv P\ Q \gg_\gamma P'\,Q' \equiv M_1$ with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$. We distinguish three subcases:

Subcase 6.1 : $M \gg_\gamma M_2$ is $P\ Q \gg_\gamma P''\,Q'' \equiv M_2$. Induction hypothesis gives P''' and Q''' such that we can choose $M_3 \equiv P'''\,Q'''$.

Subcase 6.2 : $M \gg_\gamma M_2$ is $M \equiv (\lambda x.P_1)\,Q \gg_\gamma P_1''\{x := Q''\} \equiv M_2$ with $P \equiv \lambda x.P_1$, $P_1 \gg_\gamma P_1''$ and $Q \gg_\gamma Q''$. Now, $P \gg_\gamma P'$ implies $P' \equiv \lambda x.P_1'$ with $P_1 \gg_\gamma P_1'$, and so the induction hypothesis yields P''', Q''' such that $P_1' \gg_\gamma P'''$, $P_1'' \gg_\gamma P'''$, $Q' \gg_\gamma Q'''$ and $Q'' \gg_\gamma Q'''$. Choose $M_3 \equiv P'''\{x := Q'''\}$, then $M_1 \gg_\gamma M_3$ by rule (2) of $\gg_\gamma$, and $M_2 \gg_\gamma M_3$ by Lemma 18.

Subcase 6.3 : $M \gg_\gamma M_2$ is $M \equiv \nabla(P_1)\,Q \gg_\gamma \nabla(P_1'') \equiv M_2$ with $P \equiv \nabla(P_1)$ and $P_1 \gg_\gamma P_1''$. From $P \gg_\gamma P'$ it follows that $P' \equiv \nabla(P'')$ with $P_1 \gg_\gamma P''$. By induction hypothesis there is a term P''' such that $P'' \gg_\gamma P'''$ and $P_1'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$. Then $M_2 \equiv \nabla(P'')\,Q'' \gg_\gamma \nabla(P''') \equiv M_3$ by rule (5) of $\gg_\gamma$, and $M_2 \equiv \nabla(P_1'') \gg_\gamma \nabla(P''') \equiv M_3$ by rule (8) of $\gg_\gamma$.

Case 7 : $M \gg_\gamma M_1$ is $M \equiv \lambda x.P \gg_\gamma \lambda x.P' \equiv M_1$ with $P \gg_\gamma P'$. Then it must be that $M_2 \equiv \lambda x.P''$ with $P \gg_\gamma P''$. By induction hypothesis we get a term P''' such that we can choose $M_3 \equiv \lambda x.P'''$.

Case 8 : $M \gg_\gamma M_1$ is $M \equiv \Delta x.P \gg_\gamma \Delta x.P' \equiv M_1$ with $P \gg_\gamma P'$. We distinguish three subcases:

Subcase 8.1 : $M \gg_\gamma M_2$ is $M \equiv \Delta x.x\,P_1 \gg_\gamma P_1' \equiv M_2$ with $P_1 \gg_\gamma P_1'$ and $x \notin FV(P_1)$. Here it must be that $P \equiv x\,P_1$ and $P' \equiv x\,P_1''$ with $P_1 \gg_\gamma P_1''$. By induction hypothesis there is a term P''' such that $P_1' \gg_\gamma P'''$ and $P_1'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$. Then we already have $M_2 \gg_\gamma M_3$, and $M_1 \gg_\gamma M_3$ follows by rule (3) of $\gg_\gamma$, since $x \notin FV(P_1'')$ follows from $x \notin FV(P_1)$ and $P_1 \gg_\gamma P_1''$.

Subcase 8.2 : $M \gg_\gamma M_2$ is $M \equiv \Delta x.x\,\nabla(x\,P_1) \gg_\gamma P_1' \equiv M_2$ with $P_1 \gg_\gamma P_1'$ and $x \notin FV(P_1)$. Here it must be that $P \equiv x\,\nabla(x\,P_1)$ with $P' \equiv x\,\nabla(x\,P_1'')$ and $P_1 \gg_\gamma P_1''$. By induction hypothesis there is a term P''' such that $P_1' \gg_\gamma P'''$ and $P_1'' \gg_\gamma P'''$. Choose $M_3 \equiv P'''$. Then we already have $M_2 \gg_\gamma M_3$, and $M_1 \gg_\gamma M_3$ follows by rule (4) of $\gg_\gamma$, since $x \notin FV(P_1'')$ follows from $x \notin FV(P_1)$ and $P_1 \gg_\gamma P_1''$.

Subcase 8.3 : $M \gg_\gamma M_2$ is $M \equiv \Delta x.P \gg_\gamma \Delta x.P'' \equiv M_2$ with $P \gg_\gamma P''$. By induction hypothesis we get a term P''' such that $P' \gg_\gamma P'''$ and $P'' \gg_\gamma P'''$. Choose $M_3 \equiv \Delta x.P'''$, then $M_1 \gg_\gamma M_3$ and $M_2 \gg_\gamma M_3$ follow by rule (8) of $\gg_\gamma$. $\square$

Proof of Lemma 3

To prove Lemma 3 we prove a substitution lemma for $\gg_\delta$:

LEMMA **19** *$M \gg_\delta M'$ and $N \gg_\delta N'$ imply $M\{x := N\} \gg_\delta M'\{x := N'\}$.*

PROOF: By induction on the definition of $M \gg_\delta M'$.

Case 1 : $M \gg_\delta M'$ is $M \gg_\delta M \equiv M'$. The claim is proven by structural induction on M :

$\boxed{M \equiv x}$ The claim is obvious.

$\boxed{M \equiv y \not\equiv x}$ The claim is obvious.

$\boxed{M \equiv \lambda y.P, y \not\equiv x}$ We have $M\{x := N\} \equiv \lambda y.P\{x := N\}$. By induction we get $P\{x := N\} \gg_\delta P\{x := N'\}$, and the claim follows by rule (4) of $\gg_\delta$. The case where $y \equiv x$ is trivial and omitted.

$\boxed{M \equiv P\,Q}$ $M\{x := N\} \equiv P\{x := N\}\,Q\{x := N\}$, and the claim follows by induction and rule (3) of $\gg_\delta$.

$\boxed{M \equiv \Delta y.P, y \not\equiv x}$ $M\{x := N\} \equiv \Delta y.P\{x := N\}$, and the claim follows by induction and rule (5) of $\gg_\delta$. The case where $y \equiv x$ is trivial and omitted.

Case 2 : $M \gg_\delta M'$ is $M \equiv (\Delta y.P)\,Q \gg_\delta \Delta\kappa.P'\{x := \lambda f.\kappa\,(f\,Q')\} \equiv M'$, $y \not\equiv x$, with $P \gg_\delta P'$ and $Q \gg_\delta Q'$. We have $M\{x := N\} \equiv (\Delta y.P\{x := N\})\,Q\{x := N\}$. By induction, $P\{x := N\} \gg_\delta P'\{x := N'\}$ and $Q\{x := N\} \gg_\delta Q'\{x := N'\}$. By rule (2) of $\gg_\delta$ it follows that

$$\begin{array}{rcl}
M\{x := N\} & \gg_\delta & \Delta\kappa.P'\{x := N'\}\{y := \lambda f.\kappa\,(f\,Q'\{x := N'\})\} \\
& \equiv & \Delta\kappa.P'\{x := N'\}\{y := (\lambda f.\kappa\,(f\,Q'))\{x := N'\}\} \\
& \equiv & \Delta\kappa.P'\{y := \lambda f.\kappa\,(f\,Q')\}\{x := N'\} \\
& \equiv & M'\{x := N'\}
\end{array}$$

where the standard substitution lemma was invoked at the second equivalence (its conditions being satisfied in virtue of our variable conventions.) The trivial case where $y \equiv x$ is left out.

Case 3 : $M \gg_\delta M'$ is $M \equiv P\,Q \gg_\delta P'\,Q' \equiv M'$ with $P \gg_\delta P'$ and $Q \gg_\delta Q'$. We have $M\{x := N\} \equiv P\{x := N\}\,Q\{x := N\}$, and the induction hypothesis together with rule (3) of $\gg_\delta$ yield the claim.

Case 4 : $M \gg_\delta M'$ is $M \equiv \lambda y.P \gg_\delta \lambda y.P' \equiv M'$, $y \not\equiv x$, with $P \gg_\delta P'$. $M\{x := N\} \equiv \lambda y.P\{x := N\}$, and by induction hypothesis together with rule (4) of $\gg_\delta$ we get the claim. The case where $y \equiv x$ is trivial and omitted.

Case 5 : $M \gg_\delta M'$ is $M \equiv \Delta y.P \gg_\delta \Delta y.P' \equiv M'$, $y \not\equiv x$, with $P \gg_\delta P'$. We have $M\{x := N\} \equiv \Delta y.P\{x := N\}$, and induction hypothesis together with rule (5) of $\gg_\delta$ yield the claim. The case $y \equiv x$ is again omitted. □

This enables us to prove

LEMMA 3 *$\gg_\delta \models \diamond$.*

PROOF: Suppose $M \gg_\delta M_1$ and $M \gg_\delta M_2$. By induction on the definition of $M \gg_\delta M_1$ we show $\exists M_3 : M_1 \gg_\delta M_3, M_2 \gg_\delta M_3$.

Case 1 : $M \gg_\delta M_1$ is $M \gg_\delta M \equiv M_1$. Choose $M_3 \equiv M_2$.

Case 2 : $M \gg_\delta M_1$ is $M \equiv (\Delta x.P)\, Q \gg_\delta \Delta\kappa.P'\{x := \lambda f.\kappa\,(f\, Q')\} \equiv M_1$ with $P \gg_\delta P'$ and $Q \gg_\delta Q'$. We have two subcases:

Subcase 2.1 : $M \gg_\delta M_2$ is $(\Delta x.P)\, Q \gg_\delta \Delta\kappa.P''\{x := \lambda f.\kappa\,(f\, Q'')\} \equiv M_2$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis there are terms P''' and Q''' such that $P' \gg_\delta P'''$, $P'' \gg_\delta P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\delta Q'''$. Choose $M_3 \equiv \Delta\kappa.P'''\{x := \lambda f.\kappa\,(f\, Q''')\}$. From $Q' \gg_\delta Q'''$ it follows by rules (3) and (4) of $\gg_\delta$ that $\lambda f.\kappa\,(f\, Q') \gg_\delta \lambda f.\kappa\,(f\, Q''')$. From this, together with $P' \gg_\delta P'''$, it follows by Lemma 19 that $P'\{x := \lambda f.\kappa\,(f\, Q')\} \gg_\delta P'''\{x := \lambda f.\kappa\,(f\, Q''')\}$, and so by rule (5) of $\gg_\delta$ we get $M_1 \gg_\delta M_3$. By an analogous argument we see that also $M_2 \gg_\delta M_3$.

Subcase 2.2 : $M \gg_\delta M_2$ is $(\Delta x.P)\, Q \gg_\delta (\Delta x.P'')\, Q'' \equiv M_2$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis there are terms P''' and Q''' such that $P' \gg_\delta P'''$, $P'' \gg_\delta P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\delta Q'''$. Choose $M_3 \equiv \Delta\kappa.P'''\{x := \lambda f.\kappa\,(f\, Q''')\}$. Then, by rule (2) of $\gg_\delta$, $M_2 \gg_\delta M_3$. And by Lemma 19, $P'\{x := \lambda f.\kappa\,(f\, Q')\} \gg_\delta P'''\{x := \lambda f.\kappa\,(f\, Q''')\}$, and so, by rule (5) of $\gg_\delta$, $M_1 \gg_\delta M_3$.

Case 3 : $M \gg_\delta M_1$ is $M \equiv PQ \gg_\delta P'Q' \equiv M_1$ with $P \gg_\delta P'$ and $Q \gg_\delta Q'$. There are two subcases:

Subcase 3.1 : $M \gg_\delta M_2$ is $M \equiv (\Delta x.P_1)\, Q \gg_\delta \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\, Q'')\} \equiv M_2$ with $P_1 \gg_\delta P_1'$ and $Q \gg_\delta Q''$. Here it must be that $P \equiv \Delta x.P_1$ and $P' \equiv \Delta x.P_1''$ with $P_1 \gg_\delta P_1''$. Then, by induction hypothesis, there are terms P''' and Q''' such that $P_1' \gg_\delta P'''$, $P_1'' \gg_\delta P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\delta Q'''$. Choose $M_3 \equiv \Delta\kappa.P'''\{x := \lambda f.\kappa\,(f\, Q''')\}$. Then $M_1 \gg_\delta M_3$ by rule (2) of $\gg_\delta$, and $M_2 \gg_\delta M_3$ by Lemma 19.

Subcase 3.2 : $M \gg_\delta M_2$ is $P\, Q \gg_\delta P''\, Q'' \equiv M_2$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis we have terms P''' and Q''' such that rule (3) of $\gg_\delta$ establishes the claim with $M_3 \equiv P'''\, Q'''$.

Case 4 : $M \gg_\delta M_1$ is $M \equiv \lambda x.P \gg_\delta \lambda x.P' \equiv M_1$ with $P \gg_\delta P'$. Here we must have $M_2 \equiv \lambda x.P''$ with $P \gg_\delta P''$. Induction hypothesis gives a term P''' such that rule (4) of $\gg_\delta$ establishes the claim with $M_3 \equiv \lambda x.P'''$.

Case 5 : $M \gg_\delta M_1$ is $M \equiv \Delta x.P \gg_\delta \Delta x.P' \equiv M_1$ with $P \gg_\delta P'$. Here it must be that $M_2 \equiv \Delta x.P''$ with $P \gg_\delta P''$. By induction hypothesis and rule (5) of $\gg_\delta$ there is a term P''' such that we can choose $M_3 \equiv \Delta x.P'''$. □

Proof of Lemma 4

First we note that the reduction relation $\gg_\gamma^*$ is substitutive:

LEMMA **20** *For any $M, N \in \Lambda$: if $M \gg_\gamma^* M'$ then $M\{x := N\} \gg_\gamma^* M'\{x := N\}$.*

PROOF: By induction on the number n of $M \gg_\gamma^n M'$, $n \geq 0$. For $n = 0$ the claim is obvious. Suppose that $M \gg_\gamma^{n+1} M'$ because $M \gg_\gamma^n M'' \gg_\gamma M'$. Then, by induction, $M\{x := N\} \gg_\gamma^* M''\{x := N\}$. Since $\gg_\gamma$ is substitutive by Lemma 18, it follows that $M''\{x := N\} \gg_\gamma M'\{x := N\}$. Now the claim follows by transitivity of $\gg_\gamma^*$. □

Since $\gg_\gamma^*$ is compatible, it follows by Lemma 20 that $M\{x := N\} \gg_\gamma^* M'\{x := N'\}$ whenever $M \gg_\gamma^* M'$ and $N \gg_\gamma^* N'$. Now we can prove:

LEMMA 4 *For all $M_1, M_2, M_3 \in \Lambda$: if $M_1 \gg_\gamma M_2$ and $M_1 \gg_\delta M_3$ then there is an $M_4 \in \Lambda$ such that $M_2 \gg_\delta M_4$ and $M_3 \gg_\gamma^* M_4$.*
PROOF: The proof is by induction on the definition of $M_1 \gg_\gamma M_2$.

Case 1 : $M_1 \gg_\gamma M_2$ is $M_1 \gg_\gamma M_1 \equiv M_2$. Choose $M_4 \equiv M_3$. Analogously, the claim holds in all cases where $M_1 \gg_\delta M_3 \equiv M_1$. Such cases are disregarded in what follows.

Case 2 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv (\lambda x.P)Q \gg_\gamma P'\{x := Q'\} \equiv M_2$ with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$. Here it must be the case that $M_1 \gg_\delta M_3$ is $M_1 \equiv (\lambda x.P)\,Q \gg_\delta (\lambda x.P'')\,Q'' \equiv M_3$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis there are terms P''', Q''' such that $P' \gg_\delta P'''$, $P'' \gg_\gamma^* P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\gamma^* Q'''$. Choose $M_4 \equiv P'''\{x := Q'''\}$. Then $M_2 \gg_\delta M_4$ by Lemma 19. From $P'' \gg_\gamma^* P'''$ and $Q'' \gg_\gamma^* Q'''$ we get $M_3 \equiv (\lambda x.P'')Q'' \gg_\gamma^* (\lambda x.P''')Q''' \gg_\gamma P'''\{x := Q'''\} \equiv M_4$ by compatibility of $\gg_\gamma^*$ and rule (2) of $\gg_\gamma$.

Case 3 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv \Delta x.x\,P \gg_\gamma P' \equiv M_2$ with $x \notin FV(P)$ and $P \gg_\gamma P'$. Here it must be the case that $M_1 \gg_\delta M_3$ is $\Delta x.xP \gg_\delta \Delta x.xP'' \equiv M_3$ with $P \gg_\delta P''$. By induction hypothesis there is a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv P'''$. Then $M_2 \gg_\delta M_4$ is already obtained. We have $M_3 \gg_\gamma P''$ by rule (3) of $\gg_\gamma$, since $x \notin FV(P)$ and $P \gg_\delta P''$ imply $x \notin FV(P'')$. Then, by $P'' \gg_\gamma^* P'''$, we get $M_3 \gg_\gamma^* P''' \equiv M_4$.

Case 4 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv \Delta x.x\,\nabla(x\,P) \gg_\gamma P' \equiv M_2$ with $x \notin FV(P)$ and $P \gg_\gamma P'$. Then it must be the case that $M_1 \gg_\delta M_3$ is $\Delta x.x\,\nabla(x\,P) \gg_\delta \Delta x.x\,P_1'' \equiv M_3$ with $\nabla(x\,P) \gg_\delta P_1''$ which in turn implies that $P_1'' \equiv \nabla(x\,P'')$ with $P \gg_\delta P''$. Then, by induction hypothesis, there is a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv P'''$. Then $M_2 \gg_\delta M_4$ is already obtained. We have $M_3 \gg_\gamma P''$ by rule (4) of $\gg_\gamma$, since $x \notin FV(P)$ and $P \gg_\delta P''$ imply $x \notin FV(P'')$. Then, by $P'' \gg_\gamma^* P'''$, we get $M_3 \gg_\gamma^* P''' \equiv M_4$.

Case 5 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv \nabla(P)\, Q \gg_\gamma \nabla(P') \equiv M_2$ with $P \gg_\gamma P'$. There are two subcases according to the form of $M_1 \gg_\delta M_3$:

Subcase 5.1 : $M_1 \gg_\delta M_3$ is $\nabla(P) \gg_\delta \nabla(P'') \equiv M_3$ with $P \gg_\delta P''$. Then, by induction hypothesis, there is a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv \nabla(P''')$. Then we have $M_3 \gg_\gamma^* M_4$ by compatibility, and also $M_2 \gg_\delta M_4$ by rule (5) of $\gg_\delta$.

Subcase 5.2 : $M_1 \gg_\delta M_3$ is $\nabla(P)\, Q \gg_\delta \nabla(P'')\, Q'' \equiv M_3$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis there is a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv \nabla(P''')$. Then $M_3 \gg_\gamma^* \nabla(P'')$ by rule (5) of $\gg_\gamma$, and $P'' \gg_\gamma^* P'''$ then yields $M_3 \gg_\gamma^* M_4$. $M_2 \gg_\delta M_4$ holds by rule (5) of $\gg_\delta$.

Case 6 : $M_1 \equiv P\, Q \gg_\gamma P'\, Q' \equiv M_2$ with $P \gg_\gamma P'$ and $Q \gg_\gamma Q'$.

Subcase 6.1 : $M_1 \gg_\delta M_3$ is $M_1 \equiv (\Delta x.P_1) Q \gg_\delta \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\, Q'')\} \equiv M_3$ with $P_1 \gg_\delta P_1'$ and $Q \gg_\delta Q''$. We consider three further subcases according to the form of P_1 and $P \gg_\gamma P'$:

Subcase 6.1.1 : $P_1 \equiv x\, \nabla(x\, P_2)$ with $x \notin FV(P_2)$ where $P \gg_\gamma P'$ is $P \equiv \Delta x.x\, \nabla(x\, P_2) \gg_\gamma P_2' \equiv P'$ with $P_2 \gg_\gamma P_2'$. Now, in this case $P_1 \gg_\delta P_1'$ must be $P_1 \equiv x\, \nabla(x\, P_2) \gg_\delta x\, \nabla(x\, P_2'') \equiv P_1'$ with $P_2 \gg_\delta P_2''$. By induction hypothesis there are terms P_2''' and Q''' such that $P_2' \gg_\delta P_2'''$, $P_2'' \gg_\gamma^* P_2'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\gamma^* Q'''$. Choose $M_4 \equiv P_2'''\, Q'''$. Then we have :

$$\begin{array}{lll}
M_3 & \equiv & \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\, Q'')\} \\
& \equiv & \Delta\kappa.(x\, \nabla(x\, P_2''))\{x := \lambda f.\kappa\,(f\, Q'')\} \\
& \equiv & \Delta\kappa.(\lambda f.\kappa\,(f\, Q''))\, \nabla((\lambda f.\kappa\,(f\, Q''))\, P_2'') \\
& \gg_\gamma^* & \Delta\kappa.\kappa\,(\nabla(\kappa\,(P_2''\, Q''))\, Q'') \\
& \gg_\gamma & \Delta\kappa.\kappa\, \nabla(\kappa\,(P_2''\, Q'')) \\
& \gg_\gamma & P_2''\, Q''
\end{array}$$

Since $P_2'' \gg_\gamma^* P_2'''$ and $Q'' \gg_\gamma^* Q'''$ it follows that $M_3 \gg_\gamma^* M_4$. Also, we have $M_2 \equiv P'\, Q' \equiv P_2'\, Q'$, and since $P_2' \gg_\delta P_2'''$ and $Q' \gg_\delta Q'''$ it follows that $M_2 \gg_\delta M_4$.

Subcase 6.1.2 : $P_1 \equiv x\, P_2$ and $P \gg_\gamma P'$ is $P \equiv \Delta x.x\, P_2 \gg_\gamma P_2' \equiv P'$ with $x \notin FV(P_2)$ and $P_2 \gg_\gamma P_2'$. Now, $P_1 \gg_\delta P_1'$ must be $P_1 \equiv x\, P_2 \gg_\delta x\, P_2'' \equiv P_1'$ with $P_2 \gg_\delta P_2''$. Therefore, by induction hypothesis, there are terms P_2''' and Q''' such that $P_2' \gg_\delta P_2'''$, $P_2'' \gg_\gamma^* P_2'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\gamma^* Q'''$. Choose $M_4 \equiv P_2'''\, Q'''$. Then

$$\begin{array}{lll}
M_3 & \equiv & \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\, Q'')\} \\
& \equiv & \Delta\kappa.(x\, P_2'')\{x := \lambda f.\kappa\,(f\, Q'')\} \\
& \equiv & \Delta\kappa.(\lambda f.\kappa\,(f\, Q''))\, P_2'' \\
& \gg_\gamma & \Delta\kappa.\kappa\,(P_2''\, Q'') \\
& \gg_\gamma & P_2''\, Q''
\end{array}$$

Since $P_2'' \gg_\gamma^* P_2'''$ and $Q'' \gg_\gamma^* Q'''$ it follows that $P_2''\, Q'' \gg_\gamma^* P_2'''\, Q'''$, and consequently $M_3 \gg_\gamma^* M_4$. Also, we have $M_2 \equiv P'\, Q' \equiv P_2'\, Q'$. Since $P_2' \gg_\delta P_2'''$

and $Q' \gg_\delta Q'''$ it follows that $M_2 \gg_\delta M_4$.

Subcase 6.1.3 Suppose $P \not\equiv \Delta x.x\,P_2$ and $P \not\equiv \Delta x.x\,\nabla(x\,P_2)$, $x \in FV(P_2)$. Then $M_1 \gg_\gamma M_2$ must be $M_1 \equiv (\Delta x.P_1)Q \gg_\gamma (\Delta x.P_2')Q' \equiv M_2$ with $P_1 \gg_\gamma P_2'$ and $Q \gg_\gamma Q'$. By induction hypothesis there are terms P''' and Q''' such that $P_2' \gg_\delta P'''$, $P_1' \gg_\gamma^* P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\gamma^* Q'''$. Choose $M_4 \equiv \Delta\kappa.P'''\{x := \lambda f.\kappa\,(f\,Q''')\}$. Then we have $M_2 \equiv (\Delta x.P_2')\,Q' \gg_\delta M_4$ by rule (2) of $\gg_\delta$. And $M_3 \equiv \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\,Q'')\} \gg_\gamma^* M_4$ since $P_1' \gg_\gamma^* P'''$, $Q'' \gg_\gamma^* Q'''$ and $\gg_\gamma^*$ is compatible and substitutive (Lemma 20.)

Subcase 6.2 : $M_1 \gg_\delta M_3$ is $P\,Q \gg_\delta P''\,Q'' \equiv M_3$ with $P \gg_\delta P''$ and $Q \gg_\delta Q''$. By induction hypothesis there are terms P''' and Q''' such that $P' \gg_\delta P'''$, $P'' \gg_\gamma^* P'''$, $Q' \gg_\delta Q'''$ and $Q'' \gg_\gamma^* Q'''$. Choose $M_4 \equiv P'''\,Q'''$. Then $M_3 \gg_\gamma^* M_4$ and $M_2 \gg_\delta M_4$ follow easily.

Case 7 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv \lambda x.P \gg_\gamma \lambda x.P' \equiv M_2$ with $P \gg_\gamma P'$. Here we must have that $M_1 \gg_\delta M_3$ is $\lambda x.P \gg_\delta \lambda x.P'' \equiv M_3$ with $P \gg_\delta P''$. By induction hypothesis we get a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv \lambda x.P'''$ and the desired result follows easily.

Case 8 : $M_1 \gg_\gamma M_2$ is $M_1 \equiv \Delta x.P \gg_\gamma \Delta x.P' \equiv M_2$ with $P \gg_\gamma P'$. Here it must be the case that $M_1 \gg_\delta M_3$ is $\Delta x.P \gg_\delta \Delta x.P'' \equiv M_3$ with $P \gg_\delta P''$. By induction hypothesis we get a term P''' such that $P' \gg_\delta P'''$ and $P'' \gg_\gamma^* P'''$. Choose $M_4 \equiv \Delta x.P'''$ and the desired result follows easily. □

Proof of lemma 15 (Postponement)

To get the Postponement lemma we must first note that $\rhd_{3,4}$ is substitutive:

LEMMA **21** *$M \rhd_{3,4} M'$ implies $M\{x := N\} \rhd_{3,4} M'\{x := N\}$ for any M, N.*

PROOF: Easy structural induction on M. □

Since $\rhd_{3,4}$ is compatible it follows that $M\{x := N\} \rhd_{3,4}^* M'\{x := N'\}$ whenever $M \rhd_{3,4}^* M'$ and $N \rhd_{3,4}^* N'$. Now we can prove

LEMMA 15 *If $M_1 \rhd_{3,4} M_2 \rhd_{1,2} M_3$ then there exists an M_4 such that $M_1 \rhd_{1,2}^+ M_4 \rhd_{3,4}^* M_3$*
PROOF: By structural induction on M_1.

$\boxed{M_1 \equiv x}$ Trivial.

$\boxed{M_1 \equiv \lambda x.P}$ Here $M_1 \rhd_{3,4} M_2 \rhd_{1,2} M_3$ must be $M_1 \equiv \lambda x.P \rhd_{3,4} \lambda x.P' \equiv M_2 \rhd_{1,2} \lambda x.P'' \equiv M_3$ with $P \rhd_{3,4} P' \rhd_{1,2} P''$. By induction there is a Q' such that $P \rhd_{1,2}^+ Q' \rhd_{3,4}^* P''$. Choose $M_4 \equiv \lambda x.Q'$.

$\boxed{M_1 \equiv P_1\ P_2}$ We divide the analysis into cases according to the form of $M_1 \rhd_{3,4} M_2$ and into subcases according to $M_2 \rhd_{1,2} M_3$:

Case 1 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv P_1\ P_2 \rhd_{3,4} P_1'\ P_2 \equiv M_2$ with $P_1 \rhd_{3,4} P_1'$

Subcase 1.1 : $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv (\lambda x.Q_1')\ P_2 \rhd_1 Q_1'\{x := P_2\} \equiv M_3$ with $P_1' \equiv \lambda x.Q_1'$. This implies $P_1 \equiv \lambda x.Q_1$ with $Q_1 \rhd_{3,4} Q_1'$. Choose $M_4 \equiv Q_1\{x := P_2\}$. Then $M_1 \equiv P_1 P_2 \equiv (\lambda x.Q_1) P_2 \rhd_1 M_4$, and $M_4 \rhd_{3,4} M_3$ follows from $Q_1 \rhd_{3,4} Q_1'$ by substitutivity of $\rhd_{3,4}$.

Subcase 1.2: $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv (\Delta x.Q_1') P_2 \rhd_2 \Delta\kappa.Q_1'\{x := \lambda f.\kappa\ (f\ P_2)\} \equiv M_3$ with $P_1' \equiv \Delta x.Q_1'$. We consider three situations :

Subcase 1.2.1 : $P_1 \rhd_{3,4} P_1'$ is $P_1 \equiv \Delta y.y\ \Delta x.Q_1' \rhd_3 \Delta x.Q_1' \equiv P_1'$ with $y \notin FV(Q_1')$. Put $M_4 \equiv \Delta\kappa'.\kappa'\ (\Delta\kappa.Q_1'\{x := \lambda f.\kappa\ (f\ P_2)\})$. Then $M_4 \rhd_3 M_3$, and

$$\begin{array}{lll}
M_1 & \equiv & (\Delta y.y\ \Delta x.Q_1')\ P_2 \\
 & \rhd_2 & \Delta\kappa'.(y\ \Delta x.Q_1')\{y := \lambda f'.\kappa'\ (f'\ P_2)\} \\
 & \equiv & \Delta\kappa'.(\lambda f'.\kappa'\ (f'\ P_2)\ \Delta x.Q_1') \\
 & \rhd_1 & \Delta\kappa'.\kappa'\ ((\Delta x.Q_1')\ P_2) \\
 & \rhd_2 & \Delta\kappa'.\kappa'\ (\Delta\kappa.Q_1'\{x := \lambda f.\kappa\ (f\ P_2)\}) \\
 & \equiv & M_4
\end{array}$$

Subcase 1.2.2 : $P_1 \rhd_{3,4} P_1'$ is $P_1 \equiv \Delta y.y\ \nabla(y\ Dx.Q_1') \rhd_3 \Delta x.Q_1' \equiv P_1'$ with $y \notin FV(Q_1')$. Put $M_4 \equiv \Delta\kappa'.\kappa'\ \nabla(\kappa'\ \Delta\kappa.Q_1'\{x := \lambda f.\kappa\ (f\ P_2)\})$. Then $M_4 \rhd_4 M_3$. Let $S^{P_2}_{\kappa'} \equiv \lambda f'.\kappa'\ (f'\ P_2)$. Then

$$\begin{array}{lll}
M_1 & \equiv & (\Delta y.y\ \nabla(y\ \Delta x.Q_1'))\ P_2 \\
 & \rhd_2 & \Delta\kappa'.(y\ \nabla(y\ \Delta x.Q_1'))\{y := S^{P_2}_{\kappa'}\} \\
 & \equiv & \Delta\kappa'.S^{P_2}_{\kappa'}\ \nabla(S^{P_2}_{\kappa'}\ \Delta x.Q_1') \\
 & \rhd_1^* & \Delta\kappa'.\kappa'\ (\nabla(\kappa'\ ((\Delta x.Q_1')\ P_2))\ P_2) \\
 & \rhd_2 & \Delta\kappa'.\kappa'\ \nabla(\kappa'\ ((\Delta x.Q_1')\ P_2)) \\
 & \rhd_2 & \Delta\kappa'.\kappa'\ \nabla(\kappa'\ \Delta\kappa.Q_1'\{x := \lambda f.\kappa\ (f\ P_2)\}) \\
 & \equiv & M_4
\end{array}$$

Subcase 1.2.3: $P_1 \rhd_{3,4} P_1'$ is $P_1 \equiv \Delta x.Q_1 \rhd_{3,4} \Delta x.Q_1' \equiv P_1'$ with $Q_1 \rhd_{3,4} Q_1'$. Choose $M_4 \equiv \Delta\kappa.Q_1\{x := \lambda f.\kappa\ (f\ P_2)\}$. Then $M_1 \equiv (\Delta x.Q_1)\ P_2 \rhd_2 M_4$, and $M_4 \rhd_{3,4}^* M_3$ by substitutivity of $\rhd_{3,4}$.

Subcase 1.3 : $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv P_1'\ P_2 \rhd_{1,2} P_1''\ P_2 \equiv M_3$ with $P_1' \rhd_{1,2} P_2''$. Since we now have $P_1 \rhd_{3,4} P_1' P_1''$ it follows by induction hypothesis that there is a term P such that $P_1 \rhd_{1,2}^+ P \rhd_{3,4}^* P_1''$. Choose $M_4 \equiv P\ P_2$. Then $M_1 \rhd_{1,2}^+ M_4 \rhd_{3,4}^* M_3$.

Subcase 1.4 : $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv P_1'\ P_2 \rhd_{1,2} P_1'\ P_2' \equiv M_3$ with $P_2 \rhd_{1,2} P_2'$. Choose $M_4 \equiv P_1\ P_2'$. Then $M_1 \rhd_{1,2} M_4 \rhd_{3,4} M_3$.

Case 2 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv P_1\ P_2 \rhd_{3,4} P_1\ P_2' \equiv M_2$ with $P_2 \rhd_{3,4} P_2'$.

Subcase 2.1 : $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv (\lambda x.Q)\, P_2' \rhd_1 Q\{x := P_2'\} \equiv M_3$. Choose $M_4 \equiv Q\{x := P_2\}$. Then $M_1 \rhd_1 M_4 \rhd_{3,4} M_3$ by compatibility of $\rhd_{3,4}$.

Subcase 2.2 : $M_2 \rhd_{1,2} M_3$ is $M_2 \equiv (\Delta x.Q)\, P_2' \rhd_2 \Delta\kappa.Q\{x := \lambda f.\kappa\, (f\ P_2')\} \equiv M_3$ with $P_1 \equiv \Delta x.Q$. Choose $M_4 \equiv \Delta\kappa.Q\{x := \lambda f.\kappa\, (f\ P_2)\}$. Then $M_1 \rhd_2 M_4 \rhd_{3,4}^* M_3$ by compatibility of $\rhd_{3,4}$.

Subcases 2.3 and 2.4 : $M_2 \rhd_{1,2} M_3$ is either $M_2 \equiv P_1\ P_1' \rhd_{3,4} P_1'\ P_2' \equiv M_3$ with $P_1 \rhd_{3,4} P_1'$, or it is $M_2 \equiv P_1\ P_2' \rhd_{3,4} P_1\ P_2'' \equiv M_3$ with $P_2' \rhd_{3,4} P_2''$. In the first case we choose $M_4 \equiv P_1'\ P_2$, and in the second case we choose $M_4 \equiv P_1\ P$ where P is given by induction hypothesis, as in subcase 1.3.

$\boxed{M_1 \equiv \Delta x.P}$ *Case 1* : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv \Delta x.x\ P_1 \rhd_3 P_1 \equiv M_2$ with $x \notin FV(P_1)$ and $P \equiv x\ P_1$. Here we must have $M_2 \equiv P_1 \rhd_{1,2} P_1' \equiv M_3$. Put $M_4 \equiv \Delta x.x\ P_1'$. Since $x \notin FV(P_1)$ and $P_1 \rhd_{1,2} P_1'$ imply $x \notin FV(P_1')$ we get $M_1 \rhd_{1,2} M_4 \rhd_3 M_3$.

Case 2 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv \Delta x.x\, \nabla(x\ P_1) \rhd_4 P_1 \equiv M_2$ with $x \notin FV(P_1)$ and $P \equiv x\ \nabla(x\ P_1)$. Here we must have $M_2 \equiv P_1 \rhd_{1,2} P_1' \equiv M_3$. Put $M_4 \equiv \Delta x.x\, \nabla(x\ P_1')$. Since $x \notin FV(P_1)$ and $P_1 \rhd_{1,2} P_1'$ imply $x \notin FV(P_1')$ we get $M_1 \rhd_{1,2} M_4 \rhd_3 M_3$.

Case 3 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv \Delta x.P \rhd_{3,4} \Delta x.P' \equiv M_2$ with $P \rhd_{3,4} P'$. Now, $M_2 \rhd_{1,2} M_3$ must be $M_2 \equiv \Delta x.P' \rhd_{1,2} \Delta x.P'' \equiv M_3$ with $P' \rhd_{1,2} P''$. Then we have $P \rhd_{3,4} P' \rhd_{1,2} P''$. By induction hypothesis there is a P''' such that $P \rhd_{1,2}^+ P''' \rhd_{3,4}^* P''$. Choose $M_4 \equiv \Delta x.P'''$, then the claim follows easily.

$\square$

B Proofs for section 3.2

Proof of lemma 9 of section 3.2 was left out there. The proof is given below.

Proof of Lemma 9

First we must observe that we have the usual substitution properties:

LEMMA **22** *For any $M, N \in \Lambda_\delta$: If $M \gg_{\phi\Delta} M'$ and $N \gg_{\phi\Delta} N'$ then $M\{x := N\} \gg_{\phi\Delta} M'\{x := N'\}$.*

PROOF: Analogous to proof of Lemma 18 of Appendix A. □

LEMMA **23** *For any $M, N \in \Lambda_\delta$: If $M \gg^* M'$ and $N \gg^* N'$ then $M\{x := N\} \gg^* M'\{x := N'\}$.*

PROOF: By proof analogous to proof of Lemma 18 of Appendix A we show that the claim holds with $\gg$ instead of $\gg^*$. Then the claim itself follows from this by compatibility and transitivity of $\gg^*$. □

LEMMA 9 *For any $M_1, M_2, M_3 \in \Lambda_\delta$: If $M_1 \gg M_2$ and $M_1 \gg_{\phi\Delta} M_3$ then there exists an $M_4 \in \Lambda_\delta$ such that $M_3 \gg^* M_4$ and $M_2 \gg_{\phi\Delta} M_4$.*
PROOF: By induction on $M_1 \gg M_2$, and the cases where either $M_1 \equiv M_2$ or $M_1 \equiv M_3$ are trivial and left out.

Case 1 : $M_1 \gg M_2$ is $M_1 \equiv (\lambda x.P)\, Q \gg P'\{x := Q'\} \equiv M_2$ with $P \gg P'$ and $Q \gg Q'$. Then $M_1 \gg_{\phi\Delta} M_3$ must be by $BCP3$, and the result follows by induction hypothesis together with Lemma 22.

Case 2 : $M_1 \gg M_2$ is $M_1 \equiv (\Delta x.P)\, Q \gg \Delta\kappa.P'\{x := \lambda f.\kappa\,(f\, Q')\} \equiv M_2$ with $P \gg P'$ and $Q \gg Q'$. Then $M_1 \gg_{\phi\Delta} M_3$ must be by $BCP2$, and the result follows by induction hypothesis together with Lemma 22.

Case 3 : $M_1 \gg M_2$ is $M_1 \equiv \Delta x.x\; P \gg P' \equiv M_2$ with $P \gg P'$ and $x \notin FV(P)$. Then $M_1 \gg_{\phi\Delta} M_3$ must be by $BCP4$, and the result follows from the induction hypothesis together with the fact that $\gg^*$ does not introduce free variables.

Case 4 : $M_1 \gg M_2$ is $M_1 \equiv \Delta x.x\, \nabla(x\; P) \gg P' \equiv M_2$ with $P \gg P'$ and $x \notin FV(P)$. Again, $M_1 \gg_{\phi\Delta} M_3$ must be by $BCP4$, and the case is analogous to the previous one.

Case 5 : $M_1 \gg M_2$ is $M_1 \equiv \phi\, \nabla(P) \gg \nabla(P') \equiv M_2$ with $P \gg P'$. Then one of the following two subcases must obtain:

Subcase 5.1 : $M_1 \gg_{\phi\Delta} M_3$ is $M_1 \equiv \phi\, \nabla(P) \gg_{\phi\Delta} \Delta\kappa.P''\{d := \lambda f.\kappa\,(\phi\, f)\} \equiv M_3$ where $\nabla(P) \equiv \Delta d.P$, $d \notin FV(P)$, and $P \gg_{\phi\Delta} P''$. It follows that $M_3 \equiv \Delta\kappa.P'' \equiv \nabla(P'')$. The result follows by induction hypothesis yielding P''' such that we can take $M_4 \equiv \nabla(P''')$.

Subcase 5.2 : $M_1 \gg_{\phi\Delta} M_3$ is $M_1 \equiv \phi\,\nabla(P) \gg_{\phi\Delta} \phi\,\nabla(P'') \equiv M_3$ with $P \gg_{\phi\Delta} P''$. By induction hypothesis we get P''' such that we can take $M_4 \equiv \nabla(P''')$.

Case 6 : $M_1 \gg M_2$ is $M_1 \equiv P\,Q \gg P'\,Q' \equiv M_2$ with $P \gg P'$, $Q \gg Q'$. Then we have two main possibilities:

Subcase 6.1 : $M_1 \gg_{\phi\Delta} M_3$ is $M_1 \equiv P\,Q \equiv \phi\,\Delta x.Q_1 \gg_{\phi\Delta} \Delta\kappa.Q_1'\{x := \lambda f.\kappa\,(\phi\,f)\} \equiv M_3$ with $P \equiv \phi$, $Q \equiv \Delta x.Q_1$, $Q_1 \gg_{\phi\Delta} Q_1'$. We also must have $P' \equiv \phi$. There are three further possibilities:

Subcase 6.1.1 : $Q \gg Q'$ is $Q \equiv \Delta x.Q_1 \gg \Delta x.Q_2' \equiv Q'$ with $Q_1 \gg Q_2'$. By induction we get Q''' such that $Q_1' \gg^* Q'''$ and $Q_2' \gg_{\phi\Delta} Q'''$. Choose $M_4 \equiv \Delta\kappa.Q'''\{x := \lambda f.\kappa\,(\phi\,f)\}$. Then we have $M_2 \equiv \phi\,\Delta x.Q_2' \gg_{\phi\Delta} M_4$ and also, by substitutivity of $\gg^*$, $M_3 \gg^* M_4$.

Subcase 6.1.2 : $Q \gg Q'$ is $Q \equiv \Delta x.x\,Q_2 \gg Q_2' \equiv Q'$ with $x \notin FV(Q_2)$ and $Q_2 \gg Q_2'$. In this case we have $Q_1 \equiv x\,Q_2$, and $Q_1 \gg_{\phi\Delta} Q_1'$ must be $x\,Q_2 \gg_{\phi\Delta} x\,Q_2''$ with $Q_2 \gg_{\phi\Delta} Q_2''$. To sum up, we have $M_1 \equiv \phi\,\Delta x.x\,Q_2 \gg \phi\,Q_2' \equiv M_2$ and $M_1 \gg_{\phi\Delta} \Delta\kappa.(x\,Q_2'')\{x := \lambda f.\kappa\,(\phi\,f)\} \equiv \Delta\kappa.(\lambda f.\kappa\,(\phi\,f))\,Q_2'' \equiv M_3$. Now, by induction we get Q''' such that $Q_2' \gg_{\phi\Delta} Q'''$ and $Q_2'' < prsQ'''$. Choose $M_4 \equiv \phi\,Q'''$. Then $M_2 \gg_{\phi\Delta} M_4$ is easily seen to hold, and also

$$\begin{array}{rl} M_3 & \equiv \;\; \Delta\kappa.(\lambda f.\kappa\,(\phi\,f))\,Q_2'' \\ & \gg^* \;\; \Delta\kappa.(\lambda f.\kappa\,(\phi\,f))\,Q''' \\ & \gg \;\; \Delta\kappa.\kappa\,(\phi\,Q''') \\ & \gg \;\; \phi\,Q''' \\ & \equiv \;\; M_4 \end{array}$$

Subcase 6.1.3 : $Q \gg Q'$ is $Q \equiv \Delta x.x\,\nabla(x\,Q_2) \gg Q_2'$ with $x \notin FV(Q_2)$ and $Q_2 \gg Q_2'$. In this case $Q_1 \equiv \nabla(x\,Q_2)$ and $Q_1 \gg_{\phi\Delta} Q_1'$ must be $x\,\nabla(x\,Q_2) \gg_{\phi\Delta} x\nabla(xQ_2'')$ with $Q_2 \gg_{\phi\Delta} Q_2''$. To sum up, we have $M_1 \equiv \phi\Delta x.x\nabla(xQ_2) \gg \phi Q_2' \equiv M_2$ and $M_1 \gg_{\phi\Delta} \Delta\kappa.(x\nabla(xQ_2''))\{x := \lambda f.\kappa\,(\phi\,f)\} \equiv \Delta\kappa.(\lambda f.\kappa(\phi f))\nabla((\lambda f.\kappa(\phi f))Q_2'') \equiv M_3$. By induction we get Q''' such that $Q_2' \gg_{\phi\Delta} Q'''$ and $Q_2'' \gg^* Q'''$. Choose $M_4 \equiv \phi\,Q'''$. Then $M_2 \gg_{\phi\Delta} M_4$ is easily seen to hold, and also

$$\begin{array}{rl} M_3 & \equiv \;\; \Delta\kappa.(\lambda f.\kappa\,(\phi\,f))\,\nabla((\lambda f.\kappa\,(\phi\,f))\,Q_2'') \\ & \gg^* \;\; \Delta\kappa.(\lambda f.\kappa\,(\phi\,f))\,\nabla((\lambda f.\kappa\,(\phi\,f))\,Q''') \\ & \gg^* \;\; \Delta\kappa.\kappa(\phi\,\nabla(\kappa\,(\phi\,Q'''))) \\ & \gg \;\; \Delta\kappa.\nabla(\kappa\,(\phi\,Q''')) \\ & \gg \;\; \phi\,Q''' \\ & \equiv \;\; M_4 \end{array}$$

where we used rule (5) of $\gg$ at the third reduction step.

Subcase 6.2: $M_1 \gg_{\phi\Delta} M_3$ is $M_1 \equiv P\,Q \gg_{\phi\Delta} P''\,Q'' \equiv M_3$ with $P \gg_{\phi\Delta} P''$ and $Q \gg_{\phi\Delta} Q''$. By induction we get P''' and Q''' such that we can choose $M_4 \equiv P'''\,Q'''$.

Case 7 : $M_1 \gg M_2$ is $M_1 \equiv \lambda x.P \gg \lambda x.P' \equiv M_2$ with $P \gg P'$. Only the corresponding rule $BCP3$ for $\gg_{\phi\Delta}$ is applicable, and the result follows easily by induction.

Case 8 : $M_1 \gg M_2$ is $M_1 \equiv \Delta x.P \gg \Delta x.P' \equiv M_2$ with $P \gg P'$. Only the corresponding rule $BCP4$ for $\gg_{\phi\Delta}$ is applicable, and the result follows easily by induction. $\square$

C Proof of theorem 6, section 4

Proof of Theorem 6 was left out. The proof is given below.

LEMMA **24** *If $M_1 \rhd M_2$ and $M_1 \rhd_8 M_3$ then there is an M_4 such that $M_2 \rhd_8^* M_4$ and $M_3 \rhd M_4$.*

PROOF: Structural induction on M_1.

$\boxed{M_1 \equiv x, b, \phi}$: Claim trivially true.

$\boxed{M_1 \equiv \lambda x.P}$: $M_1 \rhd M_2$ must be $\lambda x.P \rhd \lambda x.P' \equiv M_2$ with $P \rhd P'$, and $M_1 \rhd_8 M_3$ must be $\lambda x.P \rhd_8 \lambda x.P''$ with $P \rhd_8 P''$. The claim follows by induction hypothesis.

$\boxed{M_1 \equiv P\,Q}$:

Case 1 : $M_1 \equiv (\lambda x.P_1)\,Q \rhd P_1\{x := Q\} \equiv M_2$. There are two possibilities :

Subcase 1.1 : $M_1 \rhd_8 M_3$ is $M_1 \equiv (\lambda x.P_1)Q \rhd_8 (\lambda x.P_1')Q \equiv M_3$ with $P_1 \rhd_8 P_1'$. Choose $M_4 \equiv P_1'\{x := Q\}$. Then the claim follows by substitutivity of $\rhd_8$.

Subcase 1.2 : $M_1 \rhd_8 M_3$ is $M_1 \equiv (\lambda x.P_1)Q \rhd_8 (\lambda x.P_1)Q' \equiv M_3$ with $Q \rhd_8 Q'$. Choose $M_4 \equiv P_1\{x := Q'\}$, and we have $M_3 \rhd M_4$ and $M_2 \rhd_8^* M_4$.

Case 2 : $M_1 \equiv (\Delta x.P_1)\,Q \rhd \Delta\kappa.P_1\{x := \lambda f.\kappa\,(f\,Q)\} \equiv M_2$.

Subcase 2.1 : $P_1 \equiv \nabla(P_2)$ and $M_1 \equiv \Delta x.\nabla(P_2)\,Q \rhd_8 (\Delta x.P_2)\,Q \equiv M_3$. Choose $M_4 \equiv \Delta\kappa.P_2\{x := \lambda f.\kappa\,(f\,Q)\}$. Then $M_3 \rhd M_4$, and $M_2 \equiv \Delta\kappa.\nabla(P_2\{x := \lambda f.\kappa\,(f\,Q)\}) \rhd_8 M_4$.

Subcase 2.2 : $M_1 \rhd_8 M_3$ is either $(\Delta x.P_1)\,Q \rhd_8 (\Delta x.P_1')\,Q \equiv M_3$ with $P_1 \rhd_8 P_1'$, or it is $(\Delta x.P)\,Q \rhd_8 (\Delta x.P_1)\,Q' \equiv M_3$ with $Q \rhd_8 Q'$. In the first case we choose $M_4 \equiv \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\,Q)\}$, and in the second case we choose $M_4 \equiv \Delta\kappa.P_1\{x := \lambda f.\kappa\,(f\,Q')\}$.

Case 3 : $M_1 \equiv \phi\,\Delta x.Q_1 \rhd \Delta\kappa.Q_1\{x := \lambda f.\kappa\,(\phi\,f)\} \equiv M_2$.

Subcase 3.1 $Q_1 \equiv \nabla(Q_2)$ and $M_1 \rhd_8 M_3$ is $\phi\,\Delta x.\nabla(Q_2) \rhd_8 \phi\,\Delta x.Q_2 \equiv M_3$. Choose $M_4 \equiv \Delta\kappa.Q_2\{x := \lambda f.\kappa\,(\phi\,f)\}$

Subcase 3.2 : $M_1 \rhd_8 M_3$ is $\phi\,\Delta x.Q_1 \rhd_8 \phi\,\Delta x.Q_1' \equiv M_3$ with $Q_1 \rhd_8 Q_1'$. Choose $M_4 \equiv \Delta\kappa.Q_1'\{x := \lambda f.\kappa\,(\phi\,f)\}$.

Case 4 : $M_1 \rhd M_2$ is either $PQ \rhd P'Q \equiv M_2$ with $P \rhd P'$ or $PQ \rhd PQ' \equiv M_2$ with $Q \rhd Q'$. Also $M_1 \rhd_8 M_3$ must be either $P\,Q \rhd_8 P''\,Q$ with $P \rhd_8 P''$ or $P\,Q \rhd_8 P\,Q''$ with $Q \rhd_8 Q''$. The claim follows either by induction hypothesis or we can reduce each side of the application independently.

$\boxed{M_1 \equiv \Delta x.P}$:

Case 1 : $M_1 \rhd M_2$ is $M_1 \equiv \Delta x.x\, P_1 \rhd P_1 \equiv M_2$. Here $M_1 \rhd_8 M_3$ must be $\Delta x.x\, P_1 \rhd_8 \Delta x.x\, P_1' \equiv M_3$ with $P_1 \rhd_8 P_1'$. Choose $M_4 \equiv P_1'$.

Case 2 : $M_1 \rhd M_2$ is $M_1 \equiv \Delta x.x \nabla(x P_1) \rhd P_1 \equiv M_2$. This case is analogous to the previous one.

Case 3 : $M_1 \rhd M_2$ is $M_1 \equiv \Delta x.P \rhd \Delta x.P' \equiv M_2$ with $P \rhd P'$.

Subcase 3.1 : $M_1 \rhd_8 M_3$ is $M_1 \equiv \Delta x.\nabla(P_1) \rhd_8 \Delta x.P_1 \equiv M_3$. Here we must have $P \equiv \nabla(P_1) \rhd \nabla(P_1') \equiv P'$ with $P_1 \rhd P_1'$. Choose $M_4 \equiv \Delta x.P_1'$.

Subcase 3.2 : $M_1 \rhd_8 M_3$ is $M_1 \equiv \Delta x.P \rhd_8 \Delta x.P'' \equiv M_3$ with $P \rhd_8 P''$. The claim follows by induction hypothesis.

□

LEMMA **25** $\rhd_8$ *is* CR.

PROOF: $\rhd_8$ is clearly SN and locally confluent. □

THEOREM 6 $\unrhd$ *is* CR. *In particular,* λ_Δ *is incomplete.*
PROOF: By Lemma 25 $\rhd_8$ is CR, and we already know that $\rhd$ is CR, by Theorem 2. By Lemma 24 and the commuting lemma, Lemma 5, $\rhd^*$ and $\rhd_8^*$ commute. Therefore, by the Hindley-Rosen Lemma, $\rhd \cup \rhd_8 = \unrhd$ is CR. □

D Towards Standardization, Evaluation and Correspondence

In this appendix we take first steps towards a machine-calculus correspondence in the style of Plotkin [Plo75] and Felleisen [Fel87a]. The first subsection contains a proof of a standardization theorem for λ_Δ. For the question of correspondence we do not intend to define a machine semantics in the form of an abstract machine, such as the SECD machine studied by Plotkin or CEK machines studied by Felleisen, [Fel87b]. Rather, we would be content to have a computable evaluation function, preferably with a specification sufficiently algorithmic to make it clear how it could be computed. However, so far we have been unable to prove a satisfactory correspondence theorem for the evaluation functions we have considered. Therefore, in the second subsection, the reader will find only a discussion of the correspondence problem and some hints at possible ways for further work.

Longer proofs are placed in the final section.

Standardization

The overall method is an adaptation of Felleisen's adaptation (in [Fel87a]) of Plotkin's method (in [Plo75].) First we define standard reduction functions which are building blocks of standard reduction sequences. We aim for the standardization theorem, saying that for any M, N $M \rhd^* N$ if and only if there is a standard reduction sequence from M to N. The usefulness of a standardization theorem is proportional to the simplicity of standard reduction sequences. By such sequences we seek do define particularly simple, yet complete, ways of performing reductions of the calculus. Therefore, the sequences will contain an element of determinism not explicitly present in the calculus but represented by standard reduction functions. As Felleisen notes ([Fel89]), standard reduction sequences can be viewed as a form of compatible closure of the standard reduction.

Standard Reduction Functions

below we define two reduction functions, called $\rhd_{ab}$ and $\rhd_{cd}$. They are deterministic versions of $\rhd_{1,2,6,7}$-reduction and $\rhd_{3,4}$-reduction, respectively. The motivation for this two-way split is given in the section on standard reduction sequences below.

Let $\rightarrow_a = \rightarrow_1 \cup \rightarrow_2 \cup \rightarrow_6 \cup \rightarrow_7$, so $\rightarrow_a$ covers all reductions excluding the Δ-elimination rules (4) and (5). Let χ range over variables, basic constants and basic functions, and let x range over variables.

DEFINITION **31** ($\rhd_a$-reduction)

$$\begin{array}{lllll} (a1) & M \rightarrow_a N & \Rightarrow & M \rhd_a N \\ (a2) & M \rhd_a M' & \Rightarrow & M\,N \rhd_a M'\,N \\ (a3) & N \rhd_a N' & \Rightarrow & \chi\,N \rhd_a \chi\,N' \end{array}$$

□

Definition **32** ($\rhd_b$-reduction)

$$\begin{array}{lllll} (b1) & M \rhd_a M' & \Rightarrow & \Delta x.M \rhd_b \Delta x.M' \\ (b2) & M \rhd_b M' & \Rightarrow & \Delta x.M \rhd_b \Delta x.M' \\ (b3) & M \rhd_b M' & \Rightarrow & x\,M \rhd_b x\,M' \end{array}$$

□

Define $\rhd_{ab} = \rhd_a \cup \rhd_b$. Also, let $\rightarrow_c = \rightarrow_3 \cup \rightarrow_4$, and define $\rhd_c$-reduction as:

Definition **33** ($\rhd_c$-reduction)

$$\begin{array}{lllll} (c1) & M \rightarrow_c M' & \Rightarrow & M \rhd_c M' \\ (c2) & M \rhd_c M' & \Rightarrow & M\,N \rhd_c M'\,N \\ (c3) & N \rhd_c N' & \Rightarrow & \chi\,N \rhd_c \chi\,N' \end{array}$$

□

Definition **34** ($\rhd_d$-reduction)

$$\begin{array}{llllll} (d1) & M \rightarrow_c M' & \Rightarrow & \Delta x.M \rhd_d \Delta x.M' & , & \text{provided } \Delta x.M \text{ is not trivial} \\ (d2) & M \rhd_d M' & \Rightarrow & \Delta x.M \rhd_d \Delta x.M' & , & \text{provided } \Delta x.M \text{ is not trivial} \\ (d3) & N \rhd_d N' & \Rightarrow & x\,N \rhd_d x\,N' & & \end{array}$$

□

Define $\rhd_{cd} = \rhd_c \cup \rhd_d$. We wish to establish that $\rhd_{ab}$ and $\rhd_{cd}$ are functions, *i.e.* , deterministic one step reductions. We go by some simple lemmas:

Lemma **26** *If $M \rhd_a N_1$ and $M \rhd_a N_2$ then $N_1 \equiv N_2$.*

Proof: If M is not an application then the claim is trivially true. So suppose M is an application.

If $M \rhd_a N_1$ by $(a1)$ then M is an $\rightarrow_a$-redex, and by case analysis it is easily seen that $M \rightarrow_a N_2$ is the only possibility, yielding $N_1 \equiv N_2$.

If $M \rhd_a N_1$ is $M \equiv P\,Q \rhd_a P'\,Q \equiv N_1$ by $(a2)$, with $P \rhd_a P'$, then P must be an application, hence neither $(a1)$ nor $(a3)$ is applicable to M. Therefore we must also have $M \rhd_a P''\,Q \equiv N_2$ with $P \rhd_a P''$. By induction, $P' \equiv P''$, hence $N_1 \equiv N_2$.

If $M \equiv \chi\,Q \rhd_a \chi\,Q' \equiv N_1$ by $(a3)$, with $Q \rhd_a Q'$, we see that Q must be an application and so $(a3)$ is the only rule applicable to M, and the result follows by induction as before. □

LEMMA **27** *If $M \rhd_b N$ then $M \equiv x^n\, \Delta x.P$, for an $n \geq 0$, where $x^0\, M \equiv M$ and $x^{n+1}\, M \equiv x(x^n\, M)$.*

PROOF: By cases over the last rule used in deriving $M \rhd_b N$, and for $(b1)$ and $(b2)$ the claim is obvious (taking $n = 0$.) For $(b3)$ we must have $M \equiv x\ P$ with $P \rhd_b P'$, so by induction, $P \equiv x^n\, \Delta x.P''$, hence $M \equiv x^{n+1}\, \Delta x.P''$. □

LEMMA **28** *If $M \rhd_b N$ then there is no N' such that $M \rhd_a N'$.*

PROOF: From $M \rhd_b N$ we get, by Lemma 27, that there is an $n \geq 0$ such that $M \equiv x^n\, \Delta x.P$, and we prove that M cannot be reduced by $\rhd_a$, by induction on n. For $n = 0$ the claim is obvious, and for $M \equiv x(x^{n-1}\, \Delta x.P)$ we see that $M \rhd_a N'$ would have to be by $(a3)$, so that $x^{n-1}\, \Delta x.P \rhd_a N''$ for some N'', contradicting the induction hypothesis. □

LEMMA **29** *If $M \rhd_b N_1$ and $M \rhd_b N_2$ then $N_1 \equiv N_2$.*

PROOF: Structural induction on M, by cases over last rule used in derivation of $M \rhd_b N_1$, and the claim is trivially true unless M is an application or a Δ-abstraction:

$\boxed{M \equiv \Delta x.P}$: Here $M \rhd_b N_1$ must be either by $(b1)$ or by $(b2)$.

Case 1 : Suppose $M \rhd_b N_1$ by $(b1)$, so $M \equiv \Delta x.P \rhd_b \Delta x.P' \equiv N_1$ with $P \rhd_a P'$. Since $P \rhd_a P'$, P cannot be a Δ-abstraction, hence $M \rhd_b N_2$ cannot be by $(b2)$. Also, it cannot be by $(b3)$ since M is not of the form xM'. Therefore, only $(b1)$ can be chosen for $M \rhd_b N_2$, so that $N_2 \equiv \Delta x.P''$ with $P \rhd_a P''$. But then, by Lemma 26, $P' \equiv P''$, hence $N_1 \equiv N_2$.

Case 2 : Suppose $M \rhd_b N_1$ by $(b2)$, so $M \equiv \Delta x.P \rhd_b \Delta x.P' \equiv N_1$ with $P \rhd_b P'$. Now, $M \rhd_b N_2$ cannot be by $(b1)$ because this would entail $P \rhd_a P''$ which is impossible by Lemma 28, since $P \rhd_b P'$. Suppose, then, that $M \rhd_b N_2$ by $(b2)$, so we have $M \equiv \Delta x.P \rhd_b \Delta x.P'' \equiv N_2$ with $P \rhd_b P''$. But then, by induction, $P' \equiv P''$, hence $N_1 \equiv N_2$.

$\boxed{M \equiv x\ P}$: Here we must have $M \equiv x\ P \rhd_b x\ P' \equiv N_1$ with $P \rhd_b P'$ and $N_2 \equiv x\ P''$ with $P \rhd_b P''$, and the claim follows by induction.

□

LEMMA **30** *If $M \rhd_{ab} N_1$ and $M \rhd_{ab} N_2$ then $N_1 \equiv N_2$.*

PROOF: If either $M \rhd_b N_1$ and $M \rhd_b N_2$ or $M \rhd_a N_1$ and $M \rhd_a N_2$ then the claim follows from Lemma 29 and Lemma 26, respectively. No other cases are possible, by Lemma 28. □

This shows that $\rhd_{ab}$ is a reduction function. In close analogy to this proof we can also prove that $\rhd_{cd}$ is a function, but since the proof is similar to the one just given we shall leave it out:

LEMMA **31** *If $M \rhd_{cd} N_1$ and $M \rhd_{cd} N_2$ then $N_1 \equiv N_2$.*

PROOF: Left out. □

At this point we are ready to build standard reduction sequences.

Standard Reduction Sequences

The overall idea behind the following definition of standard reduction sequences is that $\rhd_{3,4}$-reductions can be postponed, as is proved below. Accordingly, we can establish seperate standard reduction sequences for $\rhd_{1,2,6,7}$-reductions and $\rhd_{3,4}$-reductions and obtain reduction sequences for arbitrary reductions by composition of the two standard reduction sequences.

DEFINITION **35** (Standard reduction sequence SRS_a)

$(Sa1)$ χ is an SRS_a , $\chi = b, x, \phi$
$(Sa2)$ $N_1, \ldots, N_k$ is an SRS_a and $M \rhd_a N_1$ $\Rightarrow$
$M, N_1, \ldots, N_k$ is an SRS_a
$(Sa3)$ $N_1, \ldots, N_k$ is an SRS_a $\Rightarrow$
$\lambda x.N_1, \ldots, \lambda x.N_k$ and $\Delta x.N_1, \ldots, \Delta x.N_k$ are SRS_a's
$(Sa4)$ $M_1, \ldots, M_j$ and $N_1, \ldots, N_k$ are SRS_a's $\Rightarrow$
$(M_1\ N_1), \ldots, (M_j\ N_1), (M_j\ N_2), \ldots, (M_j\ N_k)$ is an SRS_a

□

DEFINITION **36** (Standard reduction sequence SRS_c)

$(Sc1)$ χ is an SRS_c , $\chi = b, x, \phi$
$(Sc2)$ $N_1, \ldots, N_k$ is an SRS_c and $M \rhd_{cd} N_1$ $\Rightarrow$
$M, N_1, \ldots, N_k$ is an SRS_c
$(Sc3)$ $N_1, \ldots, N_k$ is an SRS_c $\Rightarrow$
$\lambda x.N_1, \ldots, \lambda x.N_k$ and $\Delta x.N_1, \ldots, \Delta x.N_k$ are SRS_c's
$(Sc4)$ $M_1, \ldots, M_j$ and $N_1, \ldots, N_k$ are SRS_c's $\Rightarrow$
$(M_1\ N_1), \ldots, (M_j\ N_1), (M_j\ N_2), \ldots, (M_j\ N_k)$ is an SRS_c

□

DEFINITION **37** (Standard reduction sequence SRS)

$(S1)$ An SRS_a is an SRS
$(S2)$ An SRS_c is an SRS
$(S3)$ $M_1, \ldots, M_k$ is an SRS_a and $M_k, \ldots, M_l$ is an SRS_c $\Rightarrow$
$M_1, \ldots, M_k, \ldots, M_l$ is an SRS

□

The strategy for the standardization proof by Plotkin [Plo75] is to introduce a parallel reduction equipped with a measure. We then prove, using the measure as an induction parameter, that parallel reduction can be postponed w.r.t. the standard reduction function (Lemma 39 below). This, in turn can be used to prove that a parallel reduction can be recursively transformed into a standard reduction sequence, built over the standard reduction function (Lemma 40 below.)

Define parallel reduction $\gg_a$ such that $\gg_a^* = \rhd_a^*$ by:

Definition **38** (Parallel reduction $\gg_a$)

$$\begin{array}{lll}
(P1) & M \gg_a M & \\
(P2) & M \gg_a M', N \gg_a N' & \Rightarrow \quad (\lambda x.M)\, N \gg_a M'\{x := N'\} \\
(P3) & M \gg_a M', N \gg_a N' & \Rightarrow \quad (\Delta x.M)\, N \gg_a \Delta\kappa.M'\{x := \lambda f.\kappa\ (f\ N')\} \\
(P4) & N \gg_a N' & \Rightarrow \quad \phi\ \Delta x.N \gg_a \Delta\kappa.N'\{x := \lambda f.\kappa\ (\phi\ f)\} \\
(P5) & \phi\ b \gg_a \delta(\phi, b) & \\
(P6) & M \gg_a M' & \Rightarrow \quad \lambda x.M \gg_a \lambda x.M' \\
(P7) & M \gg_a M' & \Rightarrow \quad \Delta x.M \gg_a \Delta x.M' \\
(P8) & M \gg_a M', N \gg_a N' & \Rightarrow \quad M\ N \gg_a M'\ N'
\end{array}$$

□

Define measure function s ascribing to each parallel reduction $M \gg_a M'$ a number $s_{M \gg_a M'} \geq 0$, which counts the number of $\rhd_a$-reductions implicit in $M \gg_a M'$. For any M, $n(x, M)$ is the number of free occurrences of x in M.

Definition **39** (Measure s)

$$\begin{array}{lll}
(1) & s_{M \gg_a M} & = \quad 0 \\
(2) & s_{(\lambda x.M)\, N \gg_a M'\{x:=N'\}} & = \quad s_{M \gg_a M'} + n(x, M') s_{N \gg_a N'} + 1 \\
(3) & s_{(\Delta x.M)\, N \gg_a \Delta\kappa.M'\{x:=\lambda f.\kappa\ (f\ N')\}} & = \quad s_{M \gg_a M'} + n(x, M') s_{N \gg_a N'} + 1 \\
(4) & s_{\phi\ \Delta x.N \gg_a \Delta\kappa.N'\{x:=\lambda f.\kappa\ (\phi\ f)\}} & = \quad s_{N \gg_a N'} + 1 \\
(5) & s_{\phi\ b \gg_a \delta(\phi, b)} & = \quad 1 \\
(6) & s_{\lambda x.M \gg_a \lambda x.M'} & = \quad s_{M \gg_a M'} \\
(7) & s_{\Delta x.M \gg_a \Delta x.M'} & = \quad s_{M \gg_a M'} \\
(8) & s_{M\ N \gg_a M'\ N'} & = \quad s_{M \gg_a M'} + s_{N \gg_a N'}
\end{array}$$

□

The following three technical lemmas are proved by easy but lengthy inductions. Compare [Fel87a] where analogous lemmas are used. The proofs are left out here.

Lemma **32** *Suppose $M \gg_a M'$ and $N \gg_a N'$. Then $M\{x := N\} \gg_a M'\{x := N'\}$, and with*

$$s_R = s_{M\{x:=N\} \gg_a M'\{x:=N'\}}$$

$$s_L = s_{(\lambda x.M)\ N \gg_a M'\{x:=N'\}}$$

we have $s_R < s_L$.

PROOF: Left out. □

LEMMA **33** *Suppose $M \gg_a M'$ and $N \gg_a N'$. Then $\Delta\kappa.M\{x := \lambda f.\kappa\ (f\ N)\} \gg_a \Delta\kappa.M'\{x := \lambda f.\kappa\ (f\ N')\}$, and with*

$$s_R = s_{\Delta\kappa.M\{x:=\lambda f.\kappa\ (f\ N)\} \gg_a \Delta\kappa.M'\{x:=\lambda f.\kappa\ (f\ N')\}}$$

$$s_L = s_{(\Delta x.M)\ N \gg_a \Delta\kappa.M'\{x:=\lambda f.\kappa\ (f\ N')\}}$$

we have $s_R < s_L$.

PROOF: Left out. □

LEMMA **34** *Suppose $N \gg_a N'$. Then $\Delta\kappa.N\{x := \lambda f.\kappa\ (\phi\ f)\} \gg_a \Delta\kappa.N'\{x := \lambda f.\kappa\ (\phi\ f)\}$, and with*

$$s_R = s_{\Delta\kappa.N\{x:=\lambda f.\kappa\ (\phi\ f)\} \gg_a \Delta\kappa.N'\{x:=\lambda f.\kappa\ (\phi\ f)\}}$$

$$s_L = s_{\phi\ \Delta x.N \gg_a \Delta\kappa.N'\{x:=\lambda f.\kappa\ (\phi\ f)\}}$$

we have $s_R < s_L$.

PROOF: Left out. □

To illustrate the technique used to prove the next three lemmas we give a full proof of the third which involves the Δ-operator. The other proofs are similar and found (in corresponding versions) in [Plo75] as well as in [Fel87a].

LEMMA **35** *If $M \gg_a \chi$ where M is an application and χ is a variable, a basic constant or a basic function, then $M \rhd_a^+ \chi$.*

PROOF: By induction on $s_{M \gg_a \chi}$. □

LEMMA **36** *If $M \gg_a \lambda x.N$ where M is an application, then there exists a λ-abstraction L such that $M \rhd_a^+ L \gg_a \lambda x.N$.*

PROOF: By induction on $s_{M \gg_a \lambda x.N}$. □

LEMMA **37** *If $M \gg_a \Delta x.N$ where M is an application, then there exists a Δ-abstraction L such that $M \rhd_a^+ L \gg_a \Delta x.N$.*

PROOF: Given in the last section. □

LEMMA **38** *(Second 3,4 Postponement lemma)*
If $M_1 \rhd_{3,4} M_2 \rhd_{6,7} M_3$ *then there exists an* M_4 *such that* $M_1 \rhd^+_{1,2,6,7} M_4 \rhd^*_{3,4} M_3$.

PROOF: Given in the last section. □

LEMMA **39** *($\gg_a$ Postponement)*
If $M \gg_a M' \rhd_a M''$ *then there exists an* L *such that* $M \rhd^+_a L \gg_a M''$.

PROOF: Given in the last section. □

LEMMA **40** *(Main lemma) If* $M \gg_a N_1$ *and* $N_1, \ldots, N_j$ *is an* SRS_a*, then there is an* SRS_a $L_1, \ldots, L_n$ *with* $L_1 \equiv M$ *and* $L_n \equiv N_j$.

PROOF: Given in the last section. □

LEMMA **41** *If* $M \rhd^*_{1,2,6,7} N$ *then there is an* SRS_a $L_1, \ldots, L_n$ *with* $L_1 \equiv M$ *and* $L_n \equiv N$.

PROOF: If $M \rhd^*_{1,2,6,7} N$ then $M \gg^n_a N$ for some $n \geq 0$. We proceed by induction on n: For $n = 0$ the claim is easily seen to hold, since any single term is an SRS_a. For $n > 0$, suppose $M \gg_a N_1 \gg^{n-1}_a N$. By induction, there is an SRS_a $L'_1, \ldots, L'_m$ such that $L'_1 \equiv N_1$ and $L'_m \equiv N$. By Lemma 40 there is then an SRS_a $L_1, \ldots, L_n$ with $L_1 \equiv M$ and $L_n \equiv L'_m \equiv N$. □

LEMMA **42** *If there is an* SRS_a $L_1, \ldots, L_n$ *then* $L_1 \rhd^*_{1,2,6,7} L_n$.

PROOF: Easy induction. □

Now we can prove the first half of the standardization theorem.

THEOREM **9** $M \rhd^*_{1,2,6,7} N$ *if and only if there is an* SRS_a $L_1, \ldots, L_n$ *such that* $L_1 \equiv M$ *and* $L_n \equiv N$.

PROOF: By Lemma 41 and Lemma 42. □

We move on to standardization of $\rhd_{3,4}$-reductions. Since no substitution is involved in these reductions the task is considerably simpler than the one we have just finished.

LEMMA **43** *If $M \rhd_{3,4} N_1$ and $N_1, \ldots, N_k$ is an SRS_c then there is an SRS_c $L_1, \ldots, L_n$ such that $L_1 \equiv M$ and $L_n \equiv N_k$.*

PROOF: Given in Appendix C. $\Box$

LEMMA **44** *If $M \rhd_{3,4} M' \rhd_c M''$ then there is an L such that $M \rhd_c L \rhd_{3,4} M''$.*

PROOF: Given in Appendix C. $\Box$

LEMMA **45** *If $M \rhd^*_{3,4} N$ then there is an SRS_c $L_1, \ldots, L_n$ such that $L_1 \equiv M$ and $L_n \equiv N$.*

PROOF: Easy, analogous to proof of Lemma 41, using Lemma 43. $\Box$

LEMMA **46** *$M \rhd^*_{3,4}$ if and only if there is an SRS_c $L_1, \ldots, L_n$ such that $M \equiv L_1$ and $L_n \equiv L_n$.*

PROOF: The left-to-right direction is Lemma 45, and the right-to-left direction is trivial. $\Box$

THEOREM **10** *(Standardization)*
$M \rhd^* N$ if and only if there is an SRS $L_1, \ldots, L_n$ such that $L_1 \equiv M$ and $L_n \equiv N$.

PROOF: The right-to-left direction is a trivial induction. To prove the left-to-right direction :

By our two postponement lemmas, Lemma 15 and Lemma 38, we know that $\rhd_{3,4}$-reductions can be postponed after the others, *i.e.* , if $M \rhd_{3,4} M' \rhd_{1,2,6,7} M''$ then there is an L such that $M \rhd^+_{1,2,6,7} L \rhd^*_{3,4} M''$. But postponement generalizes by induction to a factorization of arbitrary reductions : First we can show that if $M \rhd^*_{3,4} M' \rhd_{1,2,6,7} M''$ then there is an L such that $M \rhd_{1,2,6,7} L \rhd^*_{3,4} M''$. Using this we can show that if $M \rhd^*_{1,2,3,4,6,7} M''$ then there is an L such that $M \rhd^*_{1,2,6,7} L \rhd^*_{3,4} M''$.

Consequently, we may conclude that there is a K such that $M \rhd^*_{1,2,6,7} K \rhd^*_{3,4} N$. By Theorem 9 we can find an SRS_a $L'_1, \ldots, L'_k$ with $L'_1 \equiv M$ and $L'_k \equiv K$. By Lemma 46 we can find an SRS_c $L''_1, \ldots, L''_l$ with $L''_1 \equiv K \equiv L'_k$ and $L''_l \equiv N$. Then $M \equiv L'_1, \ldots, K \equiv L''_1, \ldots, L''_l \equiv N$ is an SRS with the desired properties. $\Box$

Evaluation and Correspondence. Working Note.

If we want to carry out Plotkin's program for a correspondence between a calculus and an evaluator, we should seek a computable evaluation function *Eval*, a partial function from programs (*i.e.* closed terms) to values. Given such a function and the calculus we can, for each program M, induce two series of partial functions from basic constants to basic constants : The n-ary functions $\mathcal{I}^n_M$ induced by the calculus and M, given by $\mathcal{I}^n_M(b_1 \ldots b_n) = b$ if and only if $\lambda_\Delta \vdash M b_1 \ldots b_n = b$; the n-ary functions $\mathcal{M}^n_M$ induced by *Eval* and M, given by $\mathcal{M}^n_M(b_1 \ldots b_n) = b$ if and only if $Eval(M b_1 \ldots b_n) = b$. Furthermore, *Eval* induces a congruence relation $\equiv_{Eval}$, called operational equivalence, in the standard manner (see [Plo75] or [Fel87b].) We would then seek to prove the following correspondence properties :

(C1) $Eval(M) = V$ for some value V if and only if $M \rhd^* V'$ for some value V'.

(C2) $\mathcal{I}^n_M = \mathcal{M}^n_M$

(C3) If $\lambda_\Delta \vdash M = N$ then $M \equiv_{Eval} N$.

In (C2) it is understood that the two functions are simultaneously defined. The property (C3) is often referred to as *soundness* (of the calculus with respect to operational equivalence), or simply *correctness* of the calculus w.r.t. the evaluator.

Now, as for (C1) we could not hope for the stronger result where $V \equiv V'$, since the reduction in the calculus will usually have stronger compatibility properties than evaluation. Thus, the λ_v-calculus studied by Plotkin reduces under λ's, whereas the evaluator only takes a term to weak head normal form, not touching the body of a λ-abstraction. There are good reasons for wanting the evaluator to be weak in this sense (we shall not rehearse them here.) But unfortunately, if we had an evaluator $Eval_\Delta$ for our language with this property, we could not hope for the property (C1) to hold. This is because evaluation under λ-abstractions may be necessary to reduce an expression to a value in λ_Δ. Consider as an example the expression $T \equiv \Delta x.x\ \lambda y.((\lambda d.z)\ x)$. This expression cannot be evaluated to a value, even if the evaluator were to go under Δ-abstractions (which, incidentally, it would have to in any case; see below.) But T reduces to $\lambda y.z$ in the calculus, upon reduction under the λ. This is a consequence of the restricted Δ-elimination rules we have in λ_Δ. Felleisen's original operator would bypass this problem, because x would in any case be bound to the top-level continuation and the control application eliminated.

It seems obvious that any sensible evaluator would necessarily have to go under Δ-abstractions, thus operating within the scope of a variable binding operator. This is again a consequence of the fact that a Δ can only be eliminated once its body has been transformed into some specific form to which the elimination rules (3) and (4) are applicable.

Moreover, this property has been a driving motivation behind our definition of standard reduction sequences as composed of $\rhd_{1,2,6,7}$-sequences and $\rhd_{3,4}$-

sequences, using the postponement properties. If we were to merge the $\rhd_{ab}$- and $\rhd_{cd}$-reductions into one reduction function, thereby also merging somehow the SRS_a- and SRS_c-sequences into one, then we find it very difficult to imagine how this could be done under preservation of the standardization theorem, unless the merged sequences were to be hopelessly complicated. The reason is again that a reduction may jump to contraction of a redex deeply nested within a Δ-abstraction thereby rendering the abstraction eliminable by the $\rhd_{3,4}$-reductions. This seems to be difficult to capture by sequences built around a reduction function which embodies a leftmost-outermost reduction strategy, and such a strategy, on its side, seems to be essential for an interesting standardization which must contain a considerable degree of determinism.

However, we might hope for a slightly weaker version of (C1). It might perhaps be possible to obtain $Eval_\Delta(M) = V$ if and only if $M \rhd^* V$ where $M \rhd^* V$ contracts no redex under a λ. Also, we might hope for (C2) and (C3) to hold without restrictions.

Note that the difficulty of finding a function *Eval* such that (C1) holds for it is closely connected to the demand that *Eval* be computable and given constructively (by an algorithmic specification.) As a curiosity we note that the problem could indeed be trivialized if we gave up the demand for constructivity and appealed to the Axiom of Choice (AC): Let the set $\mathcal{V}_M = \{V \in Values \mid M \rhd^* V\}$, and consider the collection $\mathcal{U} = \{\mathcal{V}_M \mid M \in \Lambda_\delta, \mathcal{V}_M \neq \emptyset\}$. By AC, there is a choice function $u : \mathcal{U} \to \Lambda_\delta$ such that for each $\mathcal{V}_M \in \mathcal{U}$ we have $u(\mathcal{V}_M) \in \mathcal{V}_M$. Now define $Eval(M) = u(\mathcal{V}_M)$. To see that (C1) holds, suppose first that $M \rhd^* V$ for some value V. Then $\mathcal{V}_M$ is not empty, so $\mathcal{V}_M \in \mathcal{U}$, hence $u(\mathcal{V}_M) = V' = Eval(M)$ is defined. Suppose $Eval(M) = V$ for some value V. Then $u(\mathcal{V}_M) = V$ with $V \in \mathcal{V}_M$, so $M \rhd^* V$. An evaluation function given in this way obviously would not satisfy us.

Now, how should we define $Eval_\Delta$? Given our standardization, an obvious way to start would be to find some class of terms, say $\mathcal{T}$, such that all terms in $\mathcal{T}$ are in normal form w.r.t. $\rhd_{ab}$ and then to stipulate $Eval_\Delta(M) = V$ if and only if there is an N in $\mathcal{T}$ such that $M \rhd_{ab}^* N \rhd_{cd}^* V$. We have, however, been unable so far to prove that $M \rhd^* b$ implies $Eval_\Delta(M) = b$, under this definition. We need something like a very strong form of postponement, *e.g.* , whenever $M \rhd^* b$ there is a $\rhd_{ab}$-normal form L such that $M \rhd_{ab}^* L \rhd_{cd}^* b$. We could alternatively try to find other suitable definitions of $\mathcal{T}$. Another appealing thought is to define $Eval_\Delta(M) = V$ if $M \rhd_{cd}^* V$ or else $Eval_\Delta(M) = Eval_\Delta(N)$ where $nf_{cd}(M) \rhd_{ab} N$, where $nf_{cd}(M)$ denotes the unique $\rhd_{cd}$-normal form of M, known to exist since $\rhd_{cd}$-reduction is evidently strongly normalizing. Again, we have been unable to prove, within the time at our disposal, the desired properties for $Eval_\Delta$ under this definition.

Proofs

LEMMA 37 *If $M \gg_a \Delta x.N$ where M is an application, then there exists a Δ-abstraction L such that $M \rhd_a^+ L \gg_a \Delta x.N$.*
PROOF: By induction on $s = s_{M \gg_a \Delta x.N}$, and $s = 0$ is impossible. For $s > 0$ we reason by cases over the last rule used to derive $M \gg_a \Delta x.N$. This must be $(P2)$, $(P3)$ or $(P4)$.

Case 1 : Last rule used is $(P2)$, so $M \equiv (\lambda y.P)\, Q \gg_a P'\{y := Q'\} \equiv \Delta x.N$ with $P \gg_a P'$ and $Q \gg_a Q'$. But then $P\{y := Q\} \gg_a P'\{y := Q'\} \equiv \Delta x.N$ with $s_{P\{y:=Q\} \gg_a P'\{y:=Q'\}} < s$ according to Lemma 32. Now, since $P\{y := Q\}$ reduces to a Δ-abstraction, $P\{y := Q\}$ must be either a Δ-abstraction or an application. These possibilities are investigated below:

Subcase 1.1 : Suppose $P\{y := Q\}$ is a Δ-abstraction. Then we take $L \equiv P\{y := Q\}$ and we have $M \equiv (\lambda y.P)\, Q \rhd_a P\{y := Q\} \equiv L \gg_a P'\{y := Q'\} \equiv \Delta x.N$.

Subcase 1.2 : Suppose $P\{y := Q\}$ is an application. Then, by induction, we can find a Δ-abstraction L' such that $P\{y := Q\} \rhd_a^+ L' \gg_a \Delta x.N$. Choose $L \equiv L'$.

Case 2 : The last rule used is $(P3)$, so $M \equiv (\Delta y.P)\, Q \gg_a \Delta x.P'\{y := \lambda f.x\,(f\,Q')\} \equiv \Delta x.N$ with $P \gg_a P'$ and $Q \gg_a Q'$. We can take $L \equiv \Delta x.P\{y := \lambda f.x\,(f\,Q)\}$ to get $M \rhd_a L \gg_a \Delta x.N$.

Case 3 : The last rule used is $(P4)$, so $M \equiv \phi\, \Delta y.P \gg_a \Delta x.P'\{y := \lambda f.x\,(\phi\, f)\} \equiv \Delta x.N$ with $P \gg_a P'$. We take $L \equiv \Delta x.P\{y := \lambda f.x\,(\phi\, f)\}$. □

LEMMA 38 *If $M_1 \rhd_{3,4} M_2 \rhd_{6,7} M_3$ then there exists an M_4 such that $M_1 \rhd^+_{1,2,6,7} M_4 \rhd^*_{3,4} M_3$.*
PROOF: Structural induction on M_1:

$\boxed{M_1 \equiv \chi}$: For $\chi = x, b, \phi$ the claim is trivially true.

$\boxed{M_1 \equiv \lambda x.P}$: The claim follows easily by induction.

$\boxed{M_1 \equiv P_1\; P_2}$: We proceed by cases over $M_1 \rhd_{3,4} M_2$.

Case 1 : $M_1 \rhd_{3,4} M_2$ is $P_1\, P_2 \rhd_{3,4} P_1'\, P_2 \equiv M_2$ with $P_1 \rhd_{3,4} P_1'$. We proceed by subcases over $M_2 \rhd_{6,7} M_3$:

Subcase 1.1 : $M_2 \rhd_{6,7} M_3$ is $M_2 \equiv P_1'\; P_2 \equiv \phi\; \Delta x.Q \rhd_7 \Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\, f)\} \equiv M_3$. There are two ways in which $P_1 \rhd_{3,4} P_1' \equiv \phi$ is possible :

Subcase 1.1.1 : $P_1 \rhd_{3,4} P_1' \equiv \phi$ is $P_1 \equiv \Delta y.y\,\phi \rhd_3 \phi$. Then

$$\begin{array}{lll} M_1 & \equiv & (\Delta y.y\,\phi)\,\Delta x.Q \\ & \rhd_2 & \Delta\kappa'.(y\,\phi)\{y := \lambda f.\kappa'\,(f\,\Delta x.Q)\} \\ & \equiv & \Delta\kappa'.(\lambda f.\kappa'\,(f\,\Delta x.Q))\,\phi \\ & \rhd_1 & \Delta\kappa'.\kappa'\,(\phi\,\Delta x.Q) \\ & \rhd_7 & \Delta\kappa'.\kappa'\,\Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\} \\ & \rhd_3 & \Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\} \end{array}$$

so we can take $M_4 \equiv \Delta\kappa'.\kappa'\,\Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\}$.

Subcase 1.1.1 : $P_1 \rhd_{3,4} P_1' \equiv \phi$ is $P_1 \equiv \Delta y.y\,\nabla(y\,\phi) \rhd_4 \phi$. Then

$$\begin{array}{lll} M_1 & \equiv & (\Delta y.y\,\nabla(y\,\phi))\,\Delta x.Q \\ & \rhd_2 & \Delta\kappa'.(y\,\nabla(y\,\phi))\{y := \lambda f.\kappa'\,(f\,\Delta x.Q)\} \\ & \equiv & \Delta\kappa'.(\lambda f.\kappa'\,(f\,\Delta x.Q))\,\nabla((\lambda f.\kappa'\,(f\,\Delta x.Q))\,\phi) \\ & \rhd_1^* & \Delta\kappa'.\kappa'\,(\nabla(\kappa'\,(\phi\,\Delta x.Q))\,\Delta x.Q) \\ & \rhd_2 & \Delta\kappa'.\kappa'\,\nabla(\kappa'\,(\phi\,\Delta x.Q)) \\ & \rhd_7 & \Delta\kappa'.\kappa'\,\nabla(\kappa'\,\Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\}) \\ & \rhd_4 & \Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\} \end{array}$$

so we can take $M_4 \equiv \Delta\kappa'.\kappa'\,\nabla(\kappa'\,\Delta\kappa.Q\{x := \lambda f.\kappa\,(\phi\,f)\})$.

Subcase 1.2 : $M_2 \rhd_{6,7} M_3$ is $M_2 \equiv P_1'\,P_2 \equiv \phi\,b \rhd_6 \delta(\phi, b) \equiv M_3$. There are two possibilities for $P_1 \rhd_{3,4} P_1' \equiv \phi$:

Subcase 1.2.1 : $P_1 \rhd_{3,4} P_1' \equiv \phi$ is $P_1 \equiv \Delta y.y\,\phi \rhd_3 \phi$. Then

$$\begin{array}{lll} M_1 & \equiv & (\Delta y.y\,\phi)\,b \\ & \rhd_2 & \Delta\kappa.(y\,\phi)\{y := \lambda f.\kappa\,(f\,b)\} \\ & \equiv & \Delta\kappa.(\lambda f.\kappa\,(f\,b))\,\phi \\ & \rhd_1 & \Delta\kappa.\kappa\,(\phi\,b) \\ & \rhd_6 & \Delta\kappa.\kappa\,\delta(\phi, b) \\ & \rhd_3 & \delta(\phi, b) \end{array}$$

so we can choose $M_4 \equiv \Delta\kappa.\kappa\,\delta(\phi, b)$.

Subcase 1.2.2 : $P_1 \rhd_{3,4} P_1'$ is $P_1 \equiv \Delta y.y\,\nabla(y\,\phi) \rhd_4 \phi$, so that

$$\begin{array}{lll} M_1 & \equiv & (\Delta y.y\,\nabla(y\,\phi))\,b \\ & \rhd_2 & \Delta\kappa.(y\,\nabla(y\,\phi))\{y := \lambda f.\kappa\,(f\,b)\} \\ & \equiv & \Delta\kappa.(\lambda f.\kappa\,(f\,b))\,\nabla((\lambda f.\kappa\,(f\,b))\,\phi) \\ & \rhd_1^* & \Delta\kappa.\kappa\,(\nabla(\kappa\,(\phi\,b))\,b) \\ & \rhd_2 & \Delta\kappa.\kappa\,\nabla(\kappa\,(\phi\,b)) \\ & \rhd_6 & \Delta\kappa.\kappa\,\nabla(\kappa\,\delta(\phi, b)) \\ & \rhd_4 & \delta(\phi, b) \end{array}$$

so we can take $M_4 \equiv \Delta\kappa.\kappa\,\nabla(\kappa\,\delta(\phi, b))$.

Subcases 1.3 and 1.4 : $M_2 \rhd_{6,7} M_3$ is either $P_1' P_2 \rhd_{6,7} P_1'' P_2$ with $P_1' \rhd_{6,7} P_1''$, or it is $P_1' \, P_2 \rhd_{6,7} P_1' \, P_2'$ with $P_2 \rhd_{6,7} P_2'$. In the first case we get our claim by induction hypothesis, and in the second case we can take $M_4 \equiv P_1 \, P_2'$.

Case 2 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv P_1 \, P_2 \rhd_{3,4} P_1 \, P_2' \equiv M_2$ with $P_2 \rhd_{3,4} P_2'$. We proceed by cases over $M_2 \rhd_{6,7} M_3$:

Subcase 2.1 : $M_2 \rhd_{6,7} M_3$ is $M_2 \equiv \phi \, b \rhd_6 \delta(\phi, b) \equiv M_3$. Then $P_1 \equiv \phi$ and $P_2 \equiv \Delta x.x \, b$ or $P_2 \equiv \Delta x.x \, \nabla(x \, b)$.

In the first case we get

$$\begin{array}{rcl} M_1 & \equiv & \phi \, \Delta x.x \, b \\ & \rhd_7 & \Delta \kappa.(x \, b)\{x := \lambda f.\kappa \, (\phi \, f)\} \\ & \equiv & \Delta \kappa.(\lambda f.\kappa \, (\phi \, f)) \, b \\ & \rhd_1 & \Delta \kappa.\kappa \, (\phi \, b) \\ & \rhd_6 & \Delta \kappa.\kappa \, \delta(\phi, b) \\ & \rhd_3 & \delta(\phi, b) \end{array}$$

so we can take $M_4 \equiv \Delta \kappa.\kappa \, \delta(\phi, b)$.

In the second case we get

$$\begin{array}{rcl} M_1 & \equiv & \phi \, \Delta x.x \, \nabla(x \, b) \\ & \rhd_7 & \Delta \kappa.(x \, \nabla(x \, b))\{x := \lambda f.\kappa \, (\phi \, f)\} \\ & \rhd_1^* & \Delta \kappa.(\lambda f.\kappa \, (\phi \, f)) \, \nabla((\lambda f.\kappa \, (\phi \, f)) \, b) \\ & \rhd_1 & \Delta \kappa.\kappa \, (\phi \, \nabla((\lambda f.\kappa \, (\phi \, f)) \, b)) \\ & \rhd_7 & \Delta \kappa.\kappa \, \nabla((\lambda f.\kappa \, (\phi \, f)) \, b) \\ & \rhd_1 & \Delta \kappa.\kappa \, \nabla(\kappa \, (\phi \, b)) \\ & \rhd_6 & \Delta \kappa.\kappa \, \nabla(\kappa \, \delta(\phi, b)) \\ & \rhd_4 & \delta(\phi, b) \end{array}$$

so we can take $M_4 \equiv \Delta \kappa.\kappa \, \nabla(\kappa \, \delta(\phi, b))$.

Subcase 2.2 : $M_2 \rhd_{6,7} M_3$ is $M_2 \equiv \phi \Delta x.Q \rhd_7 \Delta \kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\} \equiv M_3$. Then $P_1 \equiv \phi$ and $P_2 \rhd_{3,4} \Delta x.Q \equiv P_2'$. We have three possibilities for $P_2 \rhd_{3,4} P_2'$:

Subcase 2.2.1 : $P_2 \rhd_{3,4} P_2'$ is $P_2 \equiv \Delta y.y \, \Delta x.Q \rhd_3 \Delta x.Q$, $y \notin FV(Q)$. Then

$$\begin{array}{rcl} M_1 & \equiv & \phi \, \Delta y.y \, \Delta x.Q \\ & \rhd_7 & \Delta \kappa'.(y \, \Delta x.Q)\{y := \lambda f.\kappa' \, (\phi \, f)\} \\ & \equiv & \Delta \kappa'.(\lambda f.\kappa' \, (\phi \, f)) \, \Delta x.Q \\ & \rhd_1 & \Delta \kappa'.\kappa' \, (\phi \, \Delta x.Q) \\ & \rhd_7 & \Delta \kappa'.\kappa' \, (\Delta \kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\}) \\ & \rhd_3 & \Delta \kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\} \end{array}$$

so we can choose $M_4 \equiv \Delta \kappa'.\kappa' \, (\Delta \kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\})$.

Subcase 2.2.2 : $P_2 \rhd_{3,4} P_2'$ is $P_2 \equiv \Delta y.y \, \nabla(y \, \Delta x.Q) \rhd_4 \Delta x.Q$, $y \notin FV(Q)$. Then

$$\begin{array}{rcl} M_1 & \equiv & \phi \, \Delta y.y \, \nabla(y \, \Delta x.Q) \\ & \rhd_7 & \Delta\kappa'.(y \, \nabla(y \, \Delta x.Q))\{y := \lambda f.\kappa' \, (\phi \, f)\} \\ & \equiv & \Delta\kappa'.(\lambda f.\kappa' \, (\phi \, f)) \, \nabla((\lambda f.\kappa' \, (\phi \, f)) \, \Delta x.Q) \\ & \rhd_1^* & \Delta\kappa'.\kappa' \, (\phi \, \nabla(\kappa' \, (\phi \, \Delta x.Q))) \\ & \rhd_7 & \Delta\kappa'.\kappa' \, \nabla(\kappa' \, (\phi \, \Delta x.Q)) \\ & \rhd_7 & \Delta\kappa'.\kappa' \, \nabla(\kappa' \, \Delta\kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\}) \\ & \rhd_4 & \Delta\kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\} \end{array}$$

so we can take $M_4 \equiv \Delta\kappa'.\kappa' \, \nabla(\kappa' \, \Delta\kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\})$.

Subcase 2.2.3 : $P_2 \rhd_{3,4} P_2'$ is $P_2 \equiv \Delta x.Q_1 \rhd_{3,4} \Delta x.Q \equiv P_2'$ with $Q_1 \rhd_{3,4} Q$. Then

$$\begin{array}{rcl} M_1 & \equiv & \phi \, \Delta x.Q_1 \\ & \rhd_7 & \Delta\kappa.Q_1\{x := \lambda f.\kappa \, (\phi \, f)\} \\ & \rhd_{3,4} & \Delta\kappa.Q\{x := \lambda f.\kappa \, (\phi \, f)\} \end{array}$$

by substitutivity of $\rhd_{3,4}$. Choose $M_4 \equiv \Delta\kappa.Q_1\{x := \lambda f.\kappa \, (\phi \, f)\}$.

Subcases 2.3 and 2.4 : $M_2 \rhd_{6,7} M_3$ is either $P_1 \, P_2' \rhd_{6,7} P_1' \, P_2'$ with $P_1 \rhd_{6,7} P_1'$, or it is $P_1 \, P_2' \rhd_{6,7} P_1 \, P_2''$ with $P_2' \rhd_{6,7} P_2''$. In the first case we can take $M_4 \equiv P_1' \, P_2$, and in the second case the claim follows by induction hypothesis.

$\boxed{M_1 \equiv \Delta x.P}$: We proceed by cases over $N_1 \rhd_{3,4} M_2$:

Case 1 : $M_1 \rhd_{3,4} M_2$ is either $M_1 \equiv \Delta x.x \, P_1 \rhd_3 P_1 \equiv M_2$ or $M_1 \equiv \Delta x.x \nabla(x \, P_1) \rhd_4 P_1 \equiv M_2$, $x \notin FV(P_1)$. So, in any case, we must have that $M_2 \rhd_{6,7} M_3$ is $P_1 \rhd_{6,7} M_3$, and in the first case we get $M_1 \equiv \Delta x.x \, P_1 \rhd_{6,7} \Delta x.x \, M_3 \rhd_3 M_3$, so we can choose $M_4 \equiv \Delta x.x \, M_3$. The second case is similar.

Case 2 : $M_1 \rhd_{3,4} M_2$ is $M_1 \equiv \Delta x.P \rhd_{3,4} \Delta x.P' \equiv M_2$ with $P \rhd_{3,4} P'$. The only possibility for $M_2 \rhd_{6,7} M_3$ is $\Delta x.P' \rhd_{6,7} \Delta x.P''$ with $P' \rhd_{6,7} P''$, and the claim follows by induction.

□

Lemma 39 *If $M \gg_a M' \rhd_a M''$ then there exists an L such that $M \rhd_a^+ L \gg_a M''$.*
Proof: Lexicographic induction on $< s, \mid M \mid >$ where $s = s_{M \gg_a M'}$, by cases over last rule used to derive $M \gg_a M'$.

Case P1 : Here $M \equiv M'$, and we take $L \equiv M'$.

Case P2 : Here $M \equiv (\lambda x.P) \, Q \gg_a P'\{x := Q'\} \equiv M' \rhd_a M''$ with $P \gg_a P'$ and $Q \gg_a Q'$. Since $P\{x := Q\} \gg_a P'\{x := Q'\}$ with $s_{P\{x:=Q\} \gg_a P'\{x:=Q'\}} < s$, we get by induction an L' such that $P\{x := Q\} \rhd_a^+ L' \gg_a M''$. Take $L \equiv L'$, and we have $M \equiv (\lambda x.P) \, Q \rhd_a P\{x := Q\} \rhd_a^+ L \gg_a M''$.

Case P3 : Here $M \equiv (\Delta x.P)\,Q \gg_a \Delta\kappa.P'\{x := \lambda f.\kappa\,(f\,Q')\} \equiv M'$. The claim is trivially true, since $M' \rhd_a M''$ is evidently impossible.

Case P4 : This case is also impossible.

Case P5 : This case is also impossible.

Case P6 : This case is also impossible.

Case P7 : This case is also impossible.

Case P8 : Here we have $M \equiv P\,Q \gg_a P'\,Q' \equiv M'$ with $P \gg_a P'$ and $Q \gg_a Q'$. We proceed by subcases over $M' \rhd_a M''$.

Subcase 1 : $M' \equiv (\lambda x.P_1')\,Q' \rhd_a P_1'\{x := Q'\} \equiv M''$ where $P \gg_a \lambda x.P_1' \equiv P'$. Now, $P < ga\lambda x.P_1'$ implies that P is either a λ-abstraction or an application. Therefore, by Lemma 36, we conclude that there is a λ-abstraction L' such that $P \rhd_a^* L' \gg_a \lambda x.P_1'$. We must have $L' \equiv \lambda x.L_1'$ with $L_1' \gg_a P_1'$, and therefore $M \equiv PQ \rhd_a^* L'Q \equiv (\lambda x.L_1')Q \rhd_a L_1'\{x := Q\}$. Since $L_1'\{x := Q\} \gg_a P_1'\{x := Q'\}$ we can take $L \equiv L_1'\{x := Q\}$.

Subcase 2 : $M' \equiv (\Delta x.P{*_1})\,Q' \rhd_a \Delta\kappa.P_1'\{x := \lambda f.\kappa\,(f\,Q')\} \equiv M''$ with $P \gg_a \Delta x.P_1' \equiv P'$. Now, $P \gg_a \Delta x.P_1'$ implies that P is either a Δ-abstraction or an application. Therefore, by Lemma 37, we can find a Δ-abstraction $\Delta x.L'$ such that $P \rhd_a^* \Delta x.L'$ with $L' \gg_a P_1'$. Hence $M \equiv P\,Q \rhd_a^* (\Delta x.L')\,Q \rhd_a \Delta\kappa.L'\{x := \lambda f.\kappa\,(f\,Q)\}$. Since $L'\{x := \lambda f.\kappa\,(f\,Q)\} \gg_a P_1'\{x := \lambda f.\kappa\,(f\,Q')\}$, we can take $L \equiv \Delta\kappa.L'\{x := \lambda f.\kappa\,(f\,Q)\}$.

Subcase 3 : $M' \equiv \phi\,\Delta x.Q_1' \rhd_a \Delta\kappa.Q_1'\{x := \lambda f.\kappa\,(\phi\,f)\} \equiv M''$ with $P \gg_a \phi \equiv P'$ and $Q \gg_a \Delta x.Q_1' \equiv Q'$. Now, $Q \gg_a \Delta x.Q_1'$ implies that Q is either a Δ-abstraction or an application. By Lemma 37 we conclude that there is a Δ-abstraction $\Delta x.L'$ such that $Q \rhd_a^* \Delta x.L'$ with $L' \gg_a Q_1'$. Also, $P \gg_a \phi$ implies that P is either an application or $P \equiv \phi$, so by Lemma 35 $P \rhd_a^* \phi$. Consequently, $M \equiv P\,Q \rhd_a^* \phi\,Q \rhd_a^* \phi\,\Delta x.L' \rhd_a \Delta\kappa.L'\{x := \lambda f.\kappa\,(\phi\,f)\}$. Since $L'\{x := \lambda f.\kappa\,(\phi\,f)\} \gg_a Q_1'\{x := \lambda f.\kappa\,(\phi\,f)\}$ we can take $L \equiv \Delta\kappa.L'\{x := \lambda f.\kappa\,(\phi\,f)\}$.

Subcase 4 : $M' \equiv \phi\,b \rhd_a \delta(\phi, b) \equiv M''$, so $P \gg_a \phi \equiv P'$ and $Q \gg_a b \equiv Q'$. This implies that P is either an application or $P \equiv \phi$, and Q is either an application or $Q \equiv b$. By Lemma 35 we conclude $M \equiv PQ \rhd_a^* \phi Q \rhd_a^* \phi b \rhd_a \delta(\phi, b)$, and we take $L \equiv \delta(\phi, b)$.

Subcase 5 : $M' \equiv P'\,Q' \rhd_a P''\,Q' \equiv M''$ with $P' \rhd_a P''$. So we have $P \gg_a P' \rhd_a P''$, and by induction there is an L' such that $P \rhd_a^+ L' \gg_a P''$. Therefore, $M \equiv P\,Q \rhd_a^+ L'\,Q \gg_a P''\,Q \equiv M''$, and we can take $L \equiv L'\,Q$.

Subcase 6 : $M' \equiv \chi\,Q' \rhd_a \chi\,Q'' \equiv M''$ with χ a constant or a variable and $Q \rhd_a Q''$, $P \gg_a \chi$. By Lemma 35 we get $P \rhd_a^* \chi$, and $Q \gg_a Q' \rhd_a Q''$ implies by induction that $Q \rhd_a^+ L' \gg_a Q''$ for some L'. Consequently, $M \equiv P\,Q \rhd_a^* \chi\,Q \rhd_a^+ \chi\,L' \gg_a \chi\,Q'' \equiv M''$, and we can take $L \equiv \chi\,L'$. □

LEMMA 40 *If $M \gg_a N_1$ and $N_1, \ldots, N_j$ is an SRS_a, then there is an SRS_a $L_1, \ldots, L_n$ with $L_1 \equiv M$ and $L_n \equiv N_j$.*

Proof: The proof is by lexicographic induction on $< j, s, \mid M \mid>$ where $s = s_{M \gg_a N_1}$, and by cases over the last rule used to derive $M \gg_a N_1$.

Case P1 : $M \equiv N_1$. Take $n = j$ and $L_i \equiv N_i$, $i = 1 \ldots n$.

Case P2: $M \equiv (\lambda x.P)Q \gg_a P'\{x := Q'\} \equiv N_1$ with $P \gg_a P'$ and $Q \gg_a Q'$. Now, we have $P\{x := Q\} \gg_a P'\{x := Q'\}$ with $s_{P\{x:=Q\} \gg_a P'\{x:=Q'\}} < s$. So, by induction, there is an SRS_a $L'_1, \ldots, L'_m$ with $P\{x := Q\} \equiv L'_1$ and $L'_m \equiv N_j$. Since $(\lambda x.P)\ Q \rhd_a P\{x := Q\}$ it follows by $(Sa2)$ that also $M, L'_1 \equiv P\{x := Q\}, \ldots, L'_m \equiv N_j$ is an SRS_a. Take $L_1 \equiv M$, $L_2 \equiv L'_1, \ldots, L_n \equiv L'_m$.

Case P3 : $M \equiv (\Delta x.P)Q \gg_a \Delta\kappa.P'\{x := \lambda f.\kappa\ (f\ Q')\} \equiv N_1$ with $P \gg_a P'$, $Q \gg_a Q'$. Now, we have $\Delta\kappa.P\{x := \lambda f.\kappa\ (f\ Q)\} \gg_a N_1$ and $s_{\Delta\kappa.P\{x:=\lambda f.\kappa\ (f\ Q)\} \gg_a N_1} < s$. Hence, by induction, there is an SRS_a $L'_1, \ldots, L'_m$ with $\Delta\kappa.P\{x := \lambda f.\kappa\ (f\ Q)\} \equiv L'_1$ and $L'_m \equiv N_j$. Since $M \rhd_a \Delta\kappa.P\{x := \lambda f.\kappa\ (f\ Q)\}$ it follows by $(Sa2)$ that $M, L'_1 \equiv \Delta\kappa.P\{x := \lambda f.\kappa\ (f\ Q)\}, \ldots, L'_m$ is an SRS_a. Take $L_1 \equiv M, L_2 \equiv L'_2, \ldots, L_n \equiv L'_m$.

Case P4 : $M \equiv \phi\ \Delta x.Q \gg_a \Delta\kappa.Q'\{x := \lambda f.\kappa\ (\phi\ f)\} \equiv N_1$. This case is similar to the previous one.

Case P5 : $M \equiv \phi\ b \gg_a \delta(\phi, b) \equiv N_1$. Since $\phi\ b \rhd_a \delta(\phi, b)$ we take $L_1 \equiv M, L_2 \equiv \delta(\phi, b), L_3 \equiv N_2, \ldots, L_n \equiv N_j$ which is an SRS_a by $(Sa2)$.

Case P6 : $M \equiv \lambda x.P \gg_a \lambda x.P_1 \equiv N_1$ with $P \gg_a P_1$. Since $N_1 \equiv \lambda x.P_1$, it must be the case that $N_i \equiv \lambda x.P_i$ for $i = 1 \ldots j$, and $P_1, \ldots, P_j$ is an SRS_a. Since $\mid P \mid < \mid M \mid$ we get by induction an SRS_a $L'_1, \ldots, L'_m$ such that $P \equiv L'_1$ and $P_j \equiv L'_m$. Hence, by $(Sa3)$, $M \equiv \lambda x.P, \lambda x.L'_2, \ldots, \lambda x.L'_m \equiv \lambda x.P_j \equiv N_j$ is an SRS_a, and we take $L_1 \equiv M, L_1 \equiv \lambda x.L'_2, \ldots, L_n \equiv \lambda x.L'_m$.

Case P7: $M \equiv \Delta x.P \gg_a \Delta x.P_1 \equiv N_1$ with $P \gg_a P_1$. Since $N_1 \equiv \Delta x.P_1$ it must be the case that $N_i \equiv \Delta x.P_i$, $i = 1 \ldots j$, and $P_1, \ldots, P_j$ is an SRS_a. Now, $\mid P \mid < \mid M \mid$, so by induction there is an SRS_a $L'_1, \ldots, L'_m$ with $P \equiv L'_1$ and $P_j \equiv L'_m$. Consequently, by $(Sa3)$, $M \equiv \Delta x.P, \Delta x.L'_2, \ldots, \Delta x.L'_m \equiv \Delta x.P_j \equiv N_j$ is an SRS_a, and we take $L_1 \equiv M, L_2 \equiv \Delta x.L'_2, \ldots, L_n \equiv \Delta x.L'_m$.

Case P8 : $M \equiv P\ Q \gg_a P'\ Q' \equiv N_1$ with $P \gg_a P'$ and $Q \gg_a Q'$. There are two subcases: Either (Subcase 1) $N_2, \ldots, N_j$ is an SRS_a with $P'\ Q' \rhd_a N_2$, or (Subcase 2) $P', S_2, \ldots, S_k$ and $Q', T_2, \ldots, T_l$ are SRS_a's, and

$$N_1, \ldots, N_j \equiv (P'\ Q'), \ldots, (S_k\ Q'), (S_k\ T_2), \ldots, (S_k\ T_l)$$

Subcase 1 : Here we have $M \equiv P\ Q \gg_a P'\ Q' \rhd_a N_2$, and so by Lemma 39 there is an L such that $M \rhd_a L \gg_a N_2$, and since $N_2, \ldots, N_j$ has length $j - 1$ we get by induction that there is an SRS_a $L'_1, \ldots, L'_m$ with $L'_1 \equiv L$ and $L'_m \equiv N_j$. By $(Sa2)$ $M, L'_1, \ldots, L'_m$ is also an SRS_a, and we take $L_1 \equiv M, L_2 \equiv L'_1, \ldots, L_n \equiv L'_m$.

Subcase 2 : We have $P \gg_a P'$ with $P', S_2, \ldots, S_k$ an SRS_a, $k \leq j$. Since $\mid P \mid < \mid M \mid$ we get by induction that there is an SRS_a $U_1, \ldots, U_m$ such that $U_1 \equiv P$ and $U_m \equiv S_k$. Also, $Q \gg_a Q'$ with $Q', T_2, \ldots, T_l$ an SRS_a, $l \leq j$, and since $\mid Q \mid < \mid M \mid$ we get by induction an SRS_a $V_1, \ldots, V_p$ such that $V_1 \equiv Q$ and

$V_p \equiv T_l$. Therefore, by $(Sa4)$, the following is also an SRS_a :

$$M \equiv P\,Q \equiv (U_1\,V_1), \ldots, (U_m\,V_1), (U_m\,V_2), \ldots, (U_m\,V_p) \equiv (S_k\,T_l) \equiv N_j$$

We take $L_1 \equiv (U_1\,V_1), \ldots L_n \equiv (U_m\,V_p)$. □

LEMMA 44 *If $M \rhd_{3,4} M' \rhd_{cd} M''$ then there is an L such that $M \rhd_c L \rhd_{3,4} M''$.*
PROOF: By structural induction on M, and M must be either a Δ-abstraction or an application.

$\boxed{M \equiv \Delta x.P}$:

Case 1 : $M \equiv \Delta x.P \rhd_{3,4} \Delta x.P' \equiv M'$ with $P \rhd_{3,4} P'$.

Subcase 1.1 : $M' \equiv \Delta x.P' \rhd_{cd} \Delta x.P'' \equiv M''$ with $P \rhd_{cd} P''$. Then M' is not trivial, and therefore M is not trivial either, since $\rhd_{3,4}$-reduction cannot eliminate free variables. By induction there is an L' such that $P \rhd_{cd} L' \rhd_{3,4} P''$, so $M \equiv \Delta x.P \rhd_{cd} \Delta x.L' \rhd_{3,4} \Delta x.P'' \equiv M''$, and we can take $L \equiv \Delta x.L'$.

Subcase 1.2 : $M' \equiv \Delta x.x\,M'' \rhd_{cd} M''$ where M' is trivial, so M is also trivial. Now, $P \rhd_{3,4} x\,M''$, and this can only be the case if there is a $\rhd_{3,4}$-redex surrounding x (as in *e.g.* $P \equiv (\Delta y.y\,x)\,M''$), or there is one surrounding $x\,M''$ (as in *e.g.* $P \equiv \Delta y.y(x\,M'')$), or finally if $P \equiv x\,P_1$ with $P_1 \rhd_{3,4} M''$. In the first two cases we have *e.g.* $M \equiv \Delta x.(\Delta y.y\,x)\,M'' \rhd_{cd} \Delta x.x\,M'' \rhd_{3,4} M''$, so $L \equiv \Delta x.x\,M''$. In the third case we have $M \equiv \Delta x.x\,P_1 \rhd_{cd} P_1 \rhd_{3,4} M''$, and $L \equiv P_1$.

Subcase 1.3 : $M' \equiv \Delta x.x\,\nabla(x\,M'') \rhd_{cd} M''$ where M' is trivial. This is similar to the previous case.

Case 2 : $M \equiv \Delta x.x\,P_1 \rhd_{3,4} P_1 \rhd_{cd} M''$, where M is trivial. We have $M \rhd_{cd} P_1 \rhd_{3,4} M''$, since $\rhd_{cd} \subseteq \rhd_{3,4}$. The case where $M \equiv \Delta x.x\,\nabla(x\,P_1)$ is similar.

$\boxed{M \equiv P\,Q}$: We must have either $P\,Q \rhd_{3,4} P'\,Q$ or $P\,Q \rhd_{3,4} P\,Q'$ with $P \rhd_{3,4} P'$ and $Q \rhd_{3,4} Q'$, respectively. Here the claim follows either by induction, or we can reduce each side of the application in reversed order.

□

LEMMA 43 *If $M \rhd_{3,4} N_1$ and $N_1, \ldots, N_k$ is an SRS_c then there is an SRS_c $L_1, \ldots, L_n$ such that $L_1 \equiv M$ and $L_n \equiv N_k$.*
PROOF: Lexicographic induction on $< k, \mid M \mid >$, by cases over structure of M:

$\boxed{M \equiv x, b, \phi}$: The claim is trivially true.

$\boxed{M \equiv \lambda x.P}$: This case is analogous to case $(P6)$ in the proof of Lemma 40.

$\boxed{M \equiv \Delta x.P}$:

Case 1 : $M \equiv \Delta x.x\, P_1 \rhd_3 P_1 \equiv N_1$. Then $M \rhd_c N_1$ and $M, N_1, \ldots, N_k$ is an SRS_c.

Case 2 : $M \equiv \Delta x.x\, \nabla(x\, P_1) \rhd_4 P_1 \equiv N_1$. Analogous to the previous case.

Case 3 : $M \equiv \Delta x.P \rhd_{3,4} \Delta x.P_1 \equiv N_1$ with $P \rhd_{3,4} P_1$. There are two possibilities according to why $N_1, \ldots, N_k$ is an SRS_c:

Subcase 3.1 : $N_1, \ldots, N_k$ is an SRS_c because $N_1 \rhd_{cd} N_2$, and $N_2, \ldots, N_k$ is an SRS_c. Then we have $M \rhd_{3,4} N_1 \rhd_{cd} N_2$, and by Lemma 44 there is an L such that $M \rhd_{cd} L \rhd_{3,4} N_2$. Since $N_2, \ldots, N_k$ is an SRS_c of length $k-1$, we conclude by induction that there is an SRS_c $SRS_c L'm$ with $L \equiv L'_1$ and $L'_m \equiv N_k$. Then also $M, L, L'_2, \ldots, L'_m \equiv N_k$ is an SRS_c.

Subcase 3.2 : $N_1, \ldots, N_k$ is an SRS_c because $N_i \equiv \Delta x.P_i$ for $i = 1 \ldots k$ where $P_1, \ldots, P_k$ is an SRS_c. Since $\mid P \mid<\mid M \mid$, we get by induction that there is an SRS_c $L'{}_1, \ldots, L'{}_m$ such that $L'_1 \equiv P$ and $L'_m \equiv P_k$. Then also $M \equiv \Delta x.P \equiv \Delta x.L'_1, \Delta x.L'_2, \ldots, \Delta x.L'_m \equiv \Delta x.P_k \equiv N_k$ is an SRS_c.

$\boxed{M \equiv P\, Q}$: We must have either $M \equiv P\, Q \rhd_{3,4} P'\, Q \equiv N_1$ with $P \rhd_{3,4} P'$ or $M \equiv P\, Q \rhd_{3,4} P\, Q' \equiv N_1$ with $Q \rhd_{3,4} Q'$. In the first case there must be two SRS_c's, $P', S_2, \ldots, S_j$ and $Q, T_2, \ldots T_l$ such that $N_1, \ldots, N_k \equiv (P'Q), \ldots, (S_jQ), (S_jT_2), \ldots, (S_jT_l)$. So we have $P \rhd_{3,4} P'$ with $P', S_2, \ldots, S_j$ an SRS_c, $j \leq k$ and $\mid P \mid<\mid M \mid$, so by induction there is an SRS_c $U_1, \ldots, U_m$ with $U_1 \equiv P$ and $U_m \equiv S_j$. Then also $(U_1Q), \ldots, (U_mQ), (U_mT_2), \ldots, (U_mT_l)$ is an SRS_c, and we have $(U_1\, Q) \equiv M$ and $(U_m\, T_l) \equiv N_k$. The second case is analogous.

□

References

[Bar84] H.P. Barendregt. *The Lambda Calculus, its Syntax and Semantics.* Studies In Logic And The Foundations Of Mathematics, vol 103. Revised edition. North-Holland, Amsterdam, 1984.

[Bar91] H.P Barendregt. *Lambda Calculus Summer School Notes part II.* University of Nijmegen, 1991.

[Bar92] Franco Barbanera & Stafano Beradi. Continuations and Simple Types: A Strong Normalization Result. In *Proceedings of the ACM SIGPLAN Workshop on Continuations CW92.* San Francisco, California, 1992.

[Boh85] Corrado Böhm & Alessandro Beraducci. Automatic Synthesis of Typed Λ-Programs on Term Algebras. In *Theoretical Computer Science.* 39:135-154, 1985.

[Cur58] H.B. Curry & R. Feys, Combinatory Logic, vol I. Noth-Holland, 1958.

[Dub90] B.F. Duba, R. Harper, D. MacQueen. Typing first-class continuations in ML. In *Eighteenth ACM Symposium on Principles of Programming Languages*, 1991.

[Fel87a] M. Felleisen, D. Friedman, E. Kohlbecker, B. Duba, A syntactic theory of sequential Control. In *Theoretical Computer Science,* Vol. 52(3) pp205-237, 1987.

[Fel87b] Matthias Felleisen, The Calculi of λ_v-CS Conversion : A Syntactic Theory of Control and State in Imperative Higher Order Programming Languages. Ph.D. Thesis, Indiana University, 1987.

[Fel89] M.Felleisen, R.Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. In *Rice University Technical Report.* Rice COMP TR89-100 1989.

[Fel90] Matthias Felleisen. On the Expressive Power of Programming Languages. In *Lecture Notes in Computer Science, 432.* Springer, 1990.

[Fer88] A. B. Ferguson & Philip Wadler. When will Deforestation Stop?. In *1988 Glasgow Workshop on Functional Programming.* August 1988.

[Gab92] Dov Gabbay & Ruy J. G. B. de Queiroz. Extending the Curry-Howard Interpretation to Linear, Relevant and Other Resource Logics. In *Journal of Symbolic Logic.* Vol 57, Number 4, Dec. 1992.

[Gal87] J.H. Gallier. *Logic for Computer Science. Foundations of Automatic Theorem Proving.* John Wiley & Sons, 1987.

[Gal92] J.H. Gallier. *Constructive Logics. Part I: A tutorial on Proof Systems and Typed λ-Calculi.* Unpublished Manuscript, 1992.

[Gir71] J.-Y. Girard. Une extension dy système de fonctionelles recursives de G̈del et son application aux fondements de l'analyse. In *J.E. Fenstad, editor, Proceedings of the 2nd Scandinavian Logical Symposium.* North-Holland, 1971..

[Gir89] J.-Y. Girard, P. Taylor, Y. Lafont, Proofs and Types. Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press, 1989.

[Gri90] Timothy G. Griffin, *A Formulae-as-Types Notion of Control,* in POPL 1990.

[Har92] Robert Harper & Mark Lillibridge. Polymorphic Type assignment and CPS Conversion. In *Proceedings of the ACM SIGPLAN Workshop on Continuations CW92.* San Francisco, California, 1992.

[Hen93] Fritz Henglein, Dynamic Typing. Syntax and proof theory, Submitted to ??, 1993.

[Hin86] J.R. Hindley, J.P. Seldin (eds.), *Introduction to Combinators and λ-calculus,* Cambridge University Press, 1986.

[How80] W.A. Howard, The formulae-As-types Notion of Construction. *In* J.R. Hindley & J.P. Seldin, To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press, 1980.

[Kle52] S. Kleene, Introduction to Metamathematics. Van Nostrand, 1952.

[Lei83] Daniel Leivant. Reasoning about Functional Programs and Complexity Classes Associated with type disciplines.. In *Proceedings of the 24th Annual Symposiom on the Foundations of Computer Science.* 160-169, IEEE, 1983.

[Mar70] Per Martin-Lf̈. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In *J.E. Fenstad, editor, Proceedings of the 2nd Scandinavian Logical Symposium.* North-Holland, 1971..

[Men87] Elliott Mendelson, *Introduction to Mathematical Logic,* Wadsworth 1987.

[Mil78] R. Milner. A therory of type polymorphism in programming. In *Journal of Computer and System Sciences.* 17, 1978.

[Mil90] Robin Milner & Mads Tofte & Robert Harper, The Definition of Standard ML, MIT Press, 1990.

[Mit90] John C. Mitchell, Type systems for programming languages. In J. van Leeuwen (ed.) *Handbook of Theoretical Computer Science*, Volume B. pp365-458. North-Holland, 1990.

[Mor93] Luc Morreau & ?? *Parallel Evaluation of call/cc*, FPCA '93

[Mur90] Chet Murthy, *Extracting Constructive Content From Classical Proofs*, PhD. thesis.

[Mur91] Chet Murthy. An Evaluation Semantics for Classical Proofs. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*. LICS 1991.

[Mur91b] Chet Murthy. Classical Proos as Programs: How, What and Why. In *Tecnical Report TR 91-1215, Department of Computer Science, Cornell University, July 1991.* 1991.

[Mur92a] Chet Murthy, *Classical Logic as a Programming Language: Typing Nonlocal Control*, Lecture notes 1992.

[Pie89] Benjamin Pierce, Scott Dietzen & Spiro Michaylov. *Programming in Higher-Order Typed Lambda-Calculi.* Technical Report. CMU-CS-89-111, Carnegie Mellon University, March 1989.

[Plo75] G. D. Plotkin. Call-by-Name, Call-by-Value and the λ-Calculus. In *Theoretical Computer Science.* 1, 1975.

[Pra65] Dag Prawitz, *Natural Deduction*, Almquist & Wiksell, Uppsala 1965.

[Pra71] Dag Prawitz, Ideas and results in proof theory. In J.E. Fenstad, esitor, *Proceedings of the 2nd Scandinavian Logical Symposium*, pages 235-307. North-Holland, 1971.

[Sel86] Jonathan P. Seldin. On the Proof Theory of the Intermediate Logic MH. In *Journal of Symbolic Logic.* Vol. 51, Number 3, Sept. 1986.

[Sel89] Jonathan P. Seldin. Normalization and Excluded Middle. I. In *Studia Logica.* XLVIII, 1, 1989.

[Sta91] Gunnar Stålmarck. Normalization Theorems for Full First Order Classical Natural Deduction. In *Journal of Symbolic Logic.* Vol. 56, Number 1, March 1991.

[Ste84] Guy L. Steele. *Common Lisp: The Language.* Digital Press, Bedford, MA, 1984.

[Tai67] W. Tait. Intensional Interpretation of functionals of finite type I. In *Journal of Symbolic Logic.* 32, 1967.

[Tak75] Gaisi Takeuti. *Proof Theory.* Studies In Logic And The Foundations Of Mathematics, vol 81. North-Holland, Amsterdam, 1975.

[Wad84] P. L. Wadler. Listlessness is better than lazyness: Lazy evaluation and garbage collection at compile-time. In *ACM Symposium on Lisp and Functional Programming.* Austin, Texas, 1984.

[Wad85] P. L. Wadler. Listlessness is better than lazyness II: Composing Listless functions. In *Workshop on Programs as Data objects.* Lecture notes in Computer Science 217, Copenhagen, 1985.

[Wad88] P. L. Wadler. Deforestation: Transforming programs to eliminate trees. In *European symposium On programming (ESOP).* Nancy, France, 1988.

[Wad93] P. L. Wadler & Simon Marlow. *A logical basis for Deforestation.* Unpublished handwritten manuscipt, 1993.

[Wer92] Benjamin Werner. Continuations, Evaluation Styles and Type Systems. In *(SUBMITTED) LICS '92.* 1992.