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

Abstract. By restriction of Felleisen's control operator $\mathcal{F}$ we obtain an operator Δ and a fully compatible, Church-Rosser control calculus $\lambda_\Delta$enjoying a number of desirable properties. It is shown that $\lambda_\Delta$contains a strongly normalizing typed subcalculus with a reduction corresponding closely to systems of proof normalization for classical logic. The calculus is more than strong enough to express a call-by-name `catch/throw`-programming paradigm.

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.[1]

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.[2]

[1] 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.

[2] 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.

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, discovererd 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 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.[3] 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 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* standard texts on classical proof normalization, *e.g.* [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.

Denotations for classical proofs have been developed by P. de Groote [Gro92]. However, this work is not concerned with any notion of reduction on proofs.

Upon concluding this work we became aware of recent work by M. Parigot and by A. Rezus which has similar goals to the ones in the present paper. We briefly relate their work to ours in Section 7.

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

[3] 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`.

- 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 calculi we study: the untyped λ_{Δ}, $\lambda_{\Delta P}$, and the typed $\lambda_{\Delta}^{\perp,\supset}$, $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. We also study an extension of λ_{Δ} called $\lambda_{\Delta\delta}$ which includes basic constants and basic functions. *Section 3* shows that λ_{Δ} and $\lambda_{\Delta\delta}$ are Church-Rosser and that $\lambda_{\Delta}^{\perp,\supset}$ in addition has Subject Reduction and is Strongly Normalizing. *Section 4* shows that λ_{Δ} 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* compares $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ to a standard proof-theoretical formulation of classical propositional logic and develops some classical proof-theory for $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ and $\lambda_{\Delta}^{\perp,\supset}$. *Section 6* gives a notion of definability of pairs and sums and shows that in $\lambda_{\Delta}^{\perp,\supset}$ pairs and sums can be defined.

2 The λ_{Δ}-calculus

Subsection 1 describes the construction language and reduction relation of two untyped λ_{Δ}-calculi, λ_{Δ} and $\lambda_{\Delta P}$; the latter is the extension of the former with pairs and sums. Subsection 2 describes the type language and type inference system of two corresponding typed versions, $\lambda_{\Delta}^{\perp,\supset}$ and $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$, which we call classically typed λ_{Δ}-calculi; the term language and reduction relation for the typed versions are the same as for the untyped calculi. The relation between the untyped and classically typed λ_{Δ}-calculi is roughly the same as between the untyped and simply typed λ-calculus.

2.1 The untyped λ_{Δ}-calculi: λ_{Δ}, $\lambda_{\Delta P}$

We take the usual notions of *free* and *bound variable* as well as the related customary hygiene conventions for granted. $FV(M)$ denotes the set of free variables in construction M. Substitution of construction N for all occurrences of a free variable x in a construction M is written $M\{x := N\}$. The usual conventions concerning the binding and associativity of abstraction and application are adopted. The conventions for λ carry over to 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.

$$\begin{aligned} M ::= {} & x \mid \lambda x.M \mid M\ N \mid \Delta x.M \\ & < M_1, M_2 > \mid \pi_1(M) \mid \pi_2(M) \\ & \mathrm{in}_1(M) \mid \mathrm{in}_2(M) \mid \mathrm{case}(L; x_1.M; x_2.M_2) \end{aligned}$$

This language is called Λ_P and is the language of $\lambda_{\Delta P}$. The language obtained by omitting the six last clauses is called Λ and is the language of λ_Δ.

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, see Section 4.

Definition 2. (Reduction on constructions.)

(1a)	$(\lambda x.M)\ N$	$\to M\{x := N\}$
(1b)	$\pi_i(< M_1, M_2 >)$	$\to M_i$
(1c)	$\mathrm{case}(\mathrm{in}_i(L); x_1.M_1; x_2.M_2)$	$\to M_i\{x_i := L\}$
(2a)	$(\Delta x.M)\ N$	$\to \Delta z.M\{x := \lambda y.z\ (y\ N)\}$
(2b)	$\pi_i(\Delta x.M)$	$\to \Delta u.M\{x := \lambda y.u\ \pi_i(y)\}$
(2c)	$case(\Delta x.M; y_1.M_1; y_2.M_2)$	$\to \Delta u.M\{x := \lambda y.u\ case(y; y_1.M_1; y_2.M_2)\}$
(3)	$\Delta x.x\ M$	$\to M$ provided $x \notin FV(M)$
(4)	$\Delta x.x\ \nabla(x\ M)$	$\to M$ provided $x \notin FV(M)$

These rules are the reduction rules for $\lambda_{\Delta P}$. Omitting (1b),(1c),(2b),(2c) yields the reduction rules for λ_Δ.

Each of the left hand sides is called a *redex*. Here and later we call $\to$ the *notion of reduction*. The compatible closure of $\to$ 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 $=$.[4]

2.2 The classically typed λ_Δ-calculi: $\lambda_\Delta^{\bot,\supset}$, $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$

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

Definition 3. (Type language.) 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.

$$\mathcal{P} ::= P \mid \bot \mid \mathcal{P} \supset \mathcal{Q} \mid \mathcal{P} \wedge \mathcal{Q} \mid \mathcal{P} \vee \mathcal{Q}$$

This is the type language of $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$. Omitting the last two clauses yields the type language of $\lambda_\Delta^{\bot,\supset}$.

[4] Subscribing a relation $(\to, \rhd, \rhd^*, =)$ 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 $(\lambda x.\lambda y.M)\ N\ K \rhd^*_{3,4} M\{x := N\}\{y := K\}$.

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.)

We take such notions as *derivation, open assumption, closed assumption* etc. for granted, see *e.g.* [Pra71].

Definition 4. (Type Inference system.)

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

$$\frac{M_1:\mathcal{P}_1\ \ M_2:\mathcal{P}_2}{<M_1,M_2>:\mathcal{P}_1\wedge\mathcal{P}_2}\ (\wedge I) \qquad \frac{L:\mathcal{P}_1\wedge\mathcal{P}_2}{\pi_i(L):\mathcal{P}_i}\ (\wedge E_i)$$

$$\frac{M_1:\mathcal{P}_1}{\mathrm{in}_1(M_1):\mathcal{P}_1\vee\mathcal{P}_2}\ (\vee I_i) \qquad \frac{L:\mathcal{P}\vee\mathcal{Q}\ \ \begin{array}{cc}[x_1:\mathcal{P}] & [x_2:\mathcal{Q}]\\ \vdots & \vdots\\ M_1:\mathcal{R} & M_2:\mathcal{R}\end{array}}{\mathrm{case}(L;x_1.M_1;x_2.M_2):\mathcal{R}}\ (\vee E)$$

This is the inference system of $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$. Omitting the last four rules yields the inference system for $\lambda_{\Delta}^{\bot,\supset}$.

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 $\Gamma\vdash M:\mathcal{P}$.

The I rules are called *introduction* rules and the E rules are called *elimination* rules. 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.*

For the systems $\lambda_{\Delta}^{\bot,\supset}$, $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$there is another system $\lambda^{\bot,\supset}$, $\lambda_{P}^{\bot,\supset,\wedge,\vee}$, 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 $\bot_c$. These are the simply typed λ-calculus without and with pairs and sums.

Via the Curry-Howard Isomorphism, $\lambda_{\Delta}^{\bot,\supset}$is a natural deduction formulation of the implicational fragment of classical propositional logic while $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$is the full system; $\lambda^{\bot,\supset}$and $\lambda_{P}^{\bot,\supset,\wedge,\vee}$are corresponding minimal logics. In the classical (but not the minimal) case, the implicational fragment can work as the corresponding full system through suitable definitions, see section 6.

3 Fundamental Properties of λ_{Δ}, $\lambda_{\Delta\delta}$ and $\lambda_{\Delta}^{\bot,\supset}$

In this chapter we prove that the Church-Rosser property (abbreviated CR) holds for λ_{Δ} (first section), that the properties of Subject Reduction (abbreviated SR) and CR hold for the typed calculus $\lambda_{\Delta}^{\bot,\supset}$ (second section.) We briefly discuss extensions of λ_{Δ} which include basic constants and basic functions (third section), and finally (fourth section) we prove that $\lambda_{\Delta}^{\bot,\supset}$ has the Strong Normalization property(abbreviated SN.)

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. The method is a generalization of the proof by parallel reduction for the λ-calculus (see [Bar84]), similar to the method used in [Fel89].

A one step reduction $\triangleright$ has the CR property if, whenever $M_1 \triangleright^* M_2$ and $M_1 \triangleright^* M_3$ there is an M_4 such that $M_2 \triangleright^* M_4$ and $M_3 \triangleright^* M_4$. 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 proof of the CR property uses parallel reductions which enjoy certain closure properties:

Definition 5. (Basic conditions for parallelism) Let R be any binary relation over Λ. Then we say that R satisfies the basic conditions for parallelism, abbreviated $BCP(R)$, if the following hold for R, for any $M, N, M', N' \in \Lambda$:

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

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

The overall idea of the proof is as follows. We define two parallel reductions, $\gg_\gamma$ and $\gg_\delta$, each of which represents a fragment of the calculus which has a straightforward CR-proof based on parallel reduction (see [Bar84].) These results are then combined in standard fashion by the Hindley-Rosen Lemma:

Lemma 6. *(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.

Define the parallel reduction $\gg_\gamma$ to be the least relation over Λ satisfying

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

The parallel reduction $\gg_\delta$ is the least relation over Λ satisfying [5]

(1) $BCP(\gg_\delta)$
(2) $M \gg_\delta M', N \gg_\delta N' \Rightarrow (\Delta x.M)\, N \gg_\delta \Delta\kappa.M'\{x := \lambda f.\kappa\,(f\, N')\}$

Now, we can prove :

Lemma 7. $\gg_\gamma \models \Diamond$.

Proof. Induction on the definition of $\gg_\gamma$.

Lemma 8. $\gg_\delta \models \Diamond$.

Proof. Induction on the definition of $\gg_\delta$.

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 prove:

Lemma 9. *$\gg_\gamma^*$ and $\gg_\delta^*$ commute.*

Proof. We combine the following two properties:

(I) 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 $M_2 \gg_\delta M_4$.

(II) For an arbitrary binary relation R, let R^r denote the reflexive closure of R, R^t the transitive closure of R, and R^{rt} the reflexive transitive closure of R. 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, by a diagram chase (see [Bar84], 3.3.6), we can show that R_1^{rt} and R_2^{rt} commute.

Since $\gg_\delta$ and$\gg_\gamma^*$ are both reflexive, our claim follows by (I) and (II).

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

Theorem 10. $\rhd$ *is CR.*

Proof. ¿From Lemma 7 and Lemma 8 it follows that $\gg_\gamma$ and $\gg_\delta$ are CR. By Lemma 9 $\gg_\gamma^*$ 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.

[5] 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 (see below in the text), as required by the Hindley-Rosen Lemma, goes through.

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

Proof. Straightforward.

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 $M_3 \rhd^* M_4$.[6]

Corollary 12. $\lambda_{\Delta}^{\perp,\supset}$ *has the* CR *property.*

Proof. Follows from Theorem 10 and Theorem 11.

3.3 Adding Basic Constants and Basic Functions ($\lambda_{\Delta\delta}$)

In this section we briefly sketch an extension of λ_{Δ} with basic constants and eager basic functions, called $\lambda_{\Delta\delta}$. In addition to δ-reduction on applications of basic functions to basic constants we must also add a rule for applications of basic functions to Δ-abstractions. The CR property is preserved by this extension. This result is stated but, due to space limitations, not proved here. A full proof can be found in [Reh93d].

We consider the following enriched language of constructions, called Λ_{δ}:

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

Here $b \in BConsts$ ranges over a collection of basic constants such as integers and booleans, and $\phi \in FConsts$ ranges over a collection of basic functions such as arithmetical functions and conditional. To the rules of λ_{Δ} we add the following two new rules, yielding the enriched calculus $\lambda_{\Delta\delta}$:

Definition 13.

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

As is shown in detail in [Reh93d] we have

Theorem 14. *The reduction* $\rhd^*$ *of* $\lambda_{\Delta\delta}$ *is* CR.

Proof. See [Reh93d].

[6] 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.

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).[7] 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 a system containing $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$(1-2) with a slight modification in (2c) 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 [8] (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 $\vee$ is 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. We are to show that the extra inference rule does not allow so many typed λ-terms that there are non-terminating sequences of β-reductions.

The idea is that in $\lambda_{\Delta}^{\perp,\supset}$(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 if $y\ \bullet$ is a subconstruction in a construction of form $\lambda y.M$. We exploit this idea below.

Definition 15. In the context of lists $V = [x_1, \ldots x_n], W = [y_1 \ldots y_n]$ of variables define

$$K^{+} \equiv \lambda x_1 \ldots . \lambda x_n . \lambda y_1 \ldots . \lambda y_n . K$$

For $\vdash M : \mathcal{P}$, let all the variables in M which are bound by a Δ be contained in $V = [x_1 \ldots x_n]$ (of types $\mathcal{P}_1 \ldots \mathcal{P}_n$) and let $W = [y_1 \ldots y_n]$ be variables not

[7] 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).

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

occurring in M (of types $\bot \supset \mathcal{P}_1 \ldots \bot \supset \mathcal{P}_n$.) In the context of these two lists define $\underline{\bullet}$ as follows:

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

Lemma 16. *Assume $\vdash M : \mathcal{P}$. Let all the variables in M which are bound by a Δ be contained in $V = [x_1 \ldots x_n]$ (of types $\mathcal{P}_1 \ldots \mathcal{P}_n$) and let $W = [y_1 \ldots y_n]$ be variables not occurring in M (of types $\bot \supset \mathcal{P}_1 \ldots \bot \supset \mathcal{P}_n$.) Then (1) $\vdash (\underline{M})^+ : \mathcal{P}_1 \supset \ldots \mathcal{P}_n \supset (\bot \supset \mathcal{P}_1) \supset \ldots (\bot \supset \mathcal{P}_n) \supset \mathcal{P}$ in $\lambda^{\bot,\supset}$. (2) If $M \rhd_1 N$ then $(\underline{M})^+ \rhd_1 (\underline{N})^+$.*

Proof. Easy, see [Reh93d].

Corollary 17. *If $\vdash M : \mathcal{P}$ then there is no infinite reduction sequence in $\lambda_\Delta^{\bot,\supset}$ (1) starting from M.*

Proof. By the preceding lemma.

Strong Normalization for $\boldsymbol{\lambda_\Delta^{\bot,\supset}}$(1-2) The difference between $\lambda_\Delta^{\bot,\supset}$(1-2) and $\lambda_\Delta^{\bot,\supset}$(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 $\lambda_\Delta^{\bot,\supset,\forall}$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 18. Let $\vdash M : \mathcal{P}$. Define $\underline{\bullet}$ as follows:

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

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

Lemma 19. *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. Easy, see [Reh93d].

Corollary 20. *If $\vdash M : \mathcal{P}$ then there is no infinite reduction sequence starting from M in $\lambda_{\Delta}^{\perp,\supset}$ (1-2).*

Proof. By the preceding lemma.

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 21. *(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. By structural induction, see [Reh93d].

Corollary 22. *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 23. *$\lambda_{\Delta}^{\perp,\supset}$ 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 $\lambda_{\Delta}^{\perp,\supset}$(1-2).

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

In this section we first show how Δ is related to Felleisen's operators. It is then shown that $\lambda_{\Delta\delta}$contains a Church-Rosser `catch`/`throw`-subcalculus.

4.1 λ_Δ and Felleisen's Control Operators

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 ($\mathcal{F}_R$) :

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

These rules (and analogous ones for the $\mathcal{C}$-operator [9]) 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. [10]

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 $x \notin FV(M)$)

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

---

[9] $\mathcal{C}$ 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.

[10] 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.

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.)

4.2 $\lambda_{\Delta}$and the `catch/throw`-Mechanism

The λ_{Δ}-calculus can express the `catch/throw`-mechanism with operational semantics given by a rule of the form $\texttt{catch}\,x\,E[\texttt{throw}\,x\,M] \rhd M$, where E ranges over a specified notion of evaluation contexts in which the rule is parametric. We can write λ_{Δ}-expressions in which the form $\Delta x.x\,M$ can be read as $\texttt{catch}\,x\,M$, and the form $\nabla(x\ N)$ can be read as $\texttt{throw}\,x\ N$.

While rule (2) of λ_{Δ}corresponds to Felleisen's "bubble-rules", the other two Δ-rules can typically be seen as "clean-up" operations: ¿From a `catch/throw`-perspective rule (3) allows the elimination of empty `catch`'es, thus coping with the situation of unexceptional return. The rule (4) copes with exeptional return, eliminating the `catch/throw`-pair once they have met, after the ∇ has "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. It is instructive to verify, *e.g.* , that $F\ 2 \rhd^{*} 3$, $F\ 1 \rhd^{*} 2$ and $F\ 0 \rhd^{*} 0$.

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 $\mathbf{Int} \to \mathbf{Int}$, using the intuitionistic rule to get the typings $\nabla(j\ 0) : \mathbf{Int}$ and $\nabla(k\ 1) : \mathbf{Int}$ for those subexpressions. Compare reduction of $F\ 2$ (normal return) with reduction 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. Then we can write the following 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)
```

It is instructive to verify that, *e.g.* , $\mathtt{MT} = 0$, with $\mathtt{T} \equiv \mathtt{<2,<0,nil,nil>,nil>}$, noticing how an "exception" is raised as the node value 0 is encountered.

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 24. (`catch/throw`-subcalculus $\lambda_{\mathbf{ct}}$)

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

Theorem 25. *The reduction relation of $\lambda_{\mathbf{ct}}$ is CR.*

Proof. By Lemma 7 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 with the subcalculus just mentioned. Therefore rule $(ct3)$ can be added by the Hindley-Rosen Lemma. Also, rule $(ct4)$ can be added, by an analogous argument. This accounts for the CR property.

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. In other words: *Are there computationally interesting, 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. [11]

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

$$(8)\ \Delta x.\nabla(M) \to \Delta x.M$$

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

Theorem 26. *$\trianglerighteq$ is CR. In particular, λ_Δ is incomplete.*

Proof. Since $\triangleright_8$ is evidently SN and locally confluent, it is CR. Let $\triangleright$ be the reduction of $\lambda_{\Delta\delta}$. By easy induction we can show that $M_1 \triangleright M_2$ and $M_1 \triangleright_8 M_3$ implies that there is an M_4 such that $M_2 \triangleright_8^* M_4$ and $M_3 \triangleright M_4$. Using this, we can show that $\triangleright^*$ of $\lambda_{\Delta\delta}$and $\triangleright_8^*$ commute, by an argument analogous to the one sketched in the proof of Lemma 9. Then, by the Hindley-Rosen Lemma we conclude that $\triangleright \cup \triangleright_8 = \trianglerighteq$ is CR.

Rule (8) is computationally interesting in its own right. As a special case of it we have $\nabla(\nabla(M)) \trianglerighteq \nabla(M)$, 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 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 are closely related to Fellisen's $\mathcal{F}$-operator and can express the catch-throw paradigm. In this section we show that the reduction rules for Δ are closely related to classical proof normalization rules as found in the proof theory literature and 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 a standard classical proof normalization procedure

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

and compares it to the reduction rules in $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ classically typed λ_{Δ}-calculi. Subsection 3 develops proof theory for $\lambda_{\Delta}^{\perp,\supset}$ and $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ 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.*[12] There corresponds to every proof one construction, viz. the one at the root of the proof.

Also recall that in $\lambda_{P}^{\perp,\supset,\wedge,\vee}$ the inference rules are 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 *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 followed 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[13] in a derivation in $\lambda_{P}^{\perp,\supset,\wedge,\vee}$ 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 redices in the construction representing the derivation. In terms of constructions the Inversion Theorem states that for every construction of type

[12] This should not be confused with a "proof" in the metareasoning.

[13] 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.

$\mathcal{P}$ 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 $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ as for $\lambda_P^{\perp,\supset,\wedge,\vee}$. Specifically, the symmetry in introduction/elimination rules dos not hold, and $\Delta x.M$ rather plays the role of *all* the constructors, the particular choice depending on the type of x.

A formula occurrence in a derivation or proof in $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ which is both the conclusion of an I rule or $\perp_c$ and the major premise of an E rule is also called a maximum formula. Extend the notion of normal derivation accordingly. It then still holds that there is a one-one correspondence between maximum formulae and redices. 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 $\lambda_P^{\perp,\supset,\wedge,\vee}$ by defining a certain set of reduction rules. The next subsection shows how these reduction rules have been defined in the literature.

5.2 A standard classical normalization procedure

We shall consider the system for which Stålmarck proves SN in [Sta91]. This system is first-order but we only consider the propositional fragment. In the original works, the reductions are stated from derivations to derivations. We find it more convenient to state them as reductions on constructions.

The construction and type language, and the type inference system is that of $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. The set of reductions is (1a)-(1c) from $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$ and in addition the rules described below.

The following notion will be convenient.

Definition 27. (Eliminative Contexts.)

$$\begin{aligned} E ::= {} & \pi_1([]) \mid \pi_2([]) \\ & \mid case([]; x_1.M; x_2.N) \mid case(L; x_1.[]; x_2.N) \mid case(L; x_1.M; x_2.[]) \\ & \mid M\ [] \mid []\ N \end{aligned}$$

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

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

The rules (Sa)-(Sc) consitute the heart of the system, and they are almost precisely our (2a)-(2e).[14] The only difference is in (Sc) where the application of u is moved into the branches of the case-construction. The motivation for this

[14] They consitute the heart for the following reasons. (1) The remaining rules are motivated by technicalities in the proof of SN; (2) In the next section we shall

difference as well as for adopting the rules (R1)-(R2) pertains to the method that Stålmarck applies for the SN proof.

The rule (P) is under the restriction that the subterm $case(L; x_1.M_1; x_2.M_2)$ of the left hand side is normal w.r.t. the other rules they are grouped with above. The purpose of this rule is to establish the subformula property, see [Gir89] ch. 10. It is only necessary because $\vee$ is taken as primitive.

We conclude that the core of $\lambda_P^{\perp,\supset,\wedge,\vee}$ is very similar to the core of Stålmarck's system.[15]

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 $\lambda_\Delta^{\perp,\supset}$ and $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. That this can be done shows that not only are the reduction rules of *e.g.* $\lambda_\Delta^{\perp,\supset}$ 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 $\lambda_\Delta^{\perp,\supset}$.[16]

---

see that our (2a)-(2c) are essential for deriving certain proof-theoretical corollaries; (3) the other two standard classical proof-normalization procedures that we know, Prawitz' [Pra65] and Seldin's [Sel89], both contain (generalizations of) these rules. For instance, the system in [Pra65] contains rules

$(Pa)\ \mathcal{P} \equiv \mathcal{Q} \wedge \mathcal{R} : \Delta x.(M) \to < \Delta y.M\{x := \lambda u.y\ \pi_1(u)\}, \Delta z.M\{x := \lambda v.z\ \pi_2(v)\} >$
$(Pb)\ \mathcal{P} \equiv \mathcal{Q} \supset \mathcal{R} : \Delta x.(M) \to \lambda u.\Delta y.M\{x := \lambda v.y\ (v\ u)\}$

with the property that $D^1[\Delta x.M] \rhd^* \Delta z.M\{x := \lambda y.z\ D^1[y]\}$, where $D^1 ::= [] \mid D^1\ M \mid \pi_i(D^1)$, and $D^1[M]$ denotes the result of replacing [] in D^1 by M.

[15] With the purpose of making a WN proof by induction go through, Stålmarck adopts a few more rules, which look peculiar at a first glance. These rules can, however, be perceived as on-line program transformations that "make sense" operationally, see [Reh93d].

[16] We should also extend the SN result to $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. 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 $\vee$ is 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 rule corresponding to our (2c) is slightly different from ours and unnatural in a certain sense,

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 28. Define the mutually recursive syntactic classes E, C, I:

$$\begin{array}{l} E ::= x \mid E\ I \mid \pi_i(E) \mid \mathrm{case}(E; x.I; y.I) \\ C ::= E \mid \Delta x.C \\ I\ ::= C \mid \lambda x.I \mid\ < I, I > \ \mid \mathrm{in}_i(I) \end{array}$$

Theorem 29. *(Form of Normal Deductions.) Suppose that* $\Gamma\ \vdash\ M\ :\ \mathcal{P}$ *in* $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$. *If* M *is normal then* M *conforms to* I.

Proof. Straight-forward.

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

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

Notice how the grammar above for I corresponds to the characterization of derivations in [Pra65] III §2 Theorem 3. The *only* essential 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 $(\bot_c)$ rule that the conclusion be different from $\bot$, see [Pra65] p20. Alternatively, we could have adopted a rule to get rid of such sequences, *e.g.* $\Delta x.\Delta y.M \to \Delta x.M\{y := \lambda z.z\}$.

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 30. *(Consistency.) There is no proof of* $\bot$ *in* $\lambda_{\Delta P}^{\bot,\supset,\wedge,\vee}$.

We take the notions of positive and negative occurrences for granted.

---

and it is not clear whether one can change the idea in the proof to cope with our rule (2c).

Corollary 31. *(Craig-Lyndon's Interpolation Theorem.) Suppose that* $\Gamma \vdash M : \mathcal{P}$ *in* $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. *Then there is an interpolation formula* $\mathcal{Q}$ *such that* $\Gamma \vdash L : \mathcal{Q}$ *and* $x : \mathcal{Q} \vdash K : \mathcal{P}$ *in* $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$, *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 $\lambda_{\Delta}^{\perp,\supset}$. For well-known problems caused by disjunction (see [Gir89] ch. 10) the following result holds in $\lambda_{\Delta}^{\perp,\supset}$, but not in $\lambda_{\Delta P}^{\perp,\supset,\wedge,\vee}$. The proof can be adapted from [Pra65] right away.

Corollary 32. *(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:

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

The overall derivation starts with a derivation $M : \mathcal{P}$. It then turns this into a derivation $k\ M : \perp$ from assumption $k : \neg\mathcal{P}$. This is essentially the same as a derivation of $\neg\neg\mathcal{P}$ with no assumptions. This, in turn, is turned into a derivation, once again, of $\mathcal{P}$ with no assumptions. The reduction rule thus states that turning a proof into a proof of the double-negated formula and back again are inverse operations. This is a kind of classical inversion principle.

Rule (4) can be justified similarly.

6 Definability

We are not aware of any generally accepted definition of definability of constructions. In [Reh93d] we have synthesized the following definition from definitions of definability in logic (*weak* and *strong definability* of connectives [Pra65] pp58-59) and λ-calculus (*pairing construction* [Bar84] P567.) The definition says that the desired inference and reduction rules can be simulated.

For a connective γ a γ-compositional translation $\underline{\bullet}$ means a translation from formulae to formulae such that $\underline{\mathcal{P}\delta\mathcal{Q}} \equiv \underline{\mathcal{P}}\delta\underline{\mathcal{Q}}$ when δ is not γ and such that $\underline{\mathcal{R}}$ does not contain γ. 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.

Definition 33. (Definability of pairs, sums.) Let K be one of $\lambda^{\bot,\supset}$, $\lambda_{\Delta}^{\bot,\supset}$. A *pairing* in K is a $\wedge$-compositional translation on formulae and pair-compositional translation on constructions such that

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

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

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

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

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

Given a translation on formulae, the corresponding translation of constructions is naturally induced by an induction proof that the desired inference rules hold derived.

It is well-known that in $\lambda^{\bot,\supset}$no connective is weakly definable (and therefore not strongly definable either) [Pra65] p59. Further, one cannot define a pairing in $\lambda^{\bot,\supset}$[Bar84] p567. On the other hand it is well-known that both conjuction and disjunction are strongly definable in $\lambda_{\Delta}^{\bot,\supset}$(see *e.g.* [Men87].) The question now is whether this result extends to the definition of pairing and summing above which is concerned with constructions also.

Definition 34. Let $\underline{\bullet}$ be the $\wedge$- and $\vee$-compositional translation on formulae, and pair- and sum-compositional translation on constructions defined below.

$$\begin{aligned} \underline{\mathcal{P}_1 \wedge \mathcal{P}_2} &\equiv \neg(\underline{\mathcal{P}_1} \supset \neg\underline{\mathcal{P}_2}) \\ \underline{\mathcal{P}_1 \vee \mathcal{P}_2} &\equiv \neg\underline{\mathcal{P}_1} \supset \neg\neg\underline{\mathcal{P}_2} \end{aligned}$$

$$\begin{aligned} \underline{< M_1, M_2 >} &\equiv \lambda f.f\ \underline{M_1}\ \underline{M_2} \\ \underline{\pi_i(M)} &\equiv \Delta k.\underline{M}\ \lambda x_1.\lambda x_2.k\ x_i \\ \\ \underline{\mathrm{in}_i(M)} &\equiv \lambda y_1.\lambda y_2.y_i\ \underline{M} \\ \underline{\mathrm{case}(M; x_1.N_1; x_2.N_2)} &\equiv \Delta k.\underline{M}\ (\lambda x_1.k\ \underline{N_1})\ (\lambda x_2.k\ \underline{N_2}) \end{aligned}$$

The definition of conjunctive formulae is standard in logic (see *e.g.* [Men87].) The corresponding definition for pairing and projection has appeared in a typed setting in [Gri90]. The pairing construction is also 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.[17] 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:

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

This shows how Δ works as a subtyping operator.

The definition for disjunctive formulae is *not* standard in logic. The standard definition, also adopted in [Gri90], is $(\neg\mathcal{P}_1) \supset \mathcal{P}_2$. However, when one tries to prove the derived inference rules for this translation it turns out that the corresponding constructions for injection and case analysis are very different from those defining pairs and projections. Specifically, to have the desired reduction rule hold derived one would need to add extra power to Δ.[18] 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 below.)

Theorem 35. $\underline{\bullet}$ *defined above is a pairing and summing in* $\lambda_\Delta^{\bot,\supset}$.

Proof. Straight-forward.

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] 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 $\lambda_\Delta^{\bot,\supset}$. This suggests that call-by-value versions of control operators are inferior to corresponding call-by-name operators in defining constructions.

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

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

[17] 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 the simply typed λ-calculus.

[18] This was not a problem in [Gri90] because Griffin's $\mathcal{C}$ is much stronger than our Δ.

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

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 $\lambda_{\Delta}^{\perp,\supset}$ where instead of blowing up the result type of pairing to a universal 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 $\lambda_{\Delta}^{\perp,\supset}$ in the case of sums.

7 Related Work

Upon completion of this work two lines of closely related work was brought to our attention.

M. Parigot has in a series of papers [Par91], [Par92], [Par93a], [Par93b] studied a so-called $\lambda\mu$ calculus. This system incorporates both first order classical logic and second order logic, presented as a so-called free deduction system comprising features of both natural deduction and sequent calculus. Church Rosser and Strong Normalization theorems are proved for variants of the $\lambda\mu$ calculus. The classical feature of the calculus is contained in a new operator μ which can apparently express our Δ.

A. Rezus [Rez90], [Rez92] has studied various λ-calculus based presentations of classical proof theory. Again, this leads to the introduction of a new operator, γ, capturing the classical features of the system. Church Rosser and Strong Normalization theorems are proved for variants of the system. While γ and Δ are closely related, the exact relation is not clear to us.

In comparison with both of these lines of work, the present work has a stronger focus on modelling faithfully control operators of programmimg languages. Moreover, in comparison with the work by M. Parigot, we remain within the usual framework for classical natural deduction proof normalization procedures.

8 Further Work

There are numerous questions one could ask at this point.

1. The question whether there are interesting extensions of λ_{Δ} corresponding to referentially transparent operators strictly more powerful than the `catch/throw`-mechanism should be investigated.

2. An operational semantics (big step semantics, evaluation function) should be defined for the Δ-operator (or an extension thereof), and Plotkin's program should be carried out (standardization, machine-calculus correspondence.) Given a notion of operational equivalence we can ask whether the addition of Δ yields an operationally conservative extension.
3. Presumably some extension of λ_Δ has a twin call-by-value calculus. It would be interesting to find such twins and compare them.
4. Assuming we have a programming language corresponding to λ_Δ, give a CPS-semantics for that language.
5. It would appear that the λ_Δ-calculus can be extended with various reduction rules under preservation of the Church-Rosser property. It would be interesting to find *complete* extensions of the λ_Δ-calculus.[19]
6. It would be interesting to extend the set of reduction rules for the λ_Δ-calculus so that one can represent numerals and at least all primitive recursion functions on numerals. This does not appear to have been done in a pure first-order setting before.

Acknowledgements. We would like to thank our referee for directing our attention to the works of M. Parigot. Thanks to Chet Murthy for referring us to the works by A. Rezus and for explaining details of his work. We also wish to thank Phillip Wadler for his interest in this work. Finally, it is a pleasure to acknowledge our debt to Fritz Henglein who supported this work with encouragement, intuition and inspiration.

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.

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

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

[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.

[19] Recall that a system of reduction rules is complete if it cannot be strictly extended under preservation of the Church-Rosser property.

[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.

[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 Gödel 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.

[Gro92] Philippe de Groote. Denotations for Classical Proofs. Preliminary Results. In *Lecture Notes in Computer Science 624.* 1992.

[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.

[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.

[Mar70] Per Martin-Löf. 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.

[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 & Daniel Ribbens, Sound Rules Parallel Evaluation of a Functional Language with callcc, *FPCA,* 1993.

[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.

[Par91] Michel Parigot. Free Deduction : An Analysis of "Computations" in Classical Logic. In *Lecture Notes in Computer Science 592.* 1991.

[Par92] Michel Parigot. $\lambda\mu$-calculus : An Algorithmic Interpretation of Classical Natural Deduction. In *Lecture Notes in Computer Science 624*. 1992.

[Par93a] Michel Parigot. Classical Proofs as Programs. In *Lecture Notes in Computer Science 713*. 1993.

[Par93b] Michel Parigot. Strong Normalization for Second Order Classical Natural Deduction. In *Conf. Logic in Computer Science (to appear)*. 1993.

[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.

[Reh93d] Jakob Rehof & Morten Heine Sørensen, The λ_Δ-calculus. Technical Report D-175, DIKU 1993. 1993.

[Rez90] Adrian Rezus. *Classical Proofs (λ-calculus Methods in Elementary Proof Theory)*. Unpublished Manuscript,1990.

[Rez92] Adrian Rezus. *Beyond BHK*. Unpublished Manuscript,1992.

[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.

[Wer92] Benjamin Werner. *Continuations, Evaluation Styles and Type Systems*. 1992. Unpublished manuscript.

This article was processed using the LaTeX macro package with LLNCS style