<table>
<thead>
<tr>
<th>Title</th>
<th>Calculus of Classical Proofs from Programming Viewpoint (New Aspects in Non-Classical Logics and Their Kripke Semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Fujita, Ken-etsu</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (1997), 1010: 7-34</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1997-08</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/61521">http://hdl.handle.net/2433/61521</a></td>
</tr>
<tr>
<td>Right</td>
<td>Type</td>
</tr>
<tr>
<td>Textversion</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td></td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
Calculus of Classical Proofs from Programming Viewpoint

Ken-etsu Fujita (藤田 憲悦)
Kyushu Institute of Technology
Department of Artificial Intelligence
Iizuka, 820, Japan
Tel: +81-948-29-7622
Fax: +81-948-29-7601
email: fujiken@dunbo.ai.kyutech.ac.jp

Abstract

We provide a natural deduction systems $\lambda_{exc}$ of classical propositional logic and prove proof theoretical and computational properties of the system. The introduction of $\lambda_{exc}$ is a consequence of our observations of the existence of a special form of cut-free proofs in $LK$, which we call $LJK$ proofs with invariants. We first show the existence of $LJK$ proofs with invariants for any classical theorem. Although $LJK$ proofs are classical proofs, they have the disjunction property in some sense, and we can derive a general form of Glivenko's theorem from them. We show the following property: a strict fragment of $\lambda_{exc}$ that is complete with respect to classical provability; a translation from arbitrary proofs to $LJK$ proofs; the Church-Rosser property and the Strong Normalization of $\lambda_{exc}$; and an isomorphism between $\lambda_{exc}$ and Parigot's $\lambda\mu$-calculus. Secondly, we introduce a call-by-value version of $\lambda_{exc}$ and prove the following properties: the Church-Rosser property; the CPS-translation from $\lambda^v_{exc}$ to $\lambda^v$ and its correctness; and a computational use of the logical inconsistency in $\lambda^v_{exc}$, extended with a certain signature.

1 Introduction

The computational meaning of proofs has been investigated in the wide field of not only intuitionistic logic [Howa80][HN88][Naka95] and constructive type theory [NPS90] but also classical logic [Gri90][Murt91][Pari92][BB93][RS94] and modal logic [Koba93]. Algorithmic contents of proofs can be applied to obtain correct programs in the sense of satisfying logical specifications. In this paper, our motivation is to study a computational aspect of a simple classical natural deduction system based on our proof theoretical observations of a special form of cut-free proofs in $LK$, and to apply such a proof theoretical property to programming via the Curry-Howard isomorphism.

In sequent calculi, we can usually distinguish classical systems and intuitionistic systems by a cardinal restriction on the right side of the sequent [Szab69][Take87]. Especially in some systems like $LJ$ [Mae54], the Beth-tableau system in [TD88], and $IL^>$ [Sche91], this restriction is critical. We first show that at most two kinds of formulae on the right side are enough to prove arbitrary theorems in classical propositional logic. To verify
this, we introduce the notion of $LJK$ proofs with invariants\footnote{The terminology of $LJK$ proofs in this paper was called $\mu$-head form proofs in [Fuji97-1][Fuji97-2]. Both denote the same style of proofs.}. On the other hand, structural rules in logic are so important and fundamental that they drastically change logical systems without logical symbols and the decidability of logical systems depend on them. This notion is obtained by carefully considering the use of right contraction rules. Careful consideration naturally leads to separation of the succedent into two parts, i.e., a contraction allowed part and a forbidden part. In one of them, we can expect some disjunction property. We discuss that the right contraction rules can be applied to certain subformulas among the given formulae. The subformulas to which the right contraction rules are applied are specified in terms of the notion “invariant” of $LJK$ proofs. Moreover, the invariant notion plays an important role in embedding classical proofs into intuitionistic proofs. That is, depending on the invariant we have distinct embeddings.

Simple examples of $LJK$ proofs (to be defined later) of the Peirce’s law are given below. The following $proof1$ will be called an $LJK$ proof of $((A \supset B) \supset A) \supset A$ with an invariant $A$, and $proof2$ an $LJK$ proof with an invariant $((A \supset B) \supset A) \supset A$. In $LJK$ proofs, the right side of each sequent is such that every occurrence of the right side, except for at most one occurrence, is the same as the invariant. From $proof1$, one can easily obtain $\neg A \rightarrow ((A \supset B) \supset A) \supset A$ and $\rightarrow ((A \supset B) \supset A) \supset \neg \neg A$ in $LJ$, respectively. In $proof2$, the application of the right contraction rule is delayed to the end, and the proof is translated into a proof of $\neg((A \supset B) \supset A) \rightarrow ((A \supset B) \supset A) \supset A$ in $LJ$, which is a consequence of Glivenko’s theorem.

$proof1$:

$$
\frac{A \rightarrow A}{A \rightarrow A, B} \rightarrow w
\frac{\rightarrow A, A \supset B \rightarrow f}{(A \supset B) \supset A \rightarrow A} \rightarrow\rightarrow
\frac{(A \supset B) \supset A \rightarrow A}{(A \supset B) \supset A \rightarrow A} \rightarrow c
\frac{(A \supset B) \supset A \rightarrow A}{((A \supset B) \supset A) \supset A} \rightarrow\rightarrow
$$

$proof2$:

$$
\frac{A \rightarrow A}{(A \supset B) \supset A, A \rightarrow A} \rightarrow f
\frac{A \rightarrow ((A \supset B) \supset A) \supset A}{A \rightarrow ((A \supset B) \supset A) \supset A} \rightarrow w
\frac{A \rightarrow ((A \supset B) \supset A) \supset A, B}{(A \supset B) \supset A \rightarrow ((A \supset B) \supset A) \supset A} \rightarrow\rightarrow
\frac{(A \supset B) \supset A \rightarrow ((A \supset B) \supset A) \supset A}{(A \supset B) \supset A \rightarrow ((A \supset B) \supset A) \supset A} \rightarrow\rightarrow
\frac{(A \supset B) \supset A \rightarrow ((A \supset B) \supset A) \supset A}{((A \supset B) \supset A) \supset A} \rightarrow c
$$

The existence of $LJK$ proofs with invariants is important not only in formal logic but also in programming based on the notion of proofs-as-programs. The notion of $LJK$ proofs makes it possible to construct a binary-conclusion classical natural deduction system. The system is a natural extension of intuitionistic natural deduction $NJ$ with at most two consequences [Fuji94]. Moreover, $LJK$ proofs are useful for embedding classical proofs into intuitionistic proofs [Fuji95], and Glivenko’s theorem is obtained as one of the by-products.

Section 2 is devoted to preliminaries. In Section 3, we introduce a sequent calculus $LJK$ and prove proof theoretical properties of the system.
In Section 4, according to the existence of LJK proofs, we provide a simple natural deduction system $\lambda_{exc}$ of classical propositional logic using the classical rule of a variant of the excluded middle. In $\lambda_{exc}$, we study a computational property of classical proofs, and discuss the meaning of the existence of LJK proofs from a programming viewpoint. We show a direct translation from any proof in $\lambda_{exc}$ to LJK proofs. We also prove that $\lambda_{exc}$ has the Church-Rosser property. Finally, a comparison with the related work; Parigot's $\lambda\mu$-calculus, $\lambda_{\Delta}$ of Relfol & Sorensen, and Felleisen's $\lambda_c$ is given to make clear a relation and distinction to the other, by which we obtain an isomorphism between $\lambda_{exc}$ and the $\lambda\mu$-calculus, and the Strong Normalization of $\lambda_{exc}$.

In Section 5, we introduce a call-by-value version of $\lambda_{exc}$, which is called $\lambda_{exc}^v$. We prove that $\lambda_{exc}^v$ has the Church-Rosser property in Section 6. In Section 7, we provide the CPS-translation of $\lambda_{exc}^v$-terms and show the correctness of the translation with respect to conversions. In Section 8, we extend $\lambda_{exc}^v$ with a signature so that a computation in type-free $\lambda$-calculus can be simulated in a system that becomes logically inconsistent. In Section 9, we briefly investigate the relation to some existing systems: $\lambda_{ern}$ of de Groot[Groo95] and Felleisen's $\lambda_c$[FFKD86][FH92]. Section 10 is devoted to concluding remarks and remaining problems.

2 Preliminaries

To define a candidate for invariants to which only the right contraction is applied, we resolve a formula into its components and assumptions, as done in tableau systems [NS93]. This decomposition method can give the candidates strictly positive subformulae of the given formula with respect to $\supset$ and $\land$, and it gives the corresponding assumptions.

Definition 1 (Resolution of Formula) Let $\Gamma$ be a sequence of formulae. The rewriting relation $\Rightarrow$ is defined as follows.

$$(\Gamma, A_1 \supset A_2) \Rightarrow (\Gamma, A_1, A_2);$$
$$(\Gamma, A_1 \land A_2) \Rightarrow (\Gamma, A_1); (\Gamma, A_2).$$

Definition 2 (Candidate for Invariants and Assumption List) Given a formula $A$, then by the above method resolve the formula starting from $(\[, A)$ such that $P_0 \equiv (\[, A)$, $P_1 \Rightarrow \cdots \Rightarrow P_k \equiv (\Gamma_k), A_k)$, $k \equiv (\Gamma_k, A_k); (\Gamma_k, A_k); \cdots;(\Gamma_n, A_n) \Rightarrow \cdots \Rightarrow P_l \equiv (\Gamma_{n1}, A_{n1}); (\Gamma_{n2}, A_{n2}); \cdots;(\Gamma_{nm}, A_{nm}).$ This process clearly terminates, and we collect all the second elements by proj2 in each $P_i$, i.e., $\text{proj}_2(P_0) = [A], \cdots, \text{proj}_2(P_k) = [A_{k1}, \cdots, A_{kn}], \cdots, \text{proj}_2(P_l) = [A_{n1}, \cdots, A_{nm}]$. The candidate for the invariants $CI(A)$ is defined as a finite list such that $\text{proj}_2(P_0), \text{proj}_2(P_1), \cdots, [A_{k1}, \cdots, A_{kn}], \cdots, [A_{n1}, \cdots, A_{nm}]$. For each $[A_{k1}, \cdots, A_{kn}]$ in $CI(A)$, the assumption list $\text{Assume}([A_{k1}, \cdots, A_{kn}], A)$ is defined as $[\Gamma_{k1}, \cdots, \Gamma_{kn}]$ taking the corresponding assumptions.

It is clearly stated that for each $P_k \equiv (\Gamma_k), A_k; (\Gamma_k), A_k; \cdots; (\Gamma_n, A_n)$ on the resolution of $A$, LK derives $\vdash A$ from $\Gamma_k \vdash A_{k1}, \cdots$, and $\Gamma_n \vdash A_n$. For example, let $A$ be $(\neg B \supset B) \land (C \lor \neg C)$. $CI(A) = [[A], [[\neg B \supset B], C \lor \neg C], [B, C \lor \neg C]].$ $\text{Assume}([B, C \lor \neg C], A) = [[\neg B, ], [\neg B]].$ Let Peirce be $((A \supset B) \supset A) \supset A$. $\text{Assume}([A], Peirce) = ([A \supset B] \supset A)$ and $\text{Assume}([Peirce], Peirce) = [\]$. Here the candidate $[D]$ is called the innermost invariant with respect to the formula Peirce, and $[Peirce]$ is the outermost invariant. It will be clear that all of the candidates can be invariants of LJK proofs, namely, the right contraction rules can be applied only for one of them if the given formula is provable.
Since we cannot use Glivenko’s theorem, which is derived as a corollary, we first consider the problem of calculating truth tables of classical theorems in intuitionistic logic\(^2\). For instance, Peirce is a classical theorem. However, \(A \rightarrow \neg A \rightarrow \neg A\) is derivable in \(LJ\), respectively. Let \(\text{Literal}(\Gamma)\) be a sequence consisting of literals using all distinct propositional letters in \(\Gamma\). For example, \(\text{Literal}(\text{Peirce})\) is \(A, B\) or \(A, \neg B\) or \(A, B\) or \(\neg A, \neg B\). Then the problem of calculating truth tables in \(LJ\) is stated as follows:

**Lemma 1 (Calculating Truth Tables in \(LJ\))** If \(\Gamma \rightarrow A\) is provable in the propositional fragment of \(LK\), then \(\Gamma, \text{Literal}(\Gamma, A) \rightarrow A\) is provable in \(LJ\) for any \(\text{Literal}(\Gamma, A)\).

*Proof.* It is enough to show that “\(\text{Literal}(A) \rightarrow A \lor \neg A\) in \(LJ\)” implies “if \(A\) in \(LK\), then \(\text{Literal}(A) \rightarrow A\) in \(LJ\)”, which is proved *without* the use of Glivenko’s theorem. \(\Box\)

### 3 Sequent Calculus \(LJK\)

Usually \(LJ\) is defined as a subsystem of \(LK\) by a cardinality restriction on the succedent. However, to specify \(LJK\) proofs, we introduce a sequent calculus obtained by combining \(LJ\) and \(LK\) such that an intuitionistic part and a classical part are distinguished in the succedent. A sequent of the system \(LJK\)\(^3\) has the form of \(\Gamma \rightarrow \Delta; [A]\), where \(\Delta\) consists of at most one occurrence, and \([A]\), which will be called an invariant, consists only of a finite number of the occurrence of \(A\), including empty. The succedent consists of two parts, that is, the first part before the semicolon has at most one occurrence and the contraction is forbidden, roughly speaking, simulating intuitionistic proofs. The second part only has the right contraction. Our intuition behind this sequent calculus is that sequential intuitionistic proofs can be combined into a proof of any classical theorem by means of the right structural rules.

\(LJK\):

(Axioms)

\[ B \rightarrow B; \]

(Structural Rules)

\[
\begin{align*}
\frac{\Gamma \rightarrow \Delta; [A]}{C, \Gamma \rightarrow \Delta; [A]} & \quad (w \rightarrow) \\
\frac{\Gamma \rightarrow ; [A]}{\Gamma \rightarrow B; [A]} & \quad (\rightarrow w_i) \\
\frac{\Gamma \rightarrow \Delta; [A]}{\Gamma \rightarrow \Delta; A, [A]} & \quad (\rightarrow w_c)
\end{align*}
\]

\[
\begin{align*}
\frac{C, C; \Gamma \rightarrow \Delta; [A]}{C, \Gamma \rightarrow \Delta; [A]} & \quad (c \rightarrow) \\
\frac{\Gamma \rightarrow \Delta; A, A, [A]}{\Gamma \rightarrow \Delta; A, [A]} & \quad (\rightarrow c)
\end{align*}
\]

\[
\begin{align*}
\frac{\Gamma, C, D, \Pi \rightarrow \Delta; [A]}{\Gamma, D, C, \Pi \rightarrow \Delta; [A]} & \quad (e \rightarrow)
\end{align*}
\]

\[
\begin{align*}
\frac{\Gamma \rightarrow A; [A]}{\Gamma \rightarrow ; A, [A]} & \quad (\rightarrow s_c) \\
\frac{\Gamma \rightarrow A, [A]}{\Gamma \rightarrow A; [A]} & \quad (\rightarrow s_i)
\end{align*}
\]

---

\(^2\)Professor Hiroakira Ono explained this problem.

\(^3\)The notion of \(LJK\) proofs was introduced independently of Girard’s \(LC\) [Gira91] and \(LU\) [Gira93]. However, \(LJK\) could be regarded as a fragment of \(LC\).
\[
\frac{Γ → B; [A_1] B, Π → Δ; [A_2]}{Γ, Π → Δ; [A_1], [A_2]} \quad (cut_i)
\]
\[
\frac{Γ → Δ; [A_1] A, Π → ; [A_2]}{Γ, Π → Δ; [A_1], [A_2]} \quad (cut_e)
\]

(Logical Rules)

\[
\frac{C, Γ → Δ; [A]}{C ∧ D, Γ → Δ; [A]} \quad (∧ →_1)
\]
\[
\frac{D, Γ → Δ; [A]}{C ∧ D, Γ → Δ; [A]} \quad (∧ →_2)
\]

\[
\frac{Γ → B; [A]}{Γ → B ∧ C; [A]} \quad (→ ∧)
\]

\[
\frac{C, Γ → Δ; [A] D, Γ → Δ; [A]}{C ∨ D, Γ → Δ; [A]} \quad (∨ →)
\]

\[
\frac{Γ → B; [A]}{Γ → B ∨ C; [A]} \quad (→ ∨_1)
\]
\[
\frac{Γ → C; [A]}{Γ → B ∨ C; [A]} \quad (→ ∨_2)
\]

\[
\frac{Γ → B; [A_1] C, Π → Δ; [A_2]}{B ⊃ C, Γ, Π → Δ; [A_1], [A_2]} \quad (→ ⊃)
\]
\[
\frac{Γ → B; [A]}{B, Γ → ; [A]} \quad (∼ →)
\]

\[
\frac{Γ → B; [A]}{¬B, Γ → ; [A]} \quad (¬ →)
\]

\[
\frac{Γ → B; [A]}{B, Γ → ; [A]} \quad (→ ¬)
\]

Definition 3 (LJK Proofs with Invariants) An LJK proof of $Γ → Δ; [A]$ with a set of invariants $Ψ$ denoted by $P_Ψ : Γ → Δ; [A]$ is defined by a proof of the sequent in LJK such that a set of formulae $Ψ$ denotes all formulae appearing in each succedent after the semicolon throughout the proof of the sequent, that is, $Ψ$ is a collection of all $A_i$'s such that for some $Γ'$ and $Δ'$, a sequent $Γ' → Δ'; [A_i]$ appears in the proof. ⊢

By the above definition, LJK proofs with empty invariants can be identified with LJ proofs. As a variant of LJK, it is also possible to construct a sequent calculus with at most two occurrences in the succedent part [Fuji97-2], which is complete with respect to classical provability in the propositional case. A sequence $¬Ψ$ denotes a sequence in some order obtained by all negated formulae in $Ψ$. 

Lemma 2 (Embedding of LJK Proofs) If we have $P_Ψ : Γ → Δ; [A]$ in LJK, then $Γ, ¬Ψ → Δ$ is provable in LJ.

Proof. By induction on the derivation. ⊢

By the contraposition of this lemma, we can check which subformula of the theorem can be invariant. For instance, in the case of Peirce’s law there are only two invariants among the theorem, that is, proof1 and proof2 in the introduction.

Let $Γ/¬A$ be a sequence of deleting all the formulae $¬A$ from $Γ$. The following lemma plays an important role in our discussion.
Lemma 3 (From LJ Proofs to LJK Proofs) If $\Gamma \rightarrow B$ is provable in LJ, then $\Gamma \vdash \neg A \rightarrow B; A$ is provable with an invariant $A$ in LJK. Especially, cut-free LJK proofs with some invariant are obtained from cut-free LJ proofs.

Proof. By induction on the derivation. □

Corollary 1 (Cut-Free LJK Proofs) If we have $P_{\{A\}} : \Gamma \rightarrow \Delta; \{A\}$ in LJK, then there exists a cut-free LJK proof of $\Gamma \rightarrow \Delta; A$ with the invariant $A$.

Proof. From the above two lemmata and the cut-elimination property of LJ. □

Theorem 1 (Existence of LJK Proofs) If we have $\Gamma \rightarrow A$ in LK, then for any $\Psi$ in CI($A$), there is a cut-free LJK proof of $\Gamma \rightarrow A$; with invariants $\Psi$.

Proof. Let $[A_1, \ldots, A_n]$ be $\Psi$ in CI($A$) and Assume($[A_1, \ldots, A_n], A$) be $[\Pi_1, \ldots, \Pi_n]$. If $\Gamma \rightarrow A$ in LK, then by the observation of Definition 2, we have $S_1 : \Gamma, \Pi_1 \rightarrow A_1, \ldots$, and $S_n : \Gamma, \Pi_n \rightarrow A_n$ in LK, and moreover, LK derives $\Gamma \rightarrow A$ from $S_1, \ldots, S_n$. Here, we consider $S_i$ for $1 \leq i \leq n$ whose succedent is not of the form of negation, since the provability is the same in propositional LK and LJ. From Lemma 1 (Calculating truth tables), if $\Gamma, \Pi_i, \rightarrow A_i$ in LK, then $\Gamma, \Pi_i, Literal(\Gamma, \Pi_i, A_i) \rightarrow A_i$ in LJ for any Literal($\Gamma, \Pi, A_i$). Let $\Gamma, \Pi, A_i$ consist of $n$ kinds of propositional letters. Then there are $2^n$ possibilities of Literal($\Gamma, \Pi, A_i$). Hence, $2^n - 1$ applications of the cut rules lead to an LJK proof of $\Gamma, \Pi_i \rightarrow ; A_i$ with an invariant $A_i$, and it is to be cut-free by Corollary 1. Thus $\Gamma \rightarrow A$; with invariants $[A_1, \ldots, A_n]$ derived from them. □

According to Lemma 2 and Theorem 1, in the case of the outmost invariant we obtain Glivenko’s theorem. The next corollary shows that the succedent part before the semicolon has the disjunction property in this calculus.

Corollary 2 (Disjunction Property) If $\rightarrow B_1 \lor B_2; \{A\}$ is provable with an invariant $A$ in LJK, then either $\rightarrow B_1; A$ or $\rightarrow B_2; A$ is provable with the invariant $A$ in LJK.

Proof. From the above two lemmata and the disjunction property of LJ, since $\neg A$ is a Harrop formula. □

The notion of invariants gives a general form of Glivenko’s theorem in the sense that if $\Gamma \rightarrow A$; is provable with invariants $\Psi$, then a formula obtained by replacing each invariant $A_i \in \Psi$ in $A$ with $\neg \neg A_i$ is also provable from $\Gamma$ in LJ. The obtained formula is denoted by $A^\Psi$. For instance, see proof 1 and proof 2 in the introduction.

Proposition 1 (Double-Negation Translation) If $\Gamma \rightarrow A$ is provable in LK, then $\Gamma \rightarrow A^\Psi$ is provable in LJ for any $\Psi$ in CI($A$).

This proposition gives another double negation translation depending on the invariants, namely, which subformulae of the theorem are applied by the right contraction rules. It could be considered as a general form of Glivenko’s theorem; however, the embedded formulae by distinct invariants become intuitionistically equivalent since $A \supset \neg \neg B \leftrightarrow \neg \neg (A \supset B)$ and $\neg A \land \neg \neg B \leftrightarrow \neg A \land B$ in LJ. The notion of invariants explains the double-negation of strictly positive subformulae with respect to $\supset$ and $\land$ gives an embedding into LJ.
4 Application to Programming

In constructive programming, one can use proofs of logical specifications as programs satisfying the specifications [HN88][NPS90]. The constructive proofs are deduced in systems based on intuitionistic logics or constructive type systems. It has become well-known by the work of Griffin [Grif90], Murthy [Murt91], etc., that classical proofs of $\Pi^0_2$ statements can be interpreted as programs with control operators. Based on the Curry-Howard isomorphism [Howa80], the key notion of $LJK$ proofs also provides a simple method to obtain exception-handling programs. According to our discussion in the previous section, we present a simple classical natural deduction system $\lambda_{exc}$ and analyze the computational content of the proofs. It will be observed that an invariant computationally plays the role of a type of exceptional parameter. We give a translation from any proof in $\lambda_{exc}$ to a certain proof in $\lambda_{exc}$, which corresponds to the notion of $LJK$ proofs with invariants. We also prove that $\lambda_{exc}$ has the Church-Rosser property. The Strong Normalization of $\lambda_{exc}$ is obtained as one of the by-products from the existence of an isomorphism between $\lambda_{exc}$ and Parigot’s $\lambda\mu$-calculus [Pari92].

4.1 Natural Deduction System $\lambda_{exc}$

According to the proofs of Theorem 1, Lemma 2, and Lemma 3, we restate the following proposition, which is applied to obtain a classical proof from intuitionistic proofs. This proposition can be regarded as a form of a generalized Glivenko’s theorem in the sense of [Seld89].

Proposition 2 Let $[A_1, \ldots, A_n]$ be in $CI(B)$, and Assume($[A_1, \ldots, A_n], B$) be $[[\Pi_1], \ldots, [\Pi_n]]$. $\Gamma \rightarrow B$ in $LJ$ iff $\Gamma, \Pi_i, \neg A_i \rightarrow A_i$ in $LJ$ for $1 \leq i \leq n$.

This approach would be different from the existing ones in the sense that classical proofs are derived from two intuitionistic proofs by applying the classical cut-rules with the invariant $A_i$, or equivalently the excluded middle. From now on, we consider the implication fragment of the system for simplicity. Hence, each list of invariants consists of one element. Then Proposition 2 shows that we can derive a classical proof of $\Gamma \rightarrow B$ from an intuitionistic proof of $\Gamma, \Pi, \neg A \rightarrow A$. According to this result, we present a classical natural deduction system and analyze the computational meaning of proofs in this system. The types are usually defined by type variables, a constant $\bot$ and $\rightarrow$. The terms are defined by two kinds of variables $x$ and $y$, where $y$ is used only for negation types $\neg A$ defined as $A \rightarrow \bot$. $FV(M)$ stands for the set of free variables in $M$.

$\lambda_{exc}$:
Types
$A ::= a | \bot | A \rightarrow A$
Contexts
$\Gamma ::= \langle \rangle | x : A, \Gamma | y : \neg A, \Gamma$
Terms
$M ::= x | \lambda x. M | y M | M M | raise(M) | [y : \neg A] M$
Type Assignment
$\Gamma \vdash x : \Gamma(x)$
$\Gamma \vdash \lambda x. M : A \rightarrow B$ ($\rightarrow I$)
\[
\frac{\Gamma \vdash M_1 : A \rightarrow B \quad \Gamma \vdash M_2 : A}{\Gamma \vdash M_1 M_2 : B} \quad (\rightarrow E) \]  
\[
\frac{\Gamma \vdash M : A}{\Gamma \vdash \text{raise}(M) : A} \quad (\perp E) \text{ if } A \neq \perp 
\]

\[
\frac{\Gamma, y : \neg A \vdash M : A}{\Gamma \vdash [y : \neg A] M : A} \quad (\text{exc}) 
\]

The side conditions of the inference rules exclude trivial reasoning without loss of generality.

The system \( \lambda_{\text{exc}} \) without \((\text{exc})\) is denoted by \( \lambda^{-\perp} \), and the system \( \lambda^{-\perp} \) without \((\perp E)\) is denoted by \( \lambda^{-} \).

The classical rule \((\text{exc})\) is a variant of the law of the excluded middle. This rule is introduced independently of \((\perp E)\), which is in contrast to the double-negation elimination rules, such that \((\perp_C)\): infer \( \vdash A \) from \( \neg A \vdash \perp \) and that \( C \): infer \( A \) from \( \neg \neg A \). We computationally call the rule \((\text{exc})\) a rule of local exception-handling. The type \( A \) in \((\text{exc})\) is computationally called a type of exceptional parameter.

In the application of \((\perp I)\), \( y : \neg A \) in \( \Gamma \) is used as a major premise in the usual sense of \((\rightarrow E)\), and only this kind of negative assumption is discharged by \((\text{exc})\). This style of proof is called a regular proof in [Ando95]. In the \( \lambda_{\Delta} \)-calculus [RS94], not only regular but also non-regular proofs are considered. However, from a non-regular proof we can simply construct a regular proof that has the same assumptions and the same conclusion.

The reduction rules \((e2)\), \((e3-1,2)\), and \((e4-1,2)\) below are logically obvious, but they are computationally important. The reduction rule \((e5)\) is logically a kind of permutative reductions in the sense of [Praw65][Praw71][Ando93], which is also called the structural reduction in [Pari92].

Term Reductions:

\((e1)\) \( (\lambda x. M) N \triangleright M[x := N] \)  
\((e2)\) \( (\text{raise} M)N \triangleright (\text{raise} M) \)  
\((e3-1)\) \( y(\text{raise} M) \triangleright M \)  
\((e3-2)\) \( y[y_1 : \neg A] M \triangleright yM[y_1 := y] \)  
\((e4-1)\) \( [y : \neg A] M \triangleright M \) if \( y \notin FV(M) \)  
\((e4-2)\) \( [y : \neg A](\text{raise} yM) \triangleright [y : \neg A] M \)  
\((e5)\) \( [(y : \neg(A \rightarrow B)) M] N \triangleright [y : \neg B]((M[y \leftarrow N]) N) \),

where \( M[y \leftarrow N] \) is defined as follows:

\[ x[y \leftarrow N] = x; \]
\[ (\lambda x. M)[y \leftarrow N] = \lambda x. M[y \leftarrow N]; \]
\[ (y M)[y \leftarrow N] = y(M[y \leftarrow N]); \]
\[ (y' M)[y \leftarrow N] = y'(M[y \leftarrow N]) \text{ if } y' \neq y; \]
\[ (M_1 M_2)[y \leftarrow N] = (M_1[y \leftarrow N])(M_2[y \leftarrow N]); \]
\[ (\text{raise } M)[y \leftarrow N] = \text{raise}(M[y \leftarrow N]); \]
\[ ([y': \neg A'] M)[y \leftarrow N] = [y': \neg A'](M[y \leftarrow N]). \]

We identify \([y : \neg A][y_1 : \neg A] \cdots [y_n : \neg A] M\) with \([y : \neg A] M[y_1 , \cdots , y_n := y]\) for technical simplicity. We sometimes use the term \([y]M\) without type information. The reflexive transitive closure of \( \triangleright \) is denoted by \( \triangleright_{\text{exc}} \) and the binary relation \( =_{\text{exc}} \) is defined as the reflexive, symmetric, and transitive closure of \( \triangleright \). The relations \( \triangleright_{\beta} \), \( \triangleright_{\epsilon x} \) and \( =_{\beta} \) are usually defined.

**Proposition 3** There exists a term \( M \) such that \( \Gamma \vdash_{\lambda_{\text{exc}}} M : A \) iff \( A \) as a formula is classically provable from \( \Gamma \).

**Proposition 4** (Subject Reduction) Let \( \Gamma \vdash_{\lambda_{\text{exc}}} M : A \). If \( M \triangleright_{\text{exc}} N \), then \( \Gamma \vdash_{\lambda_{\text{exc}}} N : A \).
Definition 4 (\(\lambda_{\text{exc}}\)-Proofs with Invariant) We say that \(M\) is a \(\lambda_{\text{exc}}\)-proof with an invariant \(A_i\) if for some \(\Gamma\) and \(A\) there is a deduction of \(\Gamma \vdash A\) and the rule (exc) is used at most once in the deduction where, if used, the type of exceptional parameter is \(A_i\). \(\diamond\)

By Proposition 2, with respect to the implicational fragment we obtain that \(\Gamma \rightarrow B\) in LK iff \(\Gamma, \text{Assume}(A_i, B) \rightarrow A_i\), in LJK for any \(A_i \in CI(B)\) iff \(\Gamma, \text{Assume}(A_i, B), \neg A_i \rightarrow A_i\), in LJ iff \(\Gamma, \text{Assume}(A_i, B), \neg A_i \vdash A_i\) in \(\lambda^{-1}\). By an application of (exc) where the type of exceptional parameter is \(A_i\), the last statement implies \(\Gamma \vdash A\). In this sense the above definition gives a corresponding notion to that of sequent calculus. Moreover, from the above observation there is a strict fragment of \(\lambda_{\text{exc}}\), which is complete with respect to classical provability, such that the restricted term has the following syntax \(M_C\) with a single use of (exc):

\[
M_C := [y]M_f | \lambda x.M_C;
\]

\[
M_f := x | \lambda x.M_f | M_f M_f | yM_f | \text{raise}(M_f).
\]

For instance, the term \(P \equiv \lambda x_1.[y]x_1(\lambda x_2.\text{raise}(yx_2))\) of the form \(M_C\) is a proof of Peirce’s law.

Let \(C[\ ]\) be a context with a single hole \([\ ]\) such that \(C[\ ] := [\ ] | C[\ ]M\). We denote \(C[M]\) by the term obtained by replacing \([\ ]\) in \(C[\ ]\) with the term \(M\). Then we have \(C[\text{raise}M] \vdash_{\text{exc}} \text{raise} M\). If \(k \notin C[M]\), then we have that \(P\lambda k.C[kM] \vdash_{\text{exc}} M\), which can be applied for implementing a simple exit mechanism. Here, the context \(C[\ ]\) is abandoned, and the term \(M\) to be passed on has the same type as that of exceptional parameter of \(P\). This is the reason why the type \(A\) in the definition of (exc) is called a type of exceptional parameter. In terms of ML 

\(\text{ML} [\text{MTH90}]\), informally \([y:A]M\) may be read as let exception \(y\) of \(A\) in \(M\) handle \((yx) \Rightarrow x\) end, based on the correspondence of \(\bot\) with \(\text{exn}\) (type of exceptions in \(\text{ML}\)).

As a counterpart of Theorem 1, the following proposition shows that the restricted terms \(M_C\), which would represent some standard form of classical proofs are complete with respect to classical provability, and that the existence of invariants allows an effective way to determine which type has to be assumed in writing programs as classical proofs. Moreover, any invariant in \(CI(\cdot)\) can be computationally characterized as the type of exceptional parameter.

Proposition 5 Let \(A\) as a formula be classically provable. Then for any \(A_i \in CI(A)\), there exists a \(\lambda_{\text{exc}}\)-proof \(M_C\) of the type \(A\) with the invariant \(A_i\).

In the next section, we give a concrete translation to the LJK proofs in \(\lambda_{\text{exc}}\). From the definability in classical logic, the following examples are demonstrated in this strict fragment.

Example 1 (Definition of \(\times\)) \(A \times B = \neg(A \rightarrow \neg B)\):

\[
\langle M, N \rangle = \lambda x.\text{MN}; \quad \text{fst} = \lambda x.[y]\text{raise}(x\lambda x_1x_2.yx_1); \quad \text{snd} = \lambda x.[y]\text{raise}(x\lambda x_1x_2.yx_2).
\]

Then it is obtained that \(\text{fst}\langle N_1, N_2 \rangle \vdash_{\text{exc}} N_1\) and \(\text{snd}\langle N_1, N_2 \rangle \vdash_{\text{exc}} N_2\).

\(\text{ML}\) can be used to write the \(\text{ML}\) program fun \(\text{Peirce}(w) = \text{let exception } y\text{ of }'1\alpha\text{ in }w(fn z \Rightarrow \text{raise}(yz))\text{ handle } (yx) \Rightarrow x\text{ end}\) as the proof \(P\), whose type can be inferred as ((\(1\alpha \rightarrow 1\beta\)) \rightarrow (\text{\'1\alpha\text{ \text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\beta}}}}}}}}}}}}) \rightarrow \text{\'1\alpha}\text{ by the }\text{ML}\text{ system, the correspondence is informal in the sense that }\text{ML}\text{ is a call-by-value language and the occurrence of }y\text{ in exception } y\text{ is treated as a name of an exception rather than a variable, like in }[y]M.\text{ See also section 8.} \)
Example 2 (Definition of +) \( A + B = \neg A \rightarrow \neg B \):

\[
\text{inl}(M) = \lambda x v. x M; \quad \text{inr}(M) = \lambda x v. M;
\]
when \((M, [x_1]N_1, [x_2]N_2) = \text{[y]} \text{raise}(M(\lambda x_1 y N_1)(\lambda x_2 y N_2))\)
when \((\text{inl}(M), [x_1]N_1, [x_2]N_2) \triangleright^{*}_{\text{exc}} N_1[x_1 := M] \);
when \((\text{inr}(M), [x_1]N_1, [x_2]N_2) \triangleright^{*}_{\text{exc}} N_2[x_2 := M] \).

**Proposition 6 (Church-Rosser Theorem)** If \( M \triangleright^{*}_{\text{exc}} N_1 \) and \( M \triangleright^{*}_{\text{exc}} N_2 \), then \( N_1 \triangleright^{*}_{\text{exc}} M' \) and \( N_2 \triangleright^{*}_{\text{exc}} M' \) for some \( M' \).

**Proof.** Similarly to the proof in section 6.

### 4.2 Translation to LJK Proofs

According to Theorem 1, we can always obtain LJK proofs with some invariants for any classical theorem. This suggests a translation from arbitrary classical proofs to LJK proofs. We give the translation in terms of \( \lambda_{\text{exc}} \); however, it is also possible in other classical systems, e.g., in the \( \lambda_{\text{P}} \)-calculus. This analysis gives a new reduction relation to \( \lambda_{\text{exc}} \), which shifts the invariant into the inside. To establish this translation, we use an auxiliary type system \( \lambda^{-\perp} \) consisting of simply typed \( \lambda \)-calculus with the intuitionistic absurdity rule. The translation is obtained in the following way:

1. Given a proof \( M \) of type \( A \) in \( \lambda_{\text{exc}} \). Compute an embedding \( G(M) \) into \( \lambda^{-\perp} \);
2. A proof of \([y: \neg A] \text{raise}(G(M)\lambda z.y z)\) is a \( \lambda_{\text{exc}} \)-proof of \( A \) with an invariant \( A \);
3. To get a \( \lambda_{\text{exc}} \)-proof of \( A \) with an invariant \( A_i \), apply the shift reduction (to be defined later) \( i \)-times to \([y: \neg A] \text{raise}(G(M)\lambda z.y z)\), where \( CI(A) \) is \([A_0, \cdots, A_{n}] \) and \( 0 \leq i \leq n \).

**Definition 5** The embedding of \( G \) from the proof terms of \( \lambda_{\text{exc}} \) to \( \lambda^{-\perp} \) is defined.

\[
G(x) = \lambda k.xk;
G(\lambda x.M) = \lambda k.k(\lambda x.\text{raise}(G(M)\lambda m.k(\lambda v.m))));
G(yM) = \lambda k.k(G(M)\lambda m.ym);
G(MN) = \lambda k.G(M)(\lambda m.G(N)\lambda n.k(mn));
G(\text{raise}(M)) = \lambda k.G(M)\lambda x.x;
G([y: \neg A]M) = \lambda y.G(M)(\lambda m.ym). \quad \diamond
\]

**Proposition 7** If we have \( \Gamma \vdash M : A \) in \( \lambda_{\text{exc}} \), then \( \Gamma \vdash G(M) : \neg \neg A \) in \( \lambda^{-\perp} \).

**Proof.** By induction on the derivation. \( \square \)

We define an invariant shift reduction relation \( \triangleright_{s} \) for \( \lambda_{\text{exc}} \)-proofs with some invariant, which changes an outer invariant to an inner invariant:

\( [y: \neg (A \rightarrow B)]M \triangleright_{s} \lambda x.[y: \neg B](Mx[y := \lambda k.y(kx)]) \).

The \( i \) applications of \( \triangleright_{s} \) are denoted by \( \triangleright_{s}^{i} \) for \( i = 0, 1, 2, \cdots \).

Let \([A_0, A_1, \cdots, A_{n}] \) be \( CI(A) \). Then we assume on the ordering that \( A_0 \equiv A \) is the outermost invariant and \( A_{n} \) is the innermost invariant, and that \( A_{i} \equiv A_{i}' \rightarrow A_{i+1} \) for some \( A_{i}' \) where \( 0 \leq i \leq n - 1 \).

**Lemma 4** Let \([A_0, A_1, \cdots, A_{n}] \) be \( CI(A) \). If we have \( \Gamma \vdash M : A \) in \( \lambda_{\text{exc}} \), then for any \( i \) in \( 0 \leq i \leq n \), \( M' \) such that \([y: \neg A] \text{raise}(G(M)\lambda k.y k) \triangleright_{s}^{i} M' \) is a \( \lambda_{\text{exc}} \)-proof of \( A \) with an invariant \( A_{i} \).

**Proof.** By case analysis on the number of \( i \).
Case of $i = 0$:
If $\Gamma \vdash M : A$ in $\lambda_{exc}$, then $\Gamma \vdash G(M) : \neg \neg A$ in $\lambda^{-}$.
Hence, $[y : \neg A]raise(G(M)\lambda k.yk)$ is a $\lambda_{exc}$ proof of $A$ with an invariant $A \equiv A_0$.

Case of $i = k + 1$ where $0 \leq k \leq n - 1$:
Assume that $\lambda x_1 \cdots x_k[y : \neg A_k]N$ is a $\lambda_{exc}$-proof of $A$ with an invariant $A_k$ where $A_k \equiv A_k \rightarrow A_{k+1}$. Then $\lambda x_1 \cdots x_k[y : \neg A_k]N \triangleright_{\beta} M'$ gives a $\lambda_{exc}$-proof of $A$ with the invariant $A_{k+1}$ by the following replacement of each $yO$ with $(\lambda k.y(kx))O$:

$$
\frac{[y : \neg A_k]1}{\lambda x_1 \cdots x_k[y : \neg A_k]N : A} \triangleright_{\beta} \lambda x_1 \cdots x_k[y : \neg A_k]N : A
$$

$$
\frac{[y : \neg A_k]1}{\lambda k.y(kx) : \neg A_k} \quad \frac{\lambda k.y(kx)}{\perp} \quad \frac{A}{O : A_k}
$$

$$
\frac{\lambda k.y(kx) : \neg A_k}{\perp} \quad \frac{A}{O : A_k}
$$

The formula (invariant) to which only the right contraction rules are applied in terms of sequent calculus is changed to the inside by the reduction rules $\triangleright_{\beta}$. On the other hand, the shift of the invariant is characterized in terms of Theorem 1 on page 39 of [Praw65], that is, the application of $(\perp_C)$ can be restricted to atomic formulae where $\perp$ is defined in terms of the other connectives. Moreover, with respect to $\lambda_{exc}$-proofs with the innermost invariant, the application of $(exc)$ is to be a strictly positive and atomic subformula of the conclusion in the implication fragment (possibly with $\land$). In the more general case of adding a primitive $\lor$, it would not be possible to postulate $(exc)$ only for an atomic formula.

It is stated that $\triangleright_{\beta}$ and $(e5)$ have a strong connection, such that $([y := \lambda z.y(zx)[x]]N \triangleright_{\beta} \lambda x_1 \cdots x_k[y := \neg A_k]N : A)$, which leads to the same result as the one by $(e5)$, since we have that $M[y := \lambda z.y(zN)] \triangleright_{\beta} M[y \triangleright_{\beta} N]$.

### 4.3 Comparison with Related Work

In the following subsection, we briefly compare $\lambda_{exc}$ with some of the existing ones (not a call-by-value style); $\lambda\mu$-calculus[Parigot92], $\lambda_\Delta$[RS94], and a variant of $\lambda_c$[FFKD86]. As regards the relation between $\lambda\mu$ and $\lambda_{exc}$, we can obtain an isomorphism between them, and the Strong Normalization of $\lambda_{exc}$. Our observation on the relation between $\lambda_\Delta$ and $\lambda_{exc}$ suggests a generalization of some reduction rule of $\lambda_\Delta$, which can lead to an isomorphism between them. In relation to $\lambda_c$, we discuss that adding what kind of reduction rule to $\lambda_c$ makes them isomorphic.

#### 4.3.1 Relation to Parigot’s $\lambda\mu$-Calculus

To study computational interpretations of classical proofs, Parigot [Parigot92] introduced the $\lambda\mu$-calculus of 2nd order classical natural deduction with multiple conclusions. The $\lambda\mu$-calculus has elegant properties; from a proof theoretical point, in contrast to the well-known $NK$, $\lambda\mu$ has no operational rules like double-negation elimination or the absurdity rule but has multiple conclusions and structural rules. The positive fragment of $\lambda\mu$ is complete with respect to positive fragment of classical logic, namely, to prove, for example,
Peirce’s law, we do not have to use $\bot$ that is not a subformula of the theorem. On the other hand, in $NK$, $\lambda$S[RS94], $\lambda_{exc}$, and a variant of $\lambda\mu$ à la Ong [Ong96], we have to use $\bot$ in the proof, which is not contained in the conclusion. Moreover, since in $\lambda\mu$ the name $[\alpha]$ always appears as the form $[\alpha]M$ for some term $M$, the notion of regularity in [Ando95] is involved in the system.

From a computational side, in $\lambda\mu$ [Pari92][Pari93-1][Pari93-2] some proof terms of theorems may contain free name $\delta$ of $\bot$, e.g., the term $\lambda x_1.\mu\alpha.[\delta](x_1(\lambda x_2.\mu\delta.[\alpha]x_2))$ of type $\neg\neg A \rightarrow A$ has a free name $\delta$. To keep our usual intention of closed terms, we adopt a variant of $\lambda\mu$-calculus à la Ong [Ong96] and study the relation between $\lambda\mu$ à la Ong and $\lambda_{exc}$. At first appearance the $\lambda\mu$-calculus has a single conclusion, however the remaining conclusions are placed on the left side after the semicolon.

The system of $\lambda\mu$ is defined in the following. The types are usually defined from atomic types including $\bot$ using $\rightarrow$. The context $\Gamma$ and terms are defined as usual. The set of types with names is denoted by $\Delta$.

$\lambda\mu$:

\[
\Gamma := () | x : A, \Gamma;
\Delta := () | A^\alpha, \Delta;
M := x | MM | \lambda x.M | [\alpha]M | \mu\alpha.M;
\]

\[
\Gamma; \Delta \vdash x : \Gamma(x)
\]

\[
\Gamma; \Delta \vdash M : B \quad \Gamma; \Delta \vdash \lambda x.M : A \rightarrow B
\]

\[
\Gamma; \Delta \vdash M : A \quad \Gamma; \Delta \vdash A^\alpha \vdash [\alpha]M : \bot
\]

\[
\Gamma; \Delta; A^\alpha \vdash M : \bot \quad \Gamma; \Delta; \mu\alpha.M : A \text{ if } A \neq \bot
\]

The reduction relation $\triangleright_{\mu}$ of $\beta$-reductions, structural reductions, $(S1)$, and $(S2)$ in [Pari93-1] is considered, namely,

\[
(\lambda x.M)N \triangleright_{\mu} M[x := N];
\]

\[
(\mu \alpha.M)N \triangleright_{\mu} \mu \alpha.M[\alpha \leftarrow N];
\]

\[
(S1): \ [\alpha]M \beta.M \triangleright_{\mu} M[\beta := \alpha];
\]

\[
(S2): \mu \alpha.M \triangleright_{\mu} M \text{ if } \alpha \not\in \text{FreeName}(M).
\]

The binary relations $\triangleright_{\mu}^*$ and $\equiv_{\mu}$ are usually defined. As by-products, we obtain the Strong Normalization property of $\lambda_{exc}$ and an isomorphism between $\lambda_{exc}$ and $\lambda\mu$ with respect to conversions.

**Definition 6 (Translation from $\lambda_{exc}$ to $\lambda\mu$)**

\[
\begin{align*}
x &= x; & \lambda x.M &= \lambda x.M; \\
yM &= [y]M; & MN &= M \cdot N; \\
raise(M) &= \mu \alpha.M \text{ where } \alpha \text{ is a fresh name}; & [y]M &= \mu y.[y]M.
\end{align*}
\]

For this translation, we separate a context in $\lambda_{exc}$ into two parts as follows:

\[
\Gamma ::= \Gamma_1 | \Gamma_2;
\]

\[
\Gamma_1 ::= () | x : A, \Gamma_1; \quad \Gamma_2 ::= () | y : \neg A, \Gamma_2.
\]

\[
y : \neg A, \Gamma_2 = A^\alpha, \Gamma_2.
\]

**Proposition 8** If we have $\Gamma_1, \Gamma_2 \vdash_{\lambda_{exc}} M : A$, then $\Gamma_1; \Gamma_2 \vdash_{\lambda\mu} M : A$.

**Lemma 5** For any $\lambda_{exc}$-term $M$, $M[x := N] = M[x := N]$.
Lemma 6  $M[y \leftarrow N] = M[y \leftarrow N]$

The above proposition and lemmata can be proved by straightforward induction.

Lemma 7  If $M \triangleright_{\text{exec}} N$, then $M \triangleright_{\mu} N$.

Proof. By induction on the derivation $M \triangleright_{\text{exec}} N$. □

From Lemma 10, Proposition 8, and the Strong Normalization of $\lambda\mu$ [Pari93-1][Pari93-2], we obtain that well-typed $\lambda_{\text{exec}}$-terms are strongly normalizable\(^5\).

Corollary 3  Well-typed $\lambda_{\text{exec}}$-terms are strongly normalizable.

Definition 7  (Translation from $\lambda\mu$ to $\lambda_{\text{exec}}$)

$<x> = x$; $<\lambda x.M> = \lambda x.<M>$; $<MN> = <M><N>$;
$<\alpha>M> = \alpha<M>$; $<\mu\alpha.M> = \alpha\text{raise}(M)$.
$<A\alpha,\Delta> = \alpha\neg A,<\Delta>$.  ◊

Proposition 9  If $\Gamma;\Delta \vdash_{\lambda\mu} M : A$, then $\Gamma, <\Delta> \vdash_{\text{exec}} <M> : A$.

Lemma 8  $<M>[x := <N>] = <M[x := N]>$

Lemma 9  $<M>[\alpha \leftarrow <N>] = <M[\alpha \leftarrow N]>$

The above proposition and lemmata can be proved by straightforward induction.

Lemma 10  If $M \triangleright_{\mu} N$, then $<M> \triangleright_{\text{exec}} <N>$.

Proof. By induction on the derivation $M \triangleright_{\mu} N$. □

Proposition 10  For any $\lambda_{\text{exec}}$-term $M$, $<M> \triangleright_{\text{exec}} M$.
For any $\lambda\mu$-term $N$, $<N> \triangleright_{\mu} N$.

Proof. By the definitions of the translations. □

From Lemmata 10 and 13 and Proposition 10, with respect to conversions there is an isomorphism between $\lambda_{\text{exec}}$ and $\lambda\mu$.

Corollary 4  $(\lambda_{\text{exec}} \simeq \lambda\mu)$ $\lambda_{\text{exec}}$ and $\lambda\mu$ are isomorphic in the sense that $M =_{\mu} N$ iff $<M> =_{\text{exec}} <N>$ and that $M =_{\text{exec}} N$ iff $M =_{\mu} N$.

In terms of the right structural rules of sequent calculus, the operator $\mu$ in $\lambda\mu$ works both for the right contraction and the right weakening. In $\lambda_{\text{exec}}$, the right contraction can be simulated by ($\text{exec})$, and the right weakening by ($\perp I$) and ($\text{raise}$). The logical aspect of the operator $\mu$ can be split into two primitive ones of $\lambda_{\text{exec}}$, which is also computationally justified under the isomorphism, and applied to define proof terms of classical substructural logics in [Fuji95].

4.3.2 Relation to $\lambda_{\Delta}$-Calculus of Rehof and Sørensen

\(^5\)Of course, we can establish the strong normalization property of $\lambda_{\text{exec}}$ directly.
For the purpose of establishing the Curry-Howard isomorphism in classical logic, Rehof and Sørensen [RS94] introduced the $\lambda_\Delta$-calculus by restriction of Felleisen’s control operator $C$ to avoid a breakdown of neat properties like the Church-Rosser property. The $\lambda_\Delta$-calculus is natural and has good properties not only of proof theory but also of typed calculus. In relation to $\lambda_{exc}$, the $\lambda_\Delta$-calculus treats both regular proofs and non-regular proofs, in other words, there is no distinction of variables that are bound by $\lambda$-abstraction or $\Delta$-abstraction. Of course any non-regular proof can be translated into a regular proof without changing assumptions and the conclusion, such that each variable $y$ that is abstracted by $\Delta$ is replaced with $\lambda x.yx$. To study the relation between $\lambda_\Delta$ and $\lambda_{exc}$, we consider the $\lambda_\Delta$-proofs under this modification.

The definition of $\lambda_\Delta$[RS94] is briefly given below. The syntax of $\lambda_\Delta$-terms is defined as follows:

$$M ::= x \mid \lambda x.M \mid MM \mid \Delta x.M$$

The reduction rules are defined as (d1), (d2), and (d3) together with $\beta$-reductions.

(d1): $(\Delta x.M).N \triangleright \Delta x.M[x := \lambda z.x(zN)]$;

(d2): $\Delta x.x.M \triangleright M$ if $x \notin FV(M)$;

(d3): $\Delta x.(\Delta d.x.M) \triangleright M$ if $x, d \notin FV(M)$.

The type inference rules are ($\rightarrow I$), ($\rightarrow E$), and the following ($\perp_e$).

$$\frac{\Gamma, x : A \rightarrow \llcorner \vdash M : \perp_\perp}{\Gamma \vdash \Delta x.M : A} (\perp_e)$$

**Definition 8** (Translation from $\lambda_\Delta$ to $\lambda_{exc}$)

$x^\circ = x$;  $(\lambda x.M)^\circ = \lambda x.M^\circ$;

$(MN)^\circ = M^\circ N^\circ$;  $(\Delta x.M)^\circ = [x]\text{raise}M^\circ$.  \(\Diamond\)

**Proposition 11** (1) If we have $\Gamma \vdash_{\lambda_\Delta} M : A$, then $\Gamma \vdash_{\lambda_{exc}} M^\circ : A$.

(2) If we have $M \triangleright N$ in $\lambda_\Delta$, then $M^\circ =_{exc} N^\circ$ in $\lambda_{exc}$.

The above proposition can be verified by induction. Especially (2) is confirmed using that $(M[y := \lambda z.y(zN)])^\circ \triangleright_{\perp_{\perp}} M^\circ[y \leftarrow N^\circ]$, where to prove (2), in contrast to Lemma 10, the case of (d1) introduces conversions instead of reductions. From (2), equivalent $\lambda_\Delta$-terms are translated into equivalent $\lambda_{exc}$-terms with respect to conversions (correctness of the translation).

**Definition 9** (Translation from $\lambda_{exc}$ to $\lambda_\Delta$)

$x^+ = x$;  $(\lambda x.M)^+ = \lambda x.M^+$;

$(yM)^+ = yM^+$;  $(MN)^+ = M^+ N^+$;

$(\text{raise}M)^+ = \Delta d.M^+$ provided $d \notin FV(M)$;  $([y]M)^+ = \Delta y.yM^+$.  \(\Diamond\)

**Proposition 12** If $\Gamma \vdash_{\lambda_\Delta} M : A$, then $\Gamma \vdash_{\lambda_{exc}} M^+ : A$.

As regards the statement that if we have $M \triangleright_{exc} N$, then $M^+ =_{\Delta} N^+$ in $\lambda_\Delta$, where $=_{\Delta}$ is the reflexive, symmetric, and transitive closure of $\triangleright$ in $\lambda_\Delta$, our reduction rule of (e3-2) fails even if we drop (c3-1) and (c3-2). Our observation suggests adding a new reduction to $\lambda_\Delta$, instead of (d3), such that $\Delta x.x\Delta d.M \triangleright \Delta x.M$ where $d \notin FV(M)$: (d4). Here, the new rule (d4) is a general form of (d3). The dropped (d3) rule can be recovered by (d2) and (d4), and moreover the simulation of Felleisen’s $\lambda_c$ [FFKD86] by $\lambda_\Delta$ (call-by-value variant), which is observed in [RS94] is not lost. Then we can obtain that $M^+ \triangleright_{exc} N^+$ in $\lambda_\Delta$ if $M \triangleright_{exc} N$ without (c3-1) and (c3-2). Moreover, we have that $(M^+)^\circ \triangleright_{\perp_{\perp}} M$ and
that \((M^\circ)^+ \triangleright^* M\) in \(\lambda_\Delta\) with (d4) instead of (d3). Hence, as in Corollary 4, there is an isomorphism between \(\lambda_{exe}\) with (e3-1), (e3-2) and \(\lambda_\Delta\) with (d4) instead of (d3).

With respect to the remaining rules (e3-1) and (e3-2), they can be simulated in \(\lambda_\Delta\) by using the following rule:

\[ y\Delta x.M \triangleright M[x := y], \]

where the type of the variable \(y\) is of the form \(A \rightarrow \bot\). All the above modification of \(\lambda_\Delta\) can lead to an isomorphism between them \((\lambda_\Delta \simeq \lambda_{exe})\).

### 4.3.3 Relation to a variant of \(\lambda_c\)-Calculus of Felleisen\(^6\)

For reasoning about a call-by-value language, Felleisen, et al. \cite{FFKD86, FFH92} introduced the \(\lambda_c\)-calculus extending the type-free \(\lambda_c\)-calculus of Plotkin \cite{Plot75} with the control operator \(C\) and the abort operator \(A\). By Griffin \cite{Grif90}, the \(\lambda_c\)-calculus has been applied to extend the Curry-Howard isomorphism to classical logic from a computational interest. It is a distinct point that \(\lambda_c\) has the usual reduction rules and the computation rules used only at the top-level, which bring the computation of the top-level continuation to a stop. Since P.de Groote \cite{Groo94} proved that there is an isomorphism between \(\lambda\mu\) and a call-by-name variant of \(\lambda_c\), the relation may be obvious. However, we observe that the computation rules in \(\lambda_c\) are necessary to simulate some of the compatible rules in \(\lambda_{exe}\) and that \(\lambda_{exe}\) would be simulated in \(\lambda_c\) with some reduction rule. According to the observations in \cite{Groo94, RS94}, we consider a call-by-name variant of \(\lambda_c\) as follows: The terms are defined as usual.

\[ M ::= x | \lambda x.M | MM | \mathcal{F}M \]

The reduction rules are the \(\beta\)-reduction, \((F_L)\), and \((F_{top})\) as follows:

\[ (F_L): (\mathcal{F}M)N \triangleright \mathcal{F}(\lambda k.M(\lambda f. kfN)); \quad (F_{top}): \mathcal{F}M \triangleright \mathcal{F}(\lambda k.M(\lambda f. k f)). \]

The operator \(\mathcal{F}\) has the type \(\Rightarrow \mathcal{C} \rightarrow \mathcal{A}\), which is a variant of and can be defined by Felleisen's \(C\), see \cite{RS94}. In addition, the computation rule is \((F_T): \mathcal{F}M \triangleright T M \lambda x.x\) that is applied only at the top-level.

**Definition 10 (Translation from \(\lambda_c\) to \(\lambda_{exe}\))**

\[ \langle x \rangle = x; \quad \langle \lambda x.M \rangle = \lambda x.\langle M \rangle; \]
\[ \langle MN \rangle = \langle M \rangle \langle N \rangle; \quad \langle \mathcal{F}M \rangle = [y]\text{raise}(\langle M \rangle \lambda x.yx). \]

**Proposition 13** (1) If we have \(\Gamma \vdash_{\lambda_c} M : A\), then \(\Gamma \vdash_{\lambda_{exe}} \langle M \rangle : A\).

(2) If we have \(M \triangleright N\) in \(\lambda_c\), then \(\langle M \rangle =_{exe} \langle N \rangle\).

The above proposition can be proved by a straightforward induction.

**Definition 11 (Translation from \(\lambda_{exe}\) to \(\lambda_c\))**

\[ \overline{x} = x; \quad \overline{\lambda x.M} = \lambda x.\overline{M}; \]
\[ \overline{yM} = y\overline{M}; \quad \overline{MN} = \overline{M}\overline{N}; \]
\[ \overline{\text{raise}M} = \mathcal{F}(\lambda v.\overline{M}) \] where \(v\) is a fresh variable: \[ [y]\overline{M} = \mathcal{F}(\lambda y.y\overline{M}). \]

**Proposition 14** If we have \(\Gamma \vdash_{\lambda_{exe}} M : A\), then \(\Gamma \vdash_{\lambda_c} \overline{M}\).

With respect to the correctness of the translation, the reduction rules (c2) and (e5) can be simulated by \((F_L)\). We also have that \(\langle M\rangle \triangleright^*_{exe} M\). In contrast, the compatible rules (4-1) and (4-2) can be simulated by the use of the non-compatible \((F_T)\). Moreover, for (e3-1) and (e3-2), they can be simulated in \(\lambda_c\) by using the following reduction rule \(F''_R\):

\(^6\)See also 9.2 Relation to Felleisen's \(\lambda_c\).
We provide a simple natural deduction system $\lambda^v_{exc}$ of classical propositional logic, in which the reduction rules are based on a call-by-value strategy. Since there is an isomorphism with respect to conversions between $\lambda_{exc}$ and Parigot's $\lambda\mu$-calculus [Pari92], $\lambda^v_{exc}$ can also be regarded, in some sense, as a call-by-value variant of $\lambda\mu$-calculus.

The notion of values is defined as variables, $\lambda$-abstractions, and terms of the form $yV$ for a value $V$ as in [Groo95], where the variable $y$ works as a value-constructor for any value $V$. On the other hand, since a term of the form $[y]M$, like a packet opened by (ev4-1), is not regarded as a value, $(\lambda x.M_1)[y]M_2$ does not become a $\beta$-redex, but another redex that is dual to the structural reduction in [Pari92], which is logically a kind of permutative reduction in the sense of [Praw65][Praw71][Ando95].

Values

$V ::= x \mid \lambda x.M \mid yV$

Term reductions

$(\text{ev1}) \ (\lambda x.M)V \triangleright^v_{exc} M[x := V];$
$(\text{ev2-1}) \ (\text{raise } M)N \triangleright^v_{exc} \ (\text{raise } M);$ \hspace{1cm} $(\text{ev2-2}) \ V(\text{raise } M) \triangleright^v_{exc} (\text{raise } M);$ \hspace{1cm} $(\text{ev3-1}) \ y(\text{raise } M) \triangleright^v_{exc} M;$ \hspace{1cm} $(\text{ev3-2}) \ y[y_1]M \triangleright^v_{exc} yM[y_1 := y];$
$(\text{ev4-1}) \ [y]M \triangleright^v_{exc} M$ if $y \notin \text{FV}(M);$ \hspace{1cm} $(\text{ev4-2}) \ [y](\text{raise } yM) \triangleright^v_{exc} [y]M;$
$(\text{ev5-1}) \ ([y]M)N \triangleright^v_{exc} [y](M[y \leftarrow N]);$ \hspace{1cm} $(\text{ev5-2}) \ V([y]M) \triangleright^v_{exc} [y](V(MV \Rightarrow y)),$
where $M[y \leftarrow N]$ and $M[N \Rightarrow y]$ are defined respectively as follows:

$x[y \leftarrow N] = x;$
$(\lambda x.M)[y \leftarrow N] = \lambda x.M[y \leftarrow N];$
$(yM)[y \leftarrow N] = y(M[y \leftarrow N]);$
$(y'M)[y \leftarrow N] = y'(M[y \leftarrow N]) \text{ if } y' \neq y;$
$(M_1M_2)[y \leftarrow N] = (M_1[y \leftarrow N])(M_2[y \leftarrow N]);$
$(\text{raise } M)[y \leftarrow N] = \text{raise}(M[y \leftarrow N]);$
$([y' : \neg A']M)[y \leftarrow N] = [y' : \neg A'](M[y \leftarrow N]).$

The binary relation $\triangleright^v_{exc}$ is defined by the reflexive transitive closure of $\triangleright^v_{exc}$, and the congruence relation is denoted by $=^v_{exc}$. The relation $\triangleright_{\beta v}$ is defined as usual. We sometimes
use the term \([y : \neg A]M\) instead of \([y]M\).

**Proposition 15** There exists a term \(M\) such that \(\Gamma \vdash_{\lambda_{exe}} M : A\) iff \(A\) as a formula is classically provable from \(\Gamma\).

**Proposition 16 (Subject Reduction)** Let \(\Gamma \vdash_{\lambda_{exe}} M : A\). If \(M \triangleright_{exe} N\), then \(\Gamma \vdash_{\lambda_{exe}} N : A\).

Although \(\lambda_{exe}\) is simple, the data types of pair and case-analysis given below are naturally implemented by the definability in classical logic.

**Example 3 (Definition of +)** \(A + B = \neg A \to B:\)
\[
\text{\textit{inl}(M) = } \lambda x.\text{\textit{raise}}(xM); \quad \text{\textit{inr}(M) = } \lambda v.M,
\]
when \((M, [x_1]N_1, [x_2]N_2) = [y](\lambda x_2.N_2)(M\lambda x_1.yN_1)\). Then we can obtain the following computation:
\[
\text{\textit{when}(\text{\textit{inl}}(V), [x_1]N_1, [x_2]N_2)\triangleright^{\ast}_{exe}N_1[x_1 := V]; \quad \text{\textit{when}(\text{\textit{inr}}(V), [x_1]N_1, [x_2]N_2)\triangleright^{\ast}_{exe}N_2[x_2 := V].}
\]

Let a context \(E[]\) with a hole \([\underline{\phantom{\mathcal{E}[\mathcal{E}[]]}]}\) be as follows:
\[
E[] := [\underline{\phantom{\mathcal{E}[\mathcal{E}[]]}]} | V(E[]) | (E[])M.
\]
We denote \(E[M]\) by the term obtained by replacing \([\underline{\phantom{\mathcal{E}[\mathcal{E}[]]}]}\) in \(E[]\) with the term \(M\). Then we have \(E[\text{\textit{raise}}M]\triangleright^{\ast}_{exe}\text{\textit{raise}}M\) and \(E[y\text{\textit{raise}}yM]\triangleright^{\ast}_{exe}y\text{\textit{raise}}(yE[M])\) where \(y \notin FV(M)\). Here, the continuation \(E\) with respect to \(y\text{\textit{raise}}yM\) is accumulated as an argument of \(y\).

**Example 4 (Exit Mechanism by a Proof of Peirce’s Law)**
Let \(P_1\) be \(\lambda x_1.\text{\textit{y}}x_1(\lambda x_2.\text{\textit{raise}}(y x_2))\) of the type \(\text{(}(A \to B) \to A)\text{ to } A\). We consider the following two cases. The first case is called a normal case, and the second is an exceptional case.

1. Case of \(k \notin FV(M)\):
\[
P_1\lambda k.M = (\lambda x_1.\text{\textit{y}}x_1(\lambda x_2.\text{\textit{raise}}(y x_2)))\lambda k.M \triangleright^{\ast}_{exe} [y]M \triangleright^{\ast}_{exe} M.
\]
2. Case of \(k \notin FV(E[V])\):
\[
P_1\lambda k.E[kV] \triangleright^{\ast}_{exe} [y]E[\text{\textit{raise}}(yV)] \triangleright^{\ast}_{exe} [y]\text{\textit{raise}}(yV) \triangleright^{\ast}_{exe} [y]V \triangleright^{\ast}_{exe} V.
\]

In the second case, the context \(E[]\) is abandoned, and the value \(V\) to be passed on has the same type as that of the exceptional parameter of \(P_1\), which can be applied to implement a simple exit mechanism. This is the reason why type \(A\) in the definition of \((\text{exc})\) is a type of exceptional parameter. In terms of \(ML\) [MTH90], informally \([y : \neg A]M\) may be read as \textit{let} exception \(y\) of \(A\) in \(M\) handle \((y x) \Rightarrow x\) \textit{end}, based on the correspondence of \(\bot\) with \(\text{\textit{en}}\) (type of exceptions in \(ML\))^7.

When an exception arises, we often use an exception handler to continue the computations. From a programming viewpoint, we show three general programs, including programs for normal and exceptional cases. These general programs can be written in the restricted syntax.

1. \(L \equiv \lambda x q.\text{\textit{g}}(x(\lambda k.\text{\textit{raise}}(yk))) : ((A \to B) \to C) \to (C \to A) \to A\)
\[
\text{\textit{LV}_1V_2}\text{ provides the following computation: If } V_1 \text{ returns a normal value } V, \text{ then the result of } \text{\textit{LV}_1V_2 \text{ is } V_2V. If } V_1 \text{ raises an exception with a value } V', \text{ then the entire result becomes } V'. That is, } L \text{ computes a composition of } V_2 \text{ and } V_1 \text{ of a normal case. This type is a substitution instance of Łukasiewicz’s formula.}
\]

2. \(H \equiv \lambda x f.\text{\textit{g}}(x(\lambda k.\text{\textit{raise}}(yk))) : ((A \to B) \to C) \to (A \to C) \to C\)

^7See also section 8.
\( \mathcal{H} \mathcal{V}_1 \mathcal{V}_2 \) gives the following computation: If \( \mathcal{V}_1 \) returns a normal value \( \mathcal{V} \), then the whole result is \( \mathcal{V} \). If \( \mathcal{V}_1 \) raises an exception with a value \( \mathcal{V}' \), then the result of \( \mathcal{H} \mathcal{V}_1 \mathcal{V}_2 \) becomes \( \mathcal{V}_2 \mathcal{V}' \). Namely, \( \mathcal{H} \) can be regarded as a handler of an exceptional case.

(3) \( \mathcal{G} \equiv \lambda x. g. [y] (x (\lambda k. \text{raise}(y(fk)))) : ((A \rightarrow B) \rightarrow C) \rightarrow (C \rightarrow D) \rightarrow (A \rightarrow D) \rightarrow D \)

\( \mathcal{G} \) is obtained to combine the roles of \( \mathcal{L} \) and \( \mathcal{H} \) into one program.

In all of the above, the type of exceptional return, if it happens, is the same as the type of exceptional parameter.

To demonstrate simple examples we assume the constants and the constant functions used below, and the reduction rules and the inference rules are also assumed:

\[
\begin{align*}
\text{if true then } M & \text{ else } N \triangleright M, \quad \text{if false then } M & \text{ else } N ; \\
\text{fix } f.M & \triangleright M[f := \text{fix}.M];
\end{align*}
\]

(i) Let \( \text{prod} \) be

\[
\lambda l. \text{exit}. (\text{fix}. \lambda l. \text{if } l = \text{nil} \text{ then } 1 \text{ else } \text{if } \text{car}(l) = 0 \text{ then } 0 \text{ else } * (\text{car}(l)) \text{ (f(cdr(l)))) } l' \text{ with the type } \text{int list} \rightarrow (\text{int} \rightarrow \text{int}) \rightarrow \text{int}.
\]

To compute the product of all integers in the integer list \( l \), using Example 4 we define \( \text{prod} \) as \( \lambda l. \mathcal{P}_1 (\text{prod } l) \) with the type \( \text{int list} \rightarrow \text{int} \). \( \text{Prod}(l) \) makes it possible to return 0 immediately as an exception if \( l \) contains 0. For instance, we compute neither \( * 1 2 \text{ nor } * 0 3 \) in the following:

\[
\begin{align*}
\text{Prod} & \triangleright [1,2,0,3] \triangleright [y] (\text{fix } f \cdots) [1,2,0,3] \triangleright [y] * 1 \text{ (f(cdr(l)))) } l' \\
\triangleright & [y] * 1 \text{ (f(cdr(l)))) } l' \triangleright [y] * 1 \text{ (f(cdr(l)))) } l' \triangleright [y] * 1 \text{ (f(cdr(l)))) } l' \triangleright [y] * 1 \text{ (f(cdr(l)))) } l' \\
\triangleright & 0.
\end{align*}
\]

Instead of \( \mathcal{P}_1 \), when we use \( \mathcal{G} \) in the above, the program \( \mathcal{G}(\text{prod } l) \ f \ g \) computes \( g 0 \) if \( l \) contains 0, otherwise \( f \ n \) where \( n \) is the product of \( l \).

(ii) Let \( \text{quot } m \ n : \text{int be} \)

\[
(\text{fix}. \lambda a b. \text{if } a < b \text{ then } 0 \text{ else } * 1 \text{ (f (- a b b)) } m \ n
\]

where \( m, n : \text{int} \). Using Example 3, define \( g a b : \text{int} \rightarrow \text{string} \) by

\[
\text{if } b = 0 \text{ then } \text{inr}(''\text{error}'' ) \text{ else } \text{inl}(\text{quot } a b).
\]

To compute the quotient, \( \text{Quot } m \ n : \text{string} \) is defined as \( \text{when}(g m n, [x_1] \text{makestring}(x_1), [x_2] x_2) \).

6 Church-Rosser Property of \( \lambda^v_{\text{exc}} \)

In this section, we prove that \( \lambda^v_{\text{exc}} \) has the Church-Rosser property by the well-known method of parallel reductions [Bare84][Plot75][Taka89] and the Lemma of Hindley-Rosen, see [Bare84].

Proposition 17 (Church-Rosser Theorem) \( \text{If } M \overset{v_{\text{exc}}}{\Rightarrow} N_1 \text{ and } M \overset{v_{\text{exc}}}{\Rightarrow} N_2 \), then \( N_1 \overset{v_{\text{exc}}}{\Rightarrow} M' \text{ and } N_2 \overset{v_{\text{exc}}}{\Rightarrow} M' \text{ for some } M' \).

To prove this proposition, define two parallel reductions, \( \overset{1}{\Rightarrow} \text{ and } \overset{2}{\Rightarrow} \text{, on } \lambda_{\text{exc}} \text{-terms, for technical reasons (commutativity of the two parallel reductions).} \)
follows:

If we then derive $\lambda x.M \mapsto_1 \lambda x.N$; then $\text{raise } M \mapsto_1 \text{raise } N$; if $M_i \mapsto_1 N_i$ ($i = 1, 2$), then $M_1M_2 \mapsto_1 N_1N_2$; and some $\lambda x.M \mapsto_1 \lambda x.N$.

(1) $x \mapsto_1 x$;
(2) $\lambda x.M \mapsto_1 \lambda x.N$;
(3) $M \mapsto_1 N$, then $\text{raise } M \mapsto_1 \text{raise } N$;
(4) $M_i \mapsto_1 N_i$ ($i = 1, 2$), then $M_1M_2 \mapsto_1 N_1N_2$;
(5) if $M \mapsto_1 N_1$ and $V \mapsto_1 N_2$ then $(\lambda x.M)V \mapsto_1 N_1[x := N_2]$;
(6) if $M_i \mapsto_1 N_i$, then $(\text{raise } M_i)M_2 \mapsto_1 \text{raise } N_1$ for any $M_2$;
(7) if $M_1 \mapsto_1 N_1$, then $V(\text{raise } M_1) \mapsto_1 \text{raise } N_1$ for any $V$;
(8) $M_i \mapsto_1 N_i$ ($i = 1, 2$), then $([y]M_1)M_2 \mapsto_1 [y]((N_1[y \leftarrow N_2])N_2)$;
(9) if $V \mapsto_1 N_1$ and $M \mapsto_1 N_2$, then $V([y]M) \mapsto_1 [y]((N_1(N_2[N_1 \rightarrow y]))$;
(10) if $M \mapsto_1 N$, then $[y]M \mapsto_1 [y]N$;
(11) if $M \mapsto_1 N$, then $yM \mapsto_1 yN$;
(12) if $M \mapsto_1 N$, then $y(\text{raise } M) \mapsto_1 N$;
(13) if $M \mapsto_1 N$, then $y[y_1]M \mapsto_1 yN[y_1 := y]$. \\

Lemma 11 If $V \mapsto_1 M$, then $M$ is a value.

If $M \mapsto_1 N_1$ and $V \mapsto_1 N_2$, then $M[x := V] \mapsto_1 N_1[x := N_2]$.

If $M_i \mapsto_1 N_i$ ($i = 1, 2$), then $M_1[y \leftarrow M_2] \mapsto_1 N_1[y \leftarrow N_2]$.

If $M \mapsto_1 N_1$ and $V \mapsto_1 N_2$, then $M[V \rightarrow y] \mapsto_1 N_1[N_2 \rightarrow y]$.

Lemma 12 For any $N$, if we have $M \mapsto_1 N$, then $N \mapsto_1 M^{*1}$ for some $M^{*1}$.

Proof. By induction on the derivation of $\mapsto_1$. Here, $M^{*1}$ can be inductively given as follows:

(1) $x^{*1} = x$;
(2) $(\lambda x.M)^{*1} = \lambda x.M^{*1}$;
(3) $(\text{raise } M)^{*1} = \text{raise } M^{*1}$;
(4-1) $(\lambda x.M)V^{*1} = M^{*1}[x := V^{*1}]$;
(4-2) $(\text{raise } M)N^{*1} = \text{raise } M^{*1}$;
(4-3) $(V(\text{raise } M))^{*1} = \text{raise } M^{*1}$;
(4-4) $([y]M)N^{*1} = [y]((M^{*1}[y \leftarrow N^{*1}])N^{*1})$;
(4-5) $(V([y]M))^{*1} = [y](V^{*1}(M^{*1}[V^{*1} \rightarrow y]))$;
(4-6) $(MN)^{*1} = M^{*1}N^{*1}$;
(5) $(yM)^{*1} = [y]M^{*1}$;
(6-1) $(y(\text{raise } M))^{*1} = M^{*1}$;
(6-2) $(y[y_1]M)^{*1} = yM^{*1}[y_1 := y]$;
(6-3) $(yM)^{*1} = yM^{*1}$.

To cover the remaining reductions, we define $\mapsto_2$ inductively as follows:

(1) $x \mapsto_2 x$;
(2) $\lambda x.M \mapsto_2 \lambda x.N$;
(3) $M \mapsto_2 N$, then $\text{raise } M \mapsto_2 \text{raise } N$;
(4) $M_i \mapsto_2 N_i$ ($i = 1, 2$), then $M_1M_2 \mapsto_2 N_1N_2$;
(5) if $M_i \mapsto_2 N_i$, then $(\text{raise } M_i)M_2 \mapsto_2 \text{raise } N_1$ for any $M_2$;
(6) if $M_i \mapsto_2 N_i$, then $V(\text{raise } M_i) \mapsto_2 \text{raise } N_1$ for any $V$;
(7) if $M \mapsto_2 N$, then $[y]M \mapsto_2 [y]N$;
(8) if $M \mapsto_2 N$, then $[y](\text{raise } yM) \mapsto_2 [y]N$;
(9) if $M \mapsto_2 N$, then $yM \mapsto_2 yN$;
(10) if $M \mapsto_2 N$, then $y(\text{raise } M) \mapsto_2 yN$. 


Lemma 13 For any $N$, if we have $M \gg_{2} N$, then $N \gg_{2} M^{*2}$ for some $M^{*2}$.

Proof. By induction on the derivation of $\gg_{2}$. Here, $M^{*2}$ can be inductively given as follows:

1. $x^{*2} = x$;
2. $(\lambda x.M)^{*2} = \lambda x.M^{*2}$;
3. $(\text{raise } M)^{*2} = \text{raise } M^{*2}$;
4.1. $(\text{raise } M)N)^{*2} = \text{raise } M^{*2}$;
4.2. $(V(\text{raise } M))^{*2} = \text{raise } M^{*2}$;
4.3. $(MN)^{*2} = M^{*2}N^{*2}$;
5.1. $([y]M)^{*2} = M^{*2}$ if $y \notin FV(M)$;
5.2. $([y](\text{raise } yM))^{*2} = [y]M^{*2}$;
5.3. $([y]M)^{*2} = [y]M^{*2}$;
6.1. $(y(\text{raise } M))^{*2} = M^{*2}$;
6.2. $(yM)^{*2} = yM^{*2}$. □

It is clear that $M \gg_{1} M$ and $M \gg_{2} M$. Let $\gg_{1}^{*}$ and $\gg_{2}^{*}$ be the transitive closures of $\gg_{1}$ and $\gg_{2}$, respectively. Now we can obtain that $\gg_{1}^{*}$ and $\gg_{2}^{*}$ are commutative. For this, it is enough to show the following lemma [Bare84].

Lemma 14 If we have $M \gg_{1} M_{1}$ and $M \gg_{2} M_{2}$, then $M_{2} \gg_{1} N$ and $M_{1} \gg_{2} N$ for some $N$.

Proof. Some of the essential cases are as follows:

Case of $([y](\text{raise } yM))N \gg_{1} [y](\text{raise } y(\text{M}[y \leftarrow N]N)), and ([y](\text{raise } yM))N \gg_{2} ([y]M)N$:

$[y]M[y \leftarrow N]N \gg_{1} [y](\text{M}[y \leftarrow N]N), and [y](\text{raise } y(M[y \leftarrow N]N))N \gg_{2} [y](\text{raise } y(M[y \leftarrow N]N))N$.

Case of $V([y](\text{raise } yM))) \gg_{1} [y]V(\text{raise } y(\text{V}[V \Rightarrow y])), and V([y](\text{raise } yM))) \gg_{2} V([y]M)$:

$V([y]M) \gg_{1} [y]V(\text{M}[V \Rightarrow y])$, and $[y]V(\text{raise } y(\text{V}[V \Rightarrow y])) \gg_{2} [y](\text{raise } y(\text{V}[V \Rightarrow y])) \gg_{2} [y]V([\text{V}[V \Rightarrow y])$.

Case of $y[y_{1}](\text{raise } y_{1}M) \gg_{1} y(\text{raise } y_{1}M[y_{1} := y])$, and $y[y_{1}](\text{raise } y_{1}M) \gg_{2} y[y_{1}](\text{raise } y_{1}M[y_{1} := y]) \gg_{2} y[y_{1}]M[y_{1} := y], and y(\text{raise } y_{1}M[y_{1} := y]) \gg_{2} y[y_{1}]M[y_{1} := y]$. □

From Lemmata 12 and 13, we obtain that $\gg_{1}$ and $\gg_{2}$ have the diamond property, and so have $\gg_{1}^{*}$ and $\gg_{2}^{*}$. Moreover, from Lemma 14 and the Lemma of Hindley-Rosen [Bare84], $(\gg_{1} \cup \gg_{2})^{*}$ has the diamond property. Since we have $(\gg_{1} \cup \gg_{2})^{*} = \gg_{exc}^{*}$, Proposition 18 (Church-Rosser) is confirmed.

7 CPS-Translation of $\lambda_{exc}$-Terms

We provide the translation from a variant of $\lambda_{exc}$ to $\lambda^{-}$, which logically induces Kuroda's translation and is applied to show the strong normalization property with respect to the strict fragment of $\lambda_{exc}$. This translation, with an auxiliary function $\Psi$ for values, comes from Plotkin[Plot75] and de Groote[Groo95]. It is proved that the translation is sound with respect to conversions.

Definition 12 (CPS-translation from $\lambda_{exc}$ to $\lambda^{-}$)

$x = \lambda k.kx; \quad \lambda x.M = \lambda k.(\lambda\bar{M})$;
\[
\overline{yM} = \lambda k. k(\overline{My}); \quad \overline{MN} = \lambda k. \overline{M}(\lambda m. \overline{N}(\lambda n.mnk));
\]
\[\text{raise}(M) = \lambda k. M \lambda x.x; \quad \overline{[y]M} = \lambda y. \overline{My}.\]
\[\Psi(x) = x; \quad \Psi(\lambda x.M) = \lambda x. \overline{M}; \quad \Psi(yV) = y\Psi(V). \quad \square\]

**Lemma 15** For any value \( V \), \( \nabla \triangleright_{\beta}^{v} \lambda k. k(\Psi(V)) \).

**Lemma 16** For any term \( M \) and value \( V \), \( M[x := V] \triangleright_{\beta}^{v} \lambda k. k(\Psi(V)) \).

**Lemma 17** For any term \( M \) where \( k \notin \text{FV}(M) \), \( \lambda k. \overline{M} \triangleright_{\beta}^{v} \lambda k. \overline{M} \).

The above three lemmata can be proved by straightforward induction.

**Lemma 18** For any term \( M \) and \( N \), \( \overline{M[y := \lambda m. \overline{N}(\lambda n.mny)]} = \beta \overline{M[y \leftarrow N]} \).

**Proof.** By induction on the structure of \( M \). We show only the following case:

Case of \( yM \):
\[
\overline{yM[y := \lambda m. \overline{N}(\lambda n.mny)]} = \lambda k. k(\overline{My}[y := \lambda m. \overline{N}(\lambda n.mny)](\lambda m. \overline{N}(\lambda n.mny)))
\]
\[= \beta \lambda k. k(\overline{M[y \leftarrow N]})(\lambda m. \overline{N}(\lambda n.mny)))
\]
\[= \beta \lambda k. k((\lambda k'. M[y \leftarrow N](\lambda m. \overline{N}(\lambda n.mny)))y)
\]
\[= \lambda k. k(\overline{M[y \leftarrow N]N}) = \overline{y(M[y \leftarrow N]N)} = (yM)[y \leftarrow N]. \quad \square\]

**Lemma 19** For any term \( M \) and \( N \), \( \overline{M[y := \lambda n. \Psi(V)ny]} = \beta \overline{M[V \Rightarrow y]} \).

**Proof.** By induction on the structure of \( M \). Only the following case is shown:

Case of \( yM \):
\[
\overline{yM[y := \lambda n. \Psi(V)ny]} = \lambda k. k(\overline{My}[y := \lambda n. \Psi(V)ny])
\]
\[= \beta \lambda k. k(\overline{M[V \Rightarrow y](\lambda n. \Psi(V)ny)})
\]
\[= \beta \lambda k. k((\lambda k'. M[V \Rightarrow y](\lambda n. \Psi(V)(\lambda n.mnk')))(\lambda n. \Psi(V)ny))
\]
\[= \beta \lambda k. k((\lambda k'. (\lambda m. M[V \Rightarrow y](\lambda n.mnk')))(\lambda n. \Psi(V)ny))
\]
\[= \beta \lambda k. k((\lambda k'. (\lambda m. M[V \Rightarrow y](\lambda n.mnk')))(\lambda n. \Psi(V)ny))
\]
\[= \lambda k. k(VM[V \Rightarrow y]y) = \overline{y(M[V \Rightarrow y]y)} = (yM)[V \Rightarrow y]. \quad \square\]

To show the following translation property, we place a restriction such that \( \lambda_{\mathrm{exc}}^{v} \) with (ev3-1)' \( y(\text{raise } V) v_{\mathrm{exc}}^{v} V \) instead of (ev3-1), for technical reasons.

**Lemma 20** If \( M v_{\mathrm{exc}}^{v} N \), then \( \overline{M} = \beta \overline{N} \).

**Proof.** By induction on the structure of \( M v_{\mathrm{exc}}^{v} N \). We show some of the cases:

(ev5-1) \(([y]M)N \triangleright_{\beta}^{v} ([y]M)[y \leftarrow N]; \quad (\overline{[y]M}N = \lambda k. [y]M(\lambda m. \overline{N}(\lambda n.mnk)) \)
\[\overline{\lambda k. [y]M}(\lambda m. \overline{N}(\lambda n.mnk)) \]
\[= \lambda y. \overline{[y]M(\lambda m. \overline{N}(\lambda n.mnk))} \]
\[= \lambda y. \overline{M[y \leftarrow N]}(\lambda m. \overline{N}(\lambda n.mnk)))y \]
\[= \beta \lambda y. \overline{M[y \leftarrow N]}(\lambda m. \overline{N}(\lambda n.mnk)))y \]
\[= \lambda y. \overline{M[y \leftarrow N]}N = \overline{[y](M[y \leftarrow N])}N \).

(ev5-2) \( V([y]M) v_{\mathrm{exc}}^{v} ([y]M)[V \Rightarrow y]; \quad V([y]M) = \lambda k. V(\lambda m. [y]M(\lambda n.mnk)) \)
λk.\overline{V}(\lambda m.(\lambda y.\overline{M}y)(\lambda n.mnk))
\triangleright_{\beta}\lambda k.\overline{V}(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)
\triangleright_{\beta}\lambda k.(\lambda k_{1}.k_{1}\Psi(V))(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)
\triangleright_{\beta}\lambda k.(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)\Psi(V)
\triangleright_{\beta}\lambda k.\overline{M}[y := \lambda n.\Psi(V)nk]\lambda n.\Psi(V)nk
= λy.\overline{M}[y := \lambda n.\Psi(V)ny]\lambda n.\Psi(V)ny
= β λy.(λm.\overline{M}[V \Rightarrow y](\lambda n.mny))\Psi(V)
= β λy.(\lambda k.\overline{V}(\lambda m.\overline{M}[V \Rightarrow y](\lambda n.mnk)))y
= λy.\overline{V}(\overline{M}[V \Rightarrow y])y = [y]\overline{V}(\overline{M}[V \Rightarrow y]). □

Now we have confirmed the soundness of the translation in the sense that equivalent
λ_{\text{exc}}^{v}\text{-terms are translated into equivalent } \lambda\text{-terms.}

Proposition 18 (Soundness of the CPS-Translation)
If we have \( M =_{\text{exc}} N \), then \( \overline{M} =_{\beta} \overline{N} \).

The translation logically establishes the double-negation translation of Kuroda.

Definition 13 (Kuroda's Translation)
\( A^{q} = A \) where \( A \) is atomic; \( (A \to B)^{q} = A^{q} \to \neg\neg B^{q} \).
\((x : A, \Gamma)^{q} = x : A^{q}, \Gamma^{q} \); \( (y : \neg A, \Gamma)^{q} = y : \neg A^{q}, \Gamma^{q} \). □

Proposition 19 If we have \( \Gamma \vdash_{\lambda_{\text{exc}}} M : A \), then \( \Gamma^{q} \vdash_{\lambda} \overline{M} : \neg\neg A^{q} \).

It is also derived that \( \lambda_{\text{exc}}^{v} \) is consistent in the sense that there is no closed term \( M \) of
\( \vdash_{\lambda_{\text{exc}}} M : \bot \), and hence no closed term of the form \( \text{raise}(M) \) either.

8 \( \lambda_{\text{exc}}^{v} \) with Signature

From the programming side we extend \( \lambda_{\text{exc}}^{v} \) with a signature. The signature is used to
introduce constants or to declare global variables, such as exception constructors (names
of exceptions) in ML or special variables in LISP. In the following, the term \([c]M\) is
treated as a packet which can be opened by a reduction. We show that \( \lambda_{\text{exc}}^{v} \) with a certain
signature can simulate computations of type-free \( \lambda \)-calculus.

\( \lambda_{\text{exc}}^{v} + \Sigma \):
\( A ::= \alpha | \bot | A \to A ; \)
\( \Gamma ::= \langle \rangle | x : A, \Gamma | y : \neg A, \Gamma ; \) \( \Sigma ::= \langle \rangle | c : A, \Sigma ; \)
\( M ::= x | c | \lambda x.M | yM | MM | \text{raise}(M) | [y]M | [c]M ; \)
\( V ::= x | c | \lambda x.M | yV | cV ; \)

\[ \begin{align*}
\Gamma \vdash \Sigma c & : \Sigma(c) \\
\Gamma \vdash x : \Gamma(x) & \quad \frac{\Gamma \vdash \Sigma M : A}{\Gamma \vdash \Sigma yM : \bot} \quad \mathrm{if } \Gamma(y) \equiv \neg A \not\equiv \bot \\
\Gamma, x : A \vdash \Sigma M : B & \quad \frac{\Gamma, x : A \vdash \Sigma M : B}{\Gamma \vdash \Sigma \lambda x.M : A \to B} \quad (\rightarrow I) \\
\Gamma \vdash M : \bot & \quad \frac{\Gamma \vdash \Sigma M : \bot}{\Gamma \vdash \Sigma \text{raise}(M) : A} \quad \mathrm{if } A \not\equiv \bot \\
\Gamma, y : \neg A \vdash \Sigma M : A & \quad \frac{\Gamma, y : \neg A \vdash \Sigma M : A}{\Gamma \vdash \Sigma [y]M : A} \quad (\text{exc}) \\
\Gamma \vdash \Sigma [c]M : A & \quad \frac{\Gamma \vdash \Sigma M : A}{\Gamma \vdash \Sigma [c]M : A} \quad (\text{Exc}) \quad \mathrm{if } \Sigma(c) \equiv \neg A
\end{align*} \]
Since the occurrence $c$ in the definition of the reduction rules is treated as if it were a global variable, we computationally call $(Exc)$ a rule of global exception handling. In terms of ML, let exception $c$ of $A$ in $M$ handle $(c \, x) \Rightarrow x$ may be regarded as the term $[c : \neg A]M$ rather than $[y : \neg A]M$\footnote{See also footnote 4.}. Among the reduction rules, (ev6-1) is essentially used for encoding type-free $\lambda$-calculus in the next subsection.

### 8.1 Computational Use of “Inconsistency”

We show that the computation of $\vdash_\beta_N$ in type free $\lambda$-calculus can be simulated in $\lambda^e_{exc}$ with the following signature. This simulation would be regarded as a computational use of logical inconsistency.

#### Definition 14 (lam and app)

Let $\star$ be $(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$. Let $\Sigma_e$ be $E : \neg (\star \rightarrow \star)$. Let $F$ be $\lambda x_1 x_2 . x_2$ and $id$ be $\lambda x . x$.

\[
\text{lam} = \lambda x. \text{raise}(E) : (\star \rightarrow \star) \rightarrow \star; \quad \text{app} = \lambda x_1 x_2 . ([E] F(x_1 \, id)) x_2 : \star \rightarrow \star \rightarrow \star.
\]

For a term of type free $\lambda$-calculus:

\[
M ::= x \mid \lambda x . M \mid MM
\]

the following encoding into $\star$ is defined by using $\text{lam}$ and $\text{app}$.

#### Definition 15 (Encoding of Type-Free $\lambda$-Calculus in $\lambda^e_{exc} + \Sigma_e$)

$[\_] : \text{Terms} \rightarrow \star$ is defined as follows:

\[
[x] = x; \quad [\lambda x . M] = \text{lam}(\lambda x . [M]); \quad [MN] = \text{app} [M] [N]. \quad \diamond
\]

#### Proposition 20

Let $V$ be a value, i.e., a variable or a $\lambda$-abstraction.

(1) $[M[x := N]] \equiv [M][x := [N]]$.

(2) $[V]$ is also a value.

(3) $\text{app}(\text{lam}(V)) \triangleright_{e_{exc}} \lambda v . V \, v$ where $v$ is fresh.

(4) If we have $M \triangleright_{\beta_N} N$ in the type-free $\lambda^e$-calculus à la [Plot75], then $[M] \triangleright_{e_{exc}} [N]$ in $\lambda^e_{exc} + \Sigma_e$.

Proof. We verify only (4):

\[
[(\lambda x . M) V] \equiv \text{app}(\text{lam}(\lambda x . [M])) [V] \equiv \text{app}(\lambda v . \text{raise}(E(\lambda x . [M]))) [V] \triangleright_{e_{exc}} ([E] F(\text{raise}(E(\lambda x . [M]))) \triangleright_{e_{exc}} ([E] \text{raise}(E(\lambda x . [M]))) [V] \triangleright_{e_{exc}} (\lambda x . [M]) [V] \triangleright_{e_{exc}} [M][x := [V]] \equiv [M[x := V]]. \quad \square
\]

In the above proof, (ev6-1) with the call-by-value computation is essentially necessary. For instance, Turing’s fixed point combinator $Y \equiv (\lambda x . f(xxf)) \lambda x . f(xxf)$ can be simulated as $[Y V] \triangleright_{e_{exc}} [V (V V)]$ for any $V$. This encoding would be regarded as a counterpart of [Lil95] that simulates recursive types with exceptions of ML.

Now the system $\lambda^e_{exc}$ with the signature becomes logically inconsistent, so that $\lambda \star$ with Girard’s paradox $[\text{Coq86}]$[Howe87] can also be interpreted in this system by a similar method. Of course, this encoding is impossible in $\lambda^e_{exc}$ with empty signatures, which is logically consistent.
9 Comparison with Related Work

We briefly compare $\lambda^v_{exc}$ with some of the existing call-by-value styles: $\lambda^-_{exc}$ of de Groote[Groo95], and $\lambda^\nu$ of Felleisen[FFKD86][FH92]. The comparison reveals some similarities and distinctions between them.

9.1 Relation to $\lambda^-_{exc}$ of de Groote

Based on classical propositional logic, P.de Groote [Groo95] introduced the simply typed $\lambda$-calculus $\lambda^-_{exc}$ for formalizing the exception-handling mechanism as in $ML$. At first appearance, $\lambda^v_{exc}$ is a small subsystem of $\lambda^-_{exc}$, and the two systems seem similar; however, quite different permutative reduction rules are used in them.

In the following, we consider a simplified version of $\lambda^-_{exc}$ [Groo95]. The term is defined by two distinct variables, $x$ ($\lambda$-variables) and $y$ (exception variables only with negation type):

$M ::= x | y | \lambda x.M | MM | (\text{raise}\ M) | \langle y.M[x.M]\rangle$.

The value is defined as follows:

$V ::= x | \lambda x.M | yV$.

The typing rules are $(\rightarrow I), (\rightarrow E), (\perp E)$, and the following excluded middle.

$$\frac{\Gamma, y:\neg A \vdash M : B \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \langle y.M[x.N]\rangle : B}$$

The reduction rules\(^9\) are $\triangleright_{\lambda^\nu}$, (raise\_left) (i.e., ev2-2), (raise\_right) (i.e., ev2-1), and

(\text{handle\_simple}) : $\langle y.V[x.N]\rangle \triangleright_{\text{ern}} V$ if $y \notin FV(V)$;

(\text{handle\_raise}) : $\langle y.\text{raise}\ yV[x.N]\rangle \triangleright_{\text{ern}} \langle y.N[x := V]\rangle[x.N];$

(\text{handle\_left}) : $V\langle y.M[x.N]\rangle \triangleright_{\text{ern}} \langle y.VM[x.VN]\rangle;$

(\text{handle\_right}) : $\langle y.M[x.N]\rangle O \triangleright_{\text{ern}} \langle y.MO[x.NO]\rangle.$

Now we have the following natural translation from $\lambda^-_{exc}$ to $\lambda^v_{exc}$.

**Definition 16 (Translation from $\lambda^-_{exc}$ to $\lambda^v_{exc}$)**

$(x)^{\circ} = x; \quad (y)^{\circ} = \lambda k. yk; \quad (\lambda x.M)^{\circ} = \lambda x.M^{\circ};$

$(MN)^{\circ} = M^{\circ}N^{\circ}; \quad \text{(raise}\ M)^{\circ} = \text{raise}\ M^{\circ}; \quad (\langle y.M[x.N]\rangle)^{\circ} = [y'](\lambda y.M^{\circ})(\lambda x.y'N^{\circ}). \diamond$

**Proposition 21** If we have $\Gamma \vdash M : A$ in $\lambda^-_{exc}$, then $\Gamma \vdash_{\lambda^v_{exc}} M^{\circ} : A$.

**Lemma 21** If we have $M \triangleright_{\text{ern}} N$ in $\lambda^-_{exc}$, then $M^{\circ} \equiv_{\text{exc}} N^{\circ}$.

The above proposition and lemma can be proved by straightforward induction. In terms of the inverse translation, $\lambda^v_{exc}$ can be regarded as a fragment of $\lambda^-_{exc}$. However, (ev5-1) and (ev5-2) could not be interpreted in $\lambda^-_{exc}$. (handle\_left) and (handle\_right) are simple permutative reductions. On the other hand, (ev5-1) and (ev5-2) are types of permutations, but the segment, in terms of [Praw65], is separated, and we have to shift the lower rule up to the immediately higher one and the separated ones.

**Definition 17 (Translation from $\lambda^v_{exc}$ to $\lambda^-_{exc}$)**

$(x)^{\dagger} = x; \quad (\lambda x.M)^{\dagger} = \lambda x.M^{\dagger}; \quad (MN)^{\dagger} = M^{\dagger}N^{\dagger};$

$(yM)^{\dagger} = yM^{\dagger}; \quad \text{(raise}\ M)^{\dagger} = \text{raise}\ M^{\dagger}; \quad (\langle y.M\rangle)^{\dagger} = \langle y.M^{\dagger}[x.x]\rangle. \diamond$

\(^9\)Here, we take an important subset of the reduction rules from the original $\lambda^-_{exc}$ to discuss the relation.
Proposition 22 If we have $\Gamma \vdash_{\lambda^e_{cc}} M : A$, then $\Gamma \vdash_{\lambda^e_{cc}} M^+ : A$.

Comparing with (ev4-1), (ev4-2), and (handle\_simple), (handle\_raise), the latter rules are restricted to a value\(^\text{10}\). This restriction to a value breaks down the Church-Rosser property. For example, $[y)x_1)x_2$ leads to $x_1x_2$ and $[y)x_1)x_2$ in $\lambda^e_{cc}$ under the restriction, and similarly in $\lambda^e_{ren}$. In contrast, the value restriction makes it possible to simulate (ev4-1) and (ev4-2) by the rules of Felleisen’s $\lambda_c$ as described in the next subsection.

9.2 Relation to Felleisen’s $\lambda_c$

We compare $\lambda^e_{cc}$ with a variant of $\lambda_c$ of Felleisen\(^\text{11}\). We observe that the computation rules in $\lambda_c$ are necessary to simulate some of the compatible rules in $\lambda^e_{cc}$\(^\text{12}\).

According to observations in [RS94], we consider a variant of $\lambda_c$ as follows. The terms and values are defined as usual.

$M ::= x \mid \lambda x.M \mid MM \mid F.M$

The reduction rules are $\beta\lambda$, $(F_L)$, $(F_R)$, and $(F_{top})$ as follows:

$(F_L): (F.M)N \triangleright_{e} F(\lambda k.M(f.k(f.N)))$;
$(F_R): V(F.M) \triangleright_{e} F(\lambda k.M(\lambda f.k(V.f)))$

$(F_{top}): F.M \triangleright_{e} F(\lambda k.M(\lambda f.k))$.

The operator $F$ has the type $\rightarrow A \rightarrow A$, which is a variant of $\rightarrow$ and can be defined by Felleisen’s $\mathcal{C}$, see [RS94]. In addition, the computation rule is $(F_T): F.M \triangleright_{T} M(\lambda x.x)$, which is applied only at the top-level.

Definition 18 (Translation from $\lambda_c$ to $\lambda^e_{cc}$)\(^\text{13}\)

$x = x$; \hspace{1cm} $\lambda x.M = \lambda x.M$; \hspace{1cm} $M_1M_2 = M_1M_2$; \hspace{1cm} $F.M = [y]\text{raise}(M(\lambda x.yx))$. \hspace{1cm} $\Diamond$

Proposition 23 If $\Gamma \vdash_{\lambda_c} M : A$, then $\Gamma \vdash_{\lambda^e_{cc}} M : A$.

With regard to the reduction rules, $(F_{top})$ can be translated such that $F(\lambda k.M(\lambda f.k)) = [y]\text{raise}((\lambda k.M(\lambda f.k))\lambda x.yx) \triangleright_{e} [y]\text{raise}(M(\lambda f.yf)) = F.M$. However, $(F_L)$ and $(F_R)$ could not be simulated in $\lambda^e_{cc}$. The reason may be explained by the definition of (ev5-1) and (ev5-2).

In the definition, the permutations $[y \leftarrow N]$ and $[N \Rightarrow y]$ can be replaced with the substitutions $[y := \lambda x.y(xN)]$ and $[y := \lambda x.y(Nx)]$, respectively (denoted by (ev5-1'), (ev5-2')). Then $(F_L)$ and $(F_R)$ can be simulated in $\lambda^e_{cc}$. In a call-by-name system, the above replacement gives no mismatch, since we have $M[y := \lambda x.y(xN)] \triangleright_{\lambda_{\mathit{T}}} M[y \Rightarrow N]$ and $M[y := \lambda x.y(Nx)] \triangleright_{\lambda_{\mathit{T}}} M[N \Rightarrow y]$. However, in a call-by-value system, the situation is not exactly the same. We do not know whether the CPS-translation in section 4 can also be established, even with (ev5-1') and (ev5-2').

A proof of double-negation elimination is used to interpret $F$ in the above and $C$ in [Groo94]. We often adopt the following operational semantics [FFKD86][FH92]: $\mathcal{E}(CM) \triangleright M(\lambda x.A(\mathcal{E}[x]))$. This rewriting rule can be simulated in part by a proof of Peirce’s law $\mathcal{P}_1$, instead of a double-negation elimination. Consider the case $M$ of $\lambda k.\mathcal{E}'[kV]$ where $k \notin \mathcal{FV}(<V>)$. Then $\mathcal{E}[\mathcal{C}\lambda k.\mathcal{E}'[kV]] \triangleright^* \mathcal{E}[\mathcal{C}\lambda \mathcal{E}'[V]] \triangleright^* \mathcal{E}[V]$, and $\mathcal{E}[\mathcal{P}_1\lambda k.\mathcal{E}'[kV]] \triangleright^* \mathcal{E}[\mathcal{P}_1\lambda \mathcal{E}'[V]] \triangleright^* \mathcal{E}[V]$. When $k \notin \mathcal{FV}(M)$, we have that $\mathcal{E}[\mathcal{C}\lambda k.M] \triangleright^* M$, and $\mathcal{E}[\mathcal{P}_1\lambda k.M] \triangleright^* \mathcal{E}[M]$. In this sense, $\mathcal{P}_1$ behaves like call/cc, for instance see [HD93], rather than $C$.

Definition 19 (Translation from $\lambda^e_{cc}$ to $\lambda_c$)\(^\text{14}\)

$(x) = x$; \hspace{1cm} $\lambda x.M = \lambda x.M$; \hspace{1cm} $M_1M_2 = M_1M_2$; \hspace{1cm} $[y]M = y(M)$; \hspace{1cm} $\langle\text{raise}M\rangle = F(\lambda v.(M))$; \hspace{1cm} $\langle[y]M\rangle = F(\lambda y.y(M))$. \hspace{1cm} $\Diamond$

\(^{10}\)This restriction seems to be not essential in $\lambda^e_{ren}$, by personal communication from P.de Groote.

\(^{11}\)See also 4.3.3. with respect to a call-by-name version of $\lambda_c$.

\(^{12}\)Of course, any reduction rule in $\lambda^e_{cc}$ is compatible.
Proposition 24 If $\Gamma \vdash_{\lambda_{exc}} M : A$, then $\Gamma \vdash_{\lambda} \langle M \rangle : A$.

With respect to the reduction rules, (ev2-1) and (ev2-2) can be simulated by $(F_l)$ and $(F_R)$, respectively. In contrast, the compatible rules (ev4-1) and (ev4-2) with the restriction to a value, as mentioned in the previous subsection, can be simulated by the use of the non-compatible $(F_\gamma)$. $\lambda_c$ can simulate (ev5-1') and (ev5-2'), but with a value restriction such that the term before the reduction has the form $([yV]N$ and $V([y]V'))$, respectively. Finally, the remaining rules (ev3-1) and (ev3-2) with the value restriction of $([y[y]V])$ can be simulated in $\lambda_c$ by using the following reduction rule $(F''_n)$: $y(FM) \triangleright M(\lambda x. yx)$, where the type of $y$ is of the form $\rightarrow A$. This rule is a special form of $C_R^n$ in Barbanera and Berardi [BB93]. Here, $y(FM) \triangleright_{exc}^{''} M \lambda x. yx$. Moreover, using $(F_{top})$ and $(F''_R)$, we have that $\langle EM \rangle = \langle M \rangle$. We also have that $\langle [yV] \triangleright_{exc}^{''} [y][V] \rangle$, and $\langle \text{raise } M \rangle \triangleright_{exc}^{''} \text{raise } \langle M \rangle$. From the above observations, $\lambda_{exc}$ with the value restrictions and $\lambda_c$ with $(F''_n)$ have, in some sense, an isomorphism with respect to conversions.

10 Concluding Remarks

We have shown a simple natural deduction system $\lambda_{exc}$ of classical propositional logic according to our observations of $LJK$ proofs in sequent calculus. We have proved proof theoretical and computational properties of $\lambda_{exc}$. The Church-Rosser property and the Strong Normalization hold in the calculus, and there is an isomorphism between $\lambda_{exc}$ and $\lambda \mu$ with respect to conversions. We have shown that from the existence of $LJK$ proofs there is a strict fragment of $\lambda_{exc}$, which is complete with respect to classical provability and would serve as a standard form of classical proofs. Here, we observed that the invariant to be applied by the right contraction rules, in term of sequent calculus, computationally corresponds to the type of exceptional parameter, and the type can be specified as a strictly positive subformula with respect to $\rightarrow$ and $\wedge$. Such a simple fragment is also available in other systems, like $\lambda \mu$, $\lambda_{exc}$ [Groote95], etc. Moreover, this fragment would serve as a useful guide to writing programs as classical proofs.

We also have provided the call-by-value calculus, $\lambda_{exc}'$, based on classical propositional logic. There is a strict fragment of the form $MC$ in $\lambda_{exc}'$, which would represent some standard form of classical proofs. We also observed that every strictly positive subformula with respect to $\rightarrow$ can be the type of value to be passed on, which makes it possible to implement a simple exit mechanism. To model the exception-handling of $ML$, we have extended $\lambda_{exc}$ with a signature, so that the computation of type-free $\lambda$-calculus can be simulated in it.

To find similarity between $\lambda_{exc}$ and $\lambda_c$, we placed a value restriction on $\lambda_{exc}$. The notion of values has to be reconsidered. The term of the form $yV$ is regarded as a value following de Groot[Groote95], which is based on some analogy of exceptions in $ML$. However, the mechanism of exception handling in $\lambda_{exc}$ and $\lambda_{exc}$ is different from that in $ML$, which has great resemblance to the global exception in $\lambda^{\omega} + \Sigma$. A simple exit mechanism can be implemented mainly by (ev4-1) and (ev4-2). Here, in (ev4-2): $[y](\text{raise } yM) \triangleright [y][V]$, the term $M$ that is passed on and is an argument of $y$ is not restricted to a value for establishing the Church-Rosser property.\footnote{Instead of (ev4-2), if we had $[y](\text{raise } yV) \triangleright [y]V$, then $([y](\text{raise } yx_1))x_2 \triangleright [y](\text{raise } y(x_1x_2))$ and $x_1x_2$ (not confluent).} Without the loss of the Church-Rosser property, this observation may lead to the assumption of another point such that $yM$ is a value instead of $yV$. Nevertheless, the CPS-translation of $\lambda_{exc}$-terms is also obtained more
easily with a minor modification, and moreover, we can obtain that $M \triangleright_{\beta}^{*} N$ if $M \triangleright_{exe}^{\nu} N$ without (ev5-1) and (ev5-2)\textsuperscript{14}.

Besides the Strong Normalization of $\lambda_{exe}$, there are other problems to be considered. $\lambda_{exe}^{\nu} + \Sigma_e$ can interpret $\lambda*$ as in subsection 8.1. In turn, similar to the CPS-translation in section 7, we can obtain that if $\Gamma \vdash_{\Sigma_e} M : A$ in $\lambda_{exe}^{\nu} + \Sigma_e$, then $\Gamma^* \vdash M^* : \neg \neg A^*$ in $\lambda*$, where the constant $E$ in $\Sigma_e$ can be interpreted using the proof of Girard’s paradox of the type $1 \equiv \Pi x.1 : x$. Here, is there a translation such that if $M \triangleright_{exe}^{\nu} N$ in $\lambda_{exe}^{\nu} + \Sigma_e$, then $Tr(M) =_{\beta} Tr(N)$ in $\lambda*$? The positive answer could show simulation of the $Y$ combinator in $\lambda*$.

Recently, we have become aware of the work by Ong and Stewart [OS97]. They extensively studied a call-by-value programming language based on a call-by-value variant of Parigot’s $\lambda\mu$-calculus[Pari92]. We also have to relate their work to ours, since the call-by-name version $\lambda_{exe}$ is isomorphic to $\lambda\mu$-calculus.

謝辞
1997年3月4日から6日まで京都大学数理解析研究所で開催された非古典論理とKripke意味論の新局面に関する研究会にて、お世話をいただいた鈴木信行氏に感謝致します。

References


\textsuperscript{14}This implies the Strong Normalization of the strict fragment $M_C$ of $\lambda_{exe}$, with neither (ev5-1) nor (ev5-2).

\textsuperscript{15}The revised version will appear in Studia Logica, under the title “On Proof Terms and Embeddings of Classical Substructural Logics”.


