数理解析研究所講究録
1010 巻 1997 年 7-34

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@dumbo.ai.kyutech.ac.jp

Abstract

We provide a natural deduction systems λ_{exc} of classical propositional logic and prove proof theoretical and computational properties of the system. The introduction of λ_{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 λ_{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 λ_{exc}; and an isomorphism between λ_{exc} and Parigot's $\lambda\mu$-calculus. Secondly, we introduce a call-by-value version of λ_{exc} and prove the following properties: the Church-Rosser property; the CPS-translation from λ^{v}_{exc} to $\lambda^{\rightarrow}$ and its correctness; and a computational use of the logical inconsistency in λ^{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 [Grif90][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 $L'J$ [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[1]. 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 subformulae among the given formulae. The subformulae 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 $\neg A \to ((A \supset B) \supset A) \supset A$ and $\to ((A \supset B) \supset A) \supset \neg\neg A$ 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 $\neg(((A \supset B) \supset A) \supset A) \to ((A \supset B) \supset A) \supset A$ in LJ, which is a consequence of Glivenko's theorem.

proof1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to A}{A \to A, B}\;{\scriptstyle \to w}}{\to A, A \supset B}\;{\scriptstyle \to \supset} \qquad A \to A}{(A \supset B) \supset A \to A, A}\;{\scriptstyle \supset \to}}{(A \supset B) \supset A \to A}\;{\scriptstyle \to c}}{\to ((A \supset B) \supset A) \supset A}\;{\scriptstyle \to \supset}$$

proof2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to A}{(A \supset B) \supset A, A \to A}\;{\scriptstyle w \to}}{A \to ((A \supset B) \supset A) \supset A}\;{\scriptstyle \to \supset}}{A \to ((A \supset B) \supset A) \supset A, B}\;{\scriptstyle \to w}}{\to ((A \supset B) \supset A) \supset A, A \supset B}\;{\scriptstyle \to \supset} \qquad A \to A}{(A \supset B) \supset A \to ((A \supset B) \supset A) \supset A, A}\;{\scriptstyle \supset \to}}{\to ((A \supset B) \supset A) \supset A, ((A \supset B) \supset A) \supset A}\;{\scriptstyle \to \supset}}{\to ((A \supset B) \supset A) \supset A}\;{\scriptstyle \to c}$$

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.

[1] The terminology of LJK proofs in this paper was called μ-head form proofs in [Fuji97-1][Fuji97-2]. Both denote the same style of proofs.

In Section 4, according to the existence of LJK proofs, we provide a simple natural deduction system λ_{exc} of classical propositional logic using the classical rule of a variant of the *exc*luded middle. In λ_{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 λ_{exc} to LJK proofs. We also prove that λ_{exc} has the Church-Rosser property. Finally, a comparison with the related work; Parigot's $\lambda\mu$-calculus, λ_Δ of Reholf & Sørensen, and Felleisen's λ_c, is given to make clear a relation and distinction to the other, by which we obtain an isomorphism between λ_{exc} and the $\lambda\mu$-calculus, and the Strong Normalization of λ_{exc}.

In Section 5, we introduce a call-by-value version of λ_{exc}, which is called λ^v_{exc}. We prove that λ^v_{exc} has the Church-Rosser property in Section 6. In Section 7, we provide the CPS-translation of λ^v_{exc}-terms and show the correctness of the translation with respect to conversions. In Section 8, we extend λ^v_{exc} with a signature so that a computation in type-free λ-calculus can be simulated in a system that becomes logically inconsistent. In Section 9, we briefly investigate the relation to some existing systems: $\lambda^{\rightarrow}_{exn}$ of de Groote[Groo95] and Felleisen's λ_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 $\wedge$, and it gives the corresponding assumptions.

Definition 1 (Resolution of Formula) *Let Γ be a sequence of formulae. The rewriting relation $\Rightarrow$ is defined as follows.*
$([\Gamma], A_1 \supset A_2) \Rightarrow ([\Gamma, A_1], A_2)$;
$([\Gamma], A_1 \wedge A_2) \Rightarrow ([\Gamma], A_1); ([\Gamma], A_2)$. $\diamond$

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 $P_0 \equiv ([\ ], A) \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k \equiv ([\Gamma_{k1}], A_{k1}); ([\Gamma_{k2}], A_{k2}); \cdots; ([\Gamma_{kn}], A_{kn}) \Rightarrow \cdots \Rightarrow P_l \equiv ([\Gamma_{l1}], A_{l1}); ([\Gamma_{l2}], A_{l2}); \cdots; ([\Gamma_{lm}], A_{lm})$. This process clearly terminates, and we collect all the second elements by $proj_2$ in each P_i, i.e., $proj_2(P_0) = [A]$, $\cdots$, $proj_2(P_k) = [A_{k1}, \cdots, A_{kn}]$, $\cdots$, $proj_2(P_l) = [A_{l1}, \cdots, A_{lm}]$. The candidate for the invariants $CI(A)$ is defined as a finite list such that $[[proj_2(P_0)], [proj_2(P_1)], \cdots, [A_{k1}, \cdots, A_{kn}], \cdots, [A_{l1}, \cdots, A_{lm}]]$. For each $[A_{k1}, \cdots, A_{kn}]$ in $CI(A)$, the assumption list $Assume([A_{k1}, \cdots, A_{kn}], A)$ is defined as $[[\Gamma_{k1}], \cdots, [\Gamma_{kn}]]$ taking the corresponding assumptions.* $\diamond$

It is clearly stated that for each $P_k \equiv ([\Gamma_{k1}], A_{k1};\ [\Gamma_{k2}], A_{k2};\ \cdots;\ [\Gamma_{kn}], A_{kn})$ on the resolution of A, LJ derives $\rightarrow A$ from $\Gamma_{k1} \rightarrow A_{k1}$, $\cdots$, and $\Gamma_{kn} \rightarrow A_{kn}$. For example, let A be $(\neg\neg B \supset B) \wedge (C \vee \neg C)$. $CI(A) = [[A], [\neg\neg B \supset B, C \vee \neg C], [B, C \vee \neg C]]$. $Assume([B, C \vee \neg C], A) = [[\neg\neg B], [\ ]]$. Let $Peirce$ be $((A \supset B) \supset A) \supset A$. $Assume([A], Peirce) = [(A \supset B) \supset A]$ and $Assume([Peirce], Peirce) = [\ ]$. 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, $A \to Peirce$ and $\neg A \to Peirce$ are derivable in LJ, respectively. Let $Literal(\Gamma)$ be a sequence consisting of literals using all distinct propositional letters in Γ. For example, $Literal(Peirce)$ is A, B or $A, \neg B$ or $\neg A, B$ 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 $\Gamma \to A$ is provable in the propositional fragment of LK, then $\Gamma, Literal(\Gamma, A) \to A$ is provable in LJ for any $Literal(\Gamma, A)$.*

Proof. It is enough to show that "$Literal(A) \to A \vee \neg A$ in LJ" implies "if $\to A$ in LK, then $Literal(A) \to A$ in LJ", which is proved *without* the use of Glivenko's theorem. □.

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 $\Gamma \to \Delta; [A]$, where Δ 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)

$$B \to B;$$

(Structural Rules)

$$\frac{\Gamma \to \Delta; [A]}{C, \Gamma \to \Delta; [A]} (w \to) \qquad \frac{\Gamma \to \ ; [A]}{\Gamma \to B; [A]} (\to w_i) \qquad \frac{\Gamma \to \Delta; [A]}{\Gamma \to \Delta; A, [A]} (\to w_c)$$

$$\frac{C, C, \Gamma \to \Delta; [A]}{C, \Gamma \to \Delta; [A]} (c \to) \qquad \frac{\Gamma \to \Delta; A, A, [A]}{\Gamma \to \Delta; A, [A]} (\to c)$$

$$\frac{\Gamma, C, D, \Pi \to \Delta; [A]}{\Gamma, D, C, \Pi \to \Delta; [A]} (e \to)$$

$$\frac{\Gamma \to A; [A]}{\Gamma \to \ ; A, [A]} (\to s_c) \qquad \frac{\Gamma \to \ ; A, [A]}{\Gamma \to A; [A]} (\to s_i)$$

[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.

$$\frac{\Gamma \to B; [A]_1 \quad B, \Pi \to \Delta; [A]_2}{\Gamma, \Pi \to \Delta; [A]_1, [A]_2} (cut_i) \qquad \frac{\Gamma \to \Delta; A, [A]_1 \quad A, \Pi \to ; [A]_2}{\Gamma, \Pi \to \Delta; [A]_1, [A]_2} (cut_c)$$

(Logical Rules)

$$\frac{C, \Gamma \to \Delta; [A]}{C \wedge D, \Gamma \to \Delta; [A]} (\wedge \to_1) \qquad \frac{D, \Gamma \to \Delta; [A]}{C \wedge D, \Gamma \to \Delta; [A]} (\wedge \to_2)$$

$$\frac{\Gamma \to B; [A] \quad \Gamma \to C; [A]}{\Gamma \to B \wedge C; [A]} (\to \wedge)$$

$$\frac{C, \Gamma \to \Delta; [A] \quad D, \Gamma \to \Delta; [A]}{C \vee D, \Gamma \to \Delta; [A]} (\vee \to)$$

$$\frac{\Gamma \to B; [A]}{\Gamma \to B \vee C; [A]} (\to \vee_1) \qquad \frac{\Gamma \to C; [A]}{\Gamma \to B \vee C; [A]} (\to \vee_2)$$

$$\frac{\Gamma \to B; [A]_1 \quad C, \Pi \to \Delta; [A]_2}{B \supset C, \Gamma, \Pi \to \Delta; [A]_1, [A]_2} (\supset \to) \qquad \frac{B, \Gamma \to C; [A]}{\Gamma \to B \supset C; [A]} (\to \supset)$$

$$\frac{\Gamma \to B; [A]}{\neg B, \Gamma \to ; [A]} (\neg \to) \qquad \frac{B, \Gamma \to ; [A]}{\Gamma \to \neg B; [A]} (\to \neg)$$

Definition 3 (*LJK* **Proofs with Invariants**) *An LJK proof of* $\Gamma \to \Delta; [A]$ *with a set of invariants* Ψ *denoted by* $P_\Psi : \Gamma \to \Delta; [A]$ *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* $\Gamma' \to \Delta'; [A_i]$ *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 $\neg\Psi$ denotes a sequence in some order obtained by all negated formulae in Ψ.

Lemma 2 (Embedding of *LJK* **Proofs)** *If we have* $P_\Psi : \Gamma \to \Delta; [A]$ *in LJK, then* $\Gamma, \neg\Psi \to \Delta$ *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 $\Gamma/\neg A$ be a sequence of deleting all the formulae $\neg A$ from Γ. The following lemma plays an important role in our discussion.

Lemma 3 (From LJ Proofs to LJK Proofs) *If* $\Gamma \to B$ *is provable in* LJ, *then* $\Gamma/\neg A \to B; A$ *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* $P_{\{A\}} : \Gamma \to \Delta; [A]$ *in* LJK, *then there exists a cut-free* LJK *proof of* $\Gamma \to \Delta; A$ *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* $\Gamma \to A$ *in* LK, *then for any* Ψ *in* $CI(A)$, *there is a cut-free* LJK *proof of* $\Gamma \to A$; *with invariants* Ψ.

Proof. Let $[A_1, \cdots, A_n]$ be Ψ in $CI(A)$ and $Assume([A_1, \cdots, A_n], A)$ be $[\Pi_1, \cdots, \Pi_n]$. If $\Gamma \to A$ in LK, then by the observation of Definition 2, we have $S_1 : \Gamma, \Pi_1 \to A_1, \cdots$, and $S_n : \Gamma, \Pi_n \to A_n$ in LK, and moreover, LJ derives $\Gamma \to A$ from $S_1, \cdots$, and S_n. Here, we consider S_i for $1 \le i \le n$ 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 $\Gamma, \Pi_i, \to A_i$ in LK, then $\Gamma, \Pi_i, Literal(\Gamma, \Pi_i, A_i) \to A_i$ in LJ for any $Literal(\Gamma, \Pi, A_i)$. Let Γ, Π, A_i consist of n kinds of propositional letters. Then there are 2^n possibilities of $Literal(\Gamma, \Pi_i, A_i)$. Hence, $2^n - 1$ applications of the cut rules lead to an LJK proof of $\Gamma, \Pi_i \to ; A_i$ with an invariant A_i, and it is to be cut-free by Corollary 1. Thus $\Gamma \to A$; with invariants $[A_1, \cdots, A_n]$ 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* $\to B_1 \vee B_2; [A]$ *is provable with an invariant* A *in* LJK, *then either* $\to B_1; A$ *or* $\to B_2; A$ *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 $\Gamma \to A$; is provable with invariants Ψ, then a formula obtained by replacing each invariant $A_i \in \Psi$ in A with $\neg\neg A_i$ is also provable from Γ in LJ. The obtained formula is denoted by A^Ψ. For instance, see $proof1$ and $proof2$ in the introduction.

Proposition 1 (Double-Negation Translation) *If* $\Gamma \to A$ *is provable in* LK, *then* $\Gamma \to A^\Psi$ *is provable in* LJ *for any* Ψ *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 $\neg\neg A \wedge \neg\neg B \leftrightarrow \neg\neg(A \wedge B)$ in LJ. The notion of invariants explains the double-negation of strictly positive subformulae with respect to $\supset$ and $\wedge$ 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 Π^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 λ_{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 λ_{exc} to a certain proof in λ_{exc}, which corresponds to the notion of LJK proofs with invariants. We also prove that λ_{exc} has the Church-Rosser property. The Strong Normalization of λ_{exc} is obtained as one of the by-products from the existence of an isomorphism between λ_{exc} and Parigot's $\lambda\mu$-calculus [Pari92].

4.1 Natural Deduction System λ_{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* $[A_1, \cdots, A_n]$ *be in* $CI(B)$, *and* $Assume([A_1, \cdots, A_n], B)$ *be* $[[\Pi_1], \cdots, [\Pi_n]]$. $\Gamma \to B$ *in* LK *iff* $\Gamma, \Pi_i, \neg A_i \to A_i$ *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 $\Gamma \to B$ from an intuitionistic proof of $\Gamma, \Pi, \neg A \to A$. 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 $\to$. The terms are defined by two kinds of variables x and y, where y is used only for negation types $\neg A$ defined as $A \to \bot$. $FV(M)$ stands for the set of free variables in M.

λ_{exc}:
Types
$A ::= \alpha \mid \bot \mid A \to A$
Contexts
$\Gamma ::= \langle\,\rangle \mid x\!:\!A, \Gamma \mid y\!:\!\neg A, \Gamma$
Terms
$M ::= x \mid \lambda x.M \mid yM \mid MM \mid raise(M) \mid [y\!:\!\neg A]M$
Type Assignment

$$\Gamma \vdash x : \Gamma(x) \qquad\qquad \frac{\Gamma, x\!:\!A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\,(\to I)$$

$$\frac{\Gamma \vdash M_1 : A \to B \quad \Gamma \vdash M_2 : A}{\Gamma \vdash M_1 M_2 : B} (\to E) \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash yM : \bot} (\bot I) \ if \ \Gamma(y) \equiv \neg A \not\equiv \neg\bot$$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash raise(M) : A} (\bot E) \ if \ A \not\equiv \bot \qquad \frac{\Gamma, y{:}\neg A \vdash M : A}{\Gamma \vdash [y{:}\neg A]M : A} (exc)$$

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

The system λ_{exc} without (exc) is denoted by $\lambda^{\to\bot}$, and the system $\lambda^{\to\bot}$ without $(\bot E)$ is denoted by $\lambda^{\to}$.

The classical rule (exc) is a variant of the law of the *ex*cluded middle. This rule is introduced independently of $(\bot E)$, which is in contrast to the double-negation elimination rules, such that $(\bot_C)$: infer $\vdash A$ from $\neg A \vdash \bot$ and that $\mathcal{C}$: infer A from $\neg\neg A$. We computationally call the rule (exc) a rule of local *exc*eption-handling. The type A in (exc) is computationally called a type of *exc*eptional parameter.

In the application of $(\bot I)$, $y : \neg A$ in Γ is used as a major premise in the usual sense of $(\to E)$, and only this kind of negative assumption is discharged by (exc). This style of proof is called a regular proof in [Ando95]. In the λ_Δ-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][Ando95], which is also called the structural reduction in [Pari92].

Term Reductions:
(e1) $(\lambda x.M)N \ \triangleright\ M[x := N]$; (e2) $(raise\ M)N \ \triangleright\ (raise\ M)$;
(e3-1) $y(raise\ M) \ \triangleright\ M$; (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) $[y{:}\neg A](raise\ yM) \ \triangleright\ [y{:}\neg A]M$;
(e5) $([y{:}\neg(A \to B)]M)N \ \triangleright\ [y{:}\neg B]((M[y \Leftarrow N])N)$,
where $M[y \Leftarrow N]$ is defined as follows:
$x[y \Leftarrow N] = x$;
$(\lambda x.M)[y \Leftarrow N] = \lambda x.M[y \Leftarrow N]$;
$(yM)[y \Leftarrow N] = y(M[y \Leftarrow N]N)$;
$(y'M)[y \Leftarrow N] = y'(M[y \Leftarrow N])$ if $y' \not\equiv y$;
$(M_1M_2)[y \Leftarrow N] = (M_1[y \Leftarrow N])(M_2[y \Leftarrow N])$;
$(raise\ M)[y \Leftarrow N] = raise(M[y \Leftarrow N])$;
$([y'{:}\neg A']M)[y \Leftarrow N] = [y'{:}\neg A'](M[y \Leftarrow N])$.

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 $\triangleright^*_{exc}$, and the binary relation $=_{exc}$ is defined as the reflexive, symmetric, and transitive closure of $\triangleright$. The relations $\triangleright_\beta$, $\triangleright^*_\beta$ and $=_\beta$ are usually defined.

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

Proposition 4 (Subject Reduction) *Let $\Gamma \vdash_{\lambda_{exc}} M : A$. If $M \triangleright_{exc} N$, then $\Gamma \vdash_{\lambda_{exc}} N : A$.*

Definition 4 (λ_{exc}-Proofs with Invariant) *We say that M is a λ_{exc}-proof with an invariant A_i if for some Γ and A there is a deduction of $\Gamma \vdash_{\lambda_{exc}} M : A$ 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 $\Gamma \to B$ in LK iff $\Gamma, Assume(A_i, B) \to\ ; A_i$ in LJK for any $A_i \in CI(B)$ iff $\Gamma, Assume(A_i, B), \neg A_i \to A_i$ in LJ iff $\Gamma, Assume(A_i, B), \neg A_i \vdash A_i$ in $\lambda^{\to\perp}$. By an application of (exc) where the type of exceptional parameter is A_i, the last statement implies $\Gamma \vdash_{\lambda_{exc}} A$. 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 λ_{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):
$M_C ::= [y]M_I \mid \lambda x.M_C$;
$M_I ::= x \mid \lambda x.M_I \mid M_I M_I \mid y M_I \mid raise(M_I)$.

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

Let $C[\,]$ be a context with a single hole $[\,]$ such that $C[\,] ::= [\,] \mid C[\,]M$. We denote $C[M]$ by the term obtained by replacing $[\,]$ in $C[\,]$ with the term M. Then we have $C[raiseM] \rhd^*_{exc} raiseM$. If $k \notin C[M]$, then we have that $\mathcal{P}\lambda k.C[kM] \rhd^*_{exc} M$, 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 $\mathcal{P}$. This is the reason why the type A in the definition of (exc) is called a type of exceptional parameter. In terms of ML [MTH90], informally $[y:\neg A]M$ may be read as `let exception y of A in M handle (yx) => x end`, based on the correspondence of $\perp$ with *exn* (type of exceptions in ML)[4].

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 λ_{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 λ_{exc}. From the definability in classical logic, the following examples are demonstrated in this strict fragment.

Example 1 (Definition of $\times$) $A \times B = \neg(A \to \neg B)$:
$\langle M, N\rangle = \lambda x.xMN;\quad fst = \lambda x.[y]raise(x\lambda x_1x_2.yx_1);\quad snd = \lambda x.[y]raise(x\lambda x_1x_2.yx_2)$.
Then it is obtained that $fst\langle N_1, N_2\rangle \rhd^*_{exc} N_1$ and $snd\langle N_1, N_2\rangle \rhd^*_{exc} N_2$.

[4] Although we can write and use the ML program `fun Peirce(w) = let exception y of '1α in w(fn z => raise(y z)) handle (y x) => x end` as the proof $\mathcal{P}$, whose type can be inferred as `(('1α -> 'β) -> '1α) -> '1α` by the ML system, the correspondence is *informal* in the sense that ML is a call-by-value language and the occurrence of `y` in `exception y` is treated as a name of an exception rather than a variable, like in $[y]M$. See also section 8.

Example 2 (Definition of +) $A + B = \neg A \to \neg\neg B$:
$inl(M) = \lambda xv.xM$; $inr(M) = \lambda vx.xM$; $when(M, [x_1]N_1, [x_2]N_2) = [y]raise(M(\lambda x_1.yN_1)(\lambda x_2.yN_2))$
$when(inl(M), [x_1]N_1, [x_2]N_2) \rhd^*_{exc} N_1[x_1 := M]$;
$when(inr(M), [x_1]N_1, [x_2]N_2) \rhd^*_{exc} N_2[x_2 := M]$.

Proposition 6 (Church-Rosser Theorem) *If* $M \rhd^*_{exc} N_1$ *and* $M \rhd^*_{exc} N_2$, *then* $N_1 \rhd^*_{exc} M'$ *and* $N_2 \rhd^*_{exc} M'$ *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 λ_{exc}; however, it is also possible in other classical systems, e.g., in the $\lambda\mu$-calculus. This analysis gives a new reduction relation to λ_{exc}, which shifts the invariant into the inside. To establish this translation, we use an auxiliary type system $\lambda^{\to\bot}$ consisting of simply typed λ-calculus with the intuitionistic absurdity rule. The translation is obtained in the following way:
(1) Given a proof M of type A in λ_{exc}. Compute an embedding $G(M)$ into $\lambda^{\to\bot}$;
(2) A proof of $[y{:}\neg A]raise(G(M)\lambda z.yz)$ is a λ_{exc}-proof of A with an invariant A;
(3) To get a λ_{exc}-proof of A with an invariant A_i, apply the shift reduction (to be defined later) i-times to $[y{:}\neg A]raise(G(M)\lambda z.yz)$, where $CI(A)$ is $[A_0, \cdots, A_n]$ and $0 \le i \le n$.

Definition 5 *The embedding of* G *from the proof terms of* λ_{exc} *to* $\lambda^{\to\bot}$ *is defined.*
$G(x) = \lambda k.kx$;
$G(\lambda x.M) = \lambda k.k(\lambda x.raise(G(M)(\lambda m.k(\lambda v.m))))$;
$G(yM) = \lambda k.k(G(M)\lambda m.ym)$;
$G(MN) = \lambda k.G(M)(\lambda m.G(N)\lambda n.k(mn))$;
$G(raise(M)) = \lambda k.G(M)\lambda x.x$;
$G([y{:}\neg A]M) = \lambda y.G(M)(\lambda m.ym)$. $\diamond$

Proposition 7 *If we have* $\Gamma \vdash M : A$ *in* λ_{exc}, *then* $\Gamma \vdash G(M) : \neg\neg A$ *in* $\lambda^{\to\bot}$.

Proof. By induction on the derivation. □

We define an invariant shift reduction relation $\rhd_s$ for λ_{exc}-proofs with some invariant, which changes an outer invariant to an inner invariant:
$[y{:}\neg(A \to B)]M \rhd_s \lambda x.[y{:}\neg B](Mx[y := \lambda k.y(kx)])$.
The i applications of $\rhd_s$ are denoted by $\rhd^i_s$ 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 $A_i \equiv A'_i \to A_{i+1}$ for some A'_i where $0 \le i \le n-1$.

Lemma 4 *Let* $[A_0, A_1, \cdots, A_n]$ *be* $CI(A)$. *If we have* $\Gamma \vdash M : A$ *in* λ_{exc}, *then for any* i *in* $0 \le i \le n$, M' *such that* $[y{:}\neg A]raise(G(M)\lambda k.yk) \rhd^i_s M'$ *is a* λ_{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 λ_{exc}, then $\Gamma \vdash G(M) : \neg\neg A$ in $\lambda^{\rightarrow\perp}$. Hence, $[y\!:\!\neg A]raise(G(M)\lambda k.yk)$ is a λ_{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 λ_{exc}-proof of A with an invariant A_k where $A_k \equiv A'_k \rightarrow A_{k+1}$. Then $\lambda x_1 \cdots x_k.[y\!:\!\neg A_k]N \rhd_s M'$ gives a λ_{exc}-proof of A with the invariant A_{k+1} by the following replacement of each yO with $(\lambda k.y(kx))O$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{[y:\neg A_k]^1 \quad \overset{\vdots}{O : A_k}}{yO : \perp}}{\vdots} }{\dfrac{N : A_k}{[y\!:\!\neg A_k]N : A_k}\,1}}{\lambda x_1 \cdots x_k.[y\!:\!\neg A_k]N : A}
\quad \rhd_s \quad
\dfrac{\dfrac{\dfrac{\dfrac{\overset{\vdots}{\dfrac{\dfrac{[y : \neg A_{k+1}]^1 \quad \dfrac{[k : A_k]^2 \quad [x : A'_k]^3}{zx : A_{k+1}}}{\lambda k.y(kx) : \neg A_k}\,2 \qquad \overset{\vdots}{O : A_k}}{(\lambda k.y(kx))O : \perp}}}{N[y := \lambda k.y(kx)] : A_k} \qquad [x : A'_k]^3}{N[y := \lambda k.y(kx)]x : A_{k+1}}}{[y\!:\!\neg A_{k+1}](N[y := \lambda k.y(kx)]x) : A_{k+1}}\,1}{\dfrac{\lambda x_1 \cdots x_k.[y\!:\!\neg A_{k+1}](N[y := \lambda k.y(kx)]x) : A}{\lambda x x_1 \cdots x_k.[y\!:\!\neg A_{k+1}](N[y := \lambda k.y(kx)]x) : A}\,3} \quad \Box
$$

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 $\rhd_s$. 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 $\vee$ is defined in terms of the other connectives. Moreover, with respect to λ_{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 $\wedge$). In the more general case of adding a primitive $\vee$, it would *not* be possible to postulate (exc) only for an atomic formula.

It is stated that $\rhd_s$ and (e5) have a strong connection, such that $([y\!:\!\neg(A \rightarrow B)]M)N \rhd_s (\lambda x.[y : \neg B](M[y := \lambda z.y(zx)]x))N \rhd_\beta [y : \neg B](M[y := \lambda z.y(zN)]N)$, which leads to the same result as the one by (e5), since we have that $M[y := \lambda z.y(zN)] \rhd^*_\beta M[y \Leftarrow N]$.

4.3 Comparison with Related Work

In the following subsection, we briefly compare λ_{exc} with some of the existing ones (not a call-by-value style); $\lambda\mu$-calculus[Pari92], λ_Δ[RS94], and a variant of λ_c[FFKD86]. As regards the relation between $\lambda\mu$ and λ_{exc}, we can obtain an isomorphism between them, and the Strong Normalization of λ_{exc}. Our observation on the relation between λ_Δ and λ_{exc} suggests a generalization of some reduction rule of λ_Δ, which can lead to an isomorphism between them. In relation to λ_c, we discuss that adding what kind of reduction rule to λ_c makes them isomorphic.

4.3.1 Relation to Parigot's $\lambda\mu$-Calculus

To study computational interpretations of classical proofs, Parigot [Pari92] 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, λ_Δ[RS94], λ_{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 δ of $\bot$, e.g., the term $\lambda x_1.\mu\alpha.[\delta](x_1(\lambda x_2.\mu\delta.[\alpha]x_2))$ of type $\neg\neg A \to A$ has a free name δ. 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 λ_{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 $\to$. The context Γ and terms are defined as usual. The set of types with names is denoted by Δ.

$\lambda\mu$:
$\Gamma ::= \langle\,\rangle \mid x\!:\!A, \Gamma;$
$\Delta ::= \langle\,\rangle \mid A^\alpha, \Delta;$
$M ::= x \mid MM \mid \lambda x.M \mid [\alpha]M \mid \mu\alpha.M;$

$$\Gamma; \Delta \vdash x : \Gamma(x)$$

$$\frac{\Gamma, x\!:\!A; \Delta \vdash M : B}{\Gamma; \Delta \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma; \Delta \vdash M : A \to B \quad \Gamma; \Delta \vdash N : A}{\Gamma; \Delta \vdash MN : B}$$

$$\frac{\Gamma; \Delta \vdash M : A}{\Gamma; \Delta, A^\alpha \vdash [\alpha]M : \bot} \qquad \frac{\Gamma; \Delta, A^\alpha \vdash M : \bot}{\Gamma; \Delta \vdash \mu\alpha.M : A}\ if\ A \not\equiv \bot$$

The reduction relation $\triangleright_\mu$ of β-reductions, structural reductions, $(S1)$, and $(S2)$ in [Pari93-1] is considered, namely,
$(\lambda x.M)N \triangleright_\mu M[x := N]$;
$(\mu\alpha.M)N \triangleright_\mu \mu\alpha.M[\alpha \Leftarrow N]$;
$(S1)$: $[\alpha]\mu\beta.M \triangleright_\mu M[\beta := \alpha]$;
$(S2)$: $\mu\alpha.[\alpha]M \triangleright_\mu M$ if $\alpha \notin FreeName(M)$.
The binary relations $\triangleright_\mu^*$ and $=_\mu$ are usually defined. As by-products, we obtain the Strong Normalization property of λ_{exc} and an isomorphism between λ_{exc} and $\lambda\mu$ with respect to conversions.

Definition 6 (Translation from λ_{exc} to $\lambda\mu$)
$\underline{x} = x;$ $\quad \underline{\lambda x.M} = \lambda x.\underline{M};$
$\underline{yM} = [y]\underline{M};$ $\quad \underline{MN} = \underline{M}\ \underline{N};$
$\underline{raise(M)} = \mu\alpha.\underline{M}$ *where* α *is a fresh name;* $\quad \underline{[y]M} = \mu y.[y]\underline{M}$. $\diamond$

For this translation, we separate a context in λ_{exc} into two parts as follows:
$\Gamma ::= \Gamma_1 \mid \Gamma_2;$
$\Gamma_1 ::= \langle\,\rangle \mid x\!:\!A, \Gamma_1;$ $\quad \Gamma_2 ::= \langle\,\rangle \mid y\!:\!\neg A, \Gamma_2.$
$\underline{y\!:\!\neg A, \Gamma_2} = A^y, \underline{\Gamma_2}.$

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

Lemma 5 *For any* λ_{exc}*-term* M, $\underline{M}[x := \underline{N}] = \underline{M[x := N]}$.

Lemma 6 $\underline{M}[y \Leftarrow \underline{N}] = \underline{M[y \Leftarrow N]}$

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

Lemma 7 *If* $M \rhd_{exc} N$, *then* $\underline{M} \rhd_\mu \underline{N}$.

Proof. By induction on the derivation $M \rhd_{exc} N$. □

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

Corollary 3 *Well-typed* λ_{exc}*-terms are strongly normalizable.*

Definition 7 (Translation from $\lambda\mu$ to λ_{exc})
$<x> = x;\quad <\lambda x.M> = \lambda x.<M>;\quad <MN> = <M><N>;$
$<[\alpha]M> = \alpha <M>;\quad <\mu\alpha.M> = [\alpha]raise(<M>).$
$<A^\alpha, \Delta> = \alpha{:}\neg A, <\Delta>.$ ◇

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

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* $M \rhd_\mu N$, *then* $<M> \rhd^*_{exc} <N>$.

Proof. By induction on the derivation $M \rhd_\mu N$. □

Proposition 10 *For any* λ_{exc}*-term* M, $<\underline{M}> \rhd^*_{exc} M$.
For any $\lambda\mu$*-term* N, $\underline{<N>} \rhd^*_\mu N$.

Proof. By the definitions of the translations. □

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

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

In terms of the right structural rules of sequent calculus, the operator μ in $\lambda\mu$ works both for the right contraction and the right weakening. In λ_{exc}, the right contraction can be simulated by (exc), and the right weakening by $(\bot I)$ and $(raise)$. The logical aspect of the operator μ can be split into two primitive ones of λ_{exc}, 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 λ_Δ-Calculus of Rehof and Sørensen

[5]Of course, we can establish the strong normalization property of λ_{exc} directly.

For the purpose of establishing the Curry-Howard isomorphism in classical logic, Rehof and Sørensen [RS94] introduced the λ_Δ-calculus by restriction of Felleisen's control operator $\mathcal{C}$ to avoid a breakdown of neat properties like the Church-Rosser property. The λ_Δ-calculus is natural and has good properties not only of proof theory but also of typed calculus. In relation to λ_{exc}, the λ_Δ-calculus treats both regular proofs and non-regular proofs, in other words, there is no distinction of variables that are bound by λ-abstraction or Δ-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 Δ is replaced with $\lambda x.yx$. To study the relation between λ_Δ and λ_{exc}, we consider the λ_Δ-proofs under this modification.

The definition of λ_Δ[RS94] is briefly given below. The syntax of λ_Δ-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 β-reductions.

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

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

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

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

$$\frac{\Gamma, x{:}A \to \bot \vdash M : \bot}{\Gamma \vdash \Delta x.M : A}\ (\bot_c)$$

Definition 8 (Translation from λ_Δ to λ_{exc})
$x^\circ = x$; $(\lambda x.M)^\circ = \lambda x.M^\circ$;
$(MN)^\circ = M^\circ N^\circ$; $(\Delta x.M)^\circ = [x] raise M^\circ$. $\diamond$

Proposition 11 *(1) If we have $\Gamma \vdash_{\lambda_\Delta} M : A$, then $\Gamma \vdash_{\lambda_{exc}} M^\circ : A$.*
(2) If we have $M \rhd N$ in λ_Δ, then $M^\circ =_{exc} N^\circ$ in λ_{exc}.

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

Definition 9 (Translation from λ_{exc} to λ_Δ)
$x^+ = x$; $(\lambda x.M)^+ = \lambda x.M^+$;
$(yM)^+ = yM^+$; $(MN)^+ = M^+N^+$;
$(raise M)^+ = \Delta d.M^+$ *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 $M \rhd_{exc} N$, then $M^+ =_\Delta N^+$ in λ_Δ, where $=_\Delta$ is the reflexive, symmetric, and transitive closure of $\rhd$ in λ_Δ, our reduction rule of (e4-2) fails even if we drop (e3-1) and (e3-2). Our observation suggests adding a new reduction to λ_Δ, instead of (d3), such that $\Delta x.x\Delta d.M \rhd \Delta x.M$ 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 λ_c [FFKD86] by λ_Δ (call-by-value variant), which is observed in [RS94] is not lost. Then we can obtain that $M^+ \rhd^* N^+$ in λ_Δ if $M \rhd_{exc} N$ without (e3-1) and (e3-2). Moreover, we have that $(M^+)^\circ \rhd_{exc}^* M$ and that $(M^\circ)^+ \rhd^* M$ in λ_Δ with (d4) instead of (d3). Hence, as in Corollary 4, there is an isomorphism between λ_{exc} witout (e3-1),(e3-2) and λ_Δ with (d4) instead of (d3).

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

$y\Delta x.M \rhd M[x := y]$,

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

4.3.3 Relation to a variant of λ_c-Calculus of Felleisen[6]

For reasoning about a call-by-value language, Felleisen, et al. [FFKD86][FH92] introduced the λ_c-calculus extending the type-free λ_v-calculus of Plotkin [Plot75] with the control operator $\mathcal{C}$ and the abort operator $\mathcal{A}$. By Griffin [Grif90] the λ_c-calculus has been applied to extend the Curry-Howard isomorphism to classical logic from a computational interest. It is a distinct point that λ_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 [Groo94] proved that there is an isomorphism between $\lambda\mu$ and a call-by-name variant of λ_c, the relation may be obvious. However, we observe that the computation rules in λ_c are necessary to simulate some of the compatible rules in λ_{exc} and that λ_{exc} would be simulated in λ_c with some reduction rule. According to the observations in [Groo94][RS94], we consider a call-by-name variant of λ_c as follows: The terms are defined as usual.

$M ::= x \mid \lambda x.M \mid MM \mid \mathcal{F}M$

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

(F_L): $(\mathcal{F}M)N \rhd \mathcal{F}(\lambda k.M(\lambda f.k(fN)))$; $\quad (F_{top})$: $\mathcal{F}M \rhd \mathcal{F}(\lambda k.M(\lambda f.kf))$.

The operator $\mathcal{F}$ has the type $\neg\neg A \to A$, which is a variant of and can be defined by Felleisen's $\mathcal{C}$, see [RS94]. In addition, the computation rule is (F_T): $\mathcal{F}M \rhd_T M\lambda x.x$ that is applied only at the top-level.

Definition 10 (Translation from λ_c to λ_{exc})
$\langle x\rangle = x$; $\quad \langle \lambda x.M\rangle = \lambda x.\langle M\rangle$;
$\langle MN\rangle = \langle M\rangle\langle N\rangle$; $\quad \langle \mathcal{F}M\rangle = [y]raise(\langle M\rangle\lambda x.yx)$. $\diamond$

Proposition 13 *(1) If we have $\Gamma \vdash_{\lambda_c} M : A$, then $\Gamma \vdash_{\lambda_{exc}} \langle M\rangle : A$.*
(2) If we have $M \rhd N$ in λ_c, then $\langle M\rangle =_{exc} \langle N\rangle$.

The above proposition can be proved by a straightforward induction.

Definition 11 (Translation from λ_{exc} to λ_c)
$\overline{x} = x$; $\quad \overline{\lambda x.M} = \lambda x.\overline{M}$;
$\overline{yM} = y\overline{M}$; $\quad \overline{MN} = \overline{M}\ \overline{N}$;
$\overline{raiseM} = \mathcal{F}(\lambda v.\overline{M})$ *where v is a fresh variable;* $\quad \overline{[y]M} = \mathcal{F}(\lambda y.y\overline{M})$. $\diamond$

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

With respect to the correctness of the translation, the reduction rules (e2) and (e5) can be simulated by (F_L). We also have that $\langle\overline{M}\rangle \rhd^*_{exc} M$. 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 λ_c by using the following reduction rule F''_R:

[6]See also 9.2 Relation to Felleisen's λ_c.

$y(\mathcal{F}M) \rhd M(\lambda x.yx)$,
where the type of y is of the form $\neg A$. This reduction rule is a special form of C''_R in Barbanera and Berardi [BB93], which is also used in [Groo94] to simulate $(S1)$ of $\lambda\mu$ in the λ_c-calculus. With the help of (F_{top}) and (F''_R), we can show that $\overline{\langle \mathcal{F}M\rangle} = \mathcal{F}(\lambda y.y\mathcal{F}(\lambda v.\overline{\langle M\rangle}\lambda x.yx)) \rhd \mathcal{F}(\lambda y.(\lambda v.\overline{\langle M\rangle}\lambda x.yx)\lambda k.yk) \rhd \mathcal{F}(\lambda y.\overline{\langle M\rangle}(\lambda x.yx))$, and then we have that $\overline{\langle \mathcal{F}M\rangle} = \mathcal{F}M$ in λ_c, which can lead to $\overline{\langle M\rangle} = M$ in λ_c. Hence, there is an isomorphism between λ_c and λ_{exc} without (e4-1) and (e4-2), denoted by $\lambda_c \simeq \lambda_{exc}$, which is consistent with $\lambda_{exc} \simeq \lambda\mu$ (Corollary 4), and $\lambda\mu \simeq \lambda_c$ [Groo94]. However, comparing with the proof of $\lambda\mu \simeq \lambda_c$, the proof of $\overline{\langle M\rangle} = M$ in λ_c needs one more reduction rule, i.e., F''_R, which would reveal another aspect of the relation between λ_{exc} and $\lambda\mu$.

5 Call-by-Value Language λ^v_{exc}

We provide a simple natural deduction system λ^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 λ_{exc} and Parigot's $\lambda\mu$-calculus[Pari92], λ^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, λ-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 β-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
(ev1) $(\lambda x.M)V \ \rhd^v_{exc}\ M[x := V]$;
(ev2-1) $(raise\ M)N \ \rhd^v_{exc}\ (raise\ M)$; (ev2-2) $V(raise\ M) \ \rhd^v_{exc}\ (raise\ M)$;
(ev3-1) $y(raise\ M) \ \rhd^v_{exc}\ M$; (ev3-2) $y([y_1]M) \ \rhd^v_{exc}\ yM[y_1 := y]$;
(ev4-1) $[y]M \ \rhd^v_{exc}\ M$ if $y \notin FV(M)$; (ev4-2) $[y](raise\ yM) \ \rhd^v_{exc}\ [y]M$;
(ev5-1) $([y]M)N \ \rhd^v_{exc}\ [y]((M[y \Leftarrow N])N)$; (ev5-2) $V([y]M) \ \rhd^v_{exc}\ [y](V(M[V \Rightarrow y]))$,
where $M[y \Leftarrow N]$ and $M[N \Rightarrow y]$ are defined respectively as follows:
$x[y \Leftarrow N] = x$;
$(\lambda x.M)[y \Leftarrow N] = \lambda x.M[y \Leftarrow N]$;
$(yM)[y \Leftarrow N] = y(M[y \Leftarrow N]N)$;
$(y'M)[y \Leftarrow N] = y'(M[y \Leftarrow N])$ if $y' \not\equiv y$;
$(M_1M_2)[y \Leftarrow N] = (M_1[y \Leftarrow N])(M_2[y \Leftarrow N])$;
$(raise\ M)[y \Leftarrow N] = raise(M[y \Leftarrow N])$;
$([y' : \neg A']M)[y \Leftarrow N] = [y' : \neg A'](M[y \Leftarrow N])$.
$x[N \Rightarrow y] = x$;
$(\lambda x.M)[N \Rightarrow y] = \lambda x.(M[N \Rightarrow y])$;
$(yM)[N \Rightarrow y] = y(N(M[N \Rightarrow y]))$;
$(y'M)[N \Rightarrow y] = y'(M[N \Rightarrow y])$ if $y' \not\equiv y$;
$(M_1M_2)[N \Rightarrow y] = (M_1[N \Rightarrow y])(M_2[N \Rightarrow y])$;
$(raiseM)[N \Rightarrow y] = raise(M[N \Rightarrow y])$;
$([y']M)[N \Rightarrow y] = [y'](M[N \Rightarrow y])$.

The binary relation $\rhd^{v*}_{exc}$ is defined by the reflexive transitive closure of $\rhd^v_{exc}$, and the congruence relation is denoted by $=^v_{exc}$. The relation $\rhd_{\beta_V}$ 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 $\Gamma \vdash_{\lambda^v_{exc}} M : A$ iff A as a formula is classically provable from Γ.*

Proposition 16 (Subject Reduction) *Let $\Gamma \vdash_{\lambda^v_{exc}} M : A$. If $M \triangleright^v_{exc} N$, then $\Gamma \vdash_{\lambda^v_{exc}} N : A$.*

Although λ^v_{exc} 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 +) $A + B = \neg A \rightarrow B$:
$inl(M) = \lambda x.raise(xM)$; $inr(M) = \lambda v.M$; $when(M, [x_1]N_1, [x_2]N_2) = [y](\lambda x_2.N_2)(M\lambda x_1.yN_1)$.
Then we can obtain the following computation:
$when(inl(V), [x_1]N_1, [x_2]N_2) \triangleright^{v*}_{exc} N_1[x_1 := V]$; $when(inr(V), [x_1]N_1, [x_2]N_2) \triangleright^{v*}_{exc} N_2[x_2 := V]$.

Let a context $\mathcal{E}[\,]$ with a hole $[\,]$ be as follows:
$\mathcal{E}[\,] ::= [\,] \mid V(\mathcal{E}[\,]) \mid (\mathcal{E}[\,])M$.
We denote $\mathcal{E}[M]$ by the term obtained by replacing $[\,]$ in $\mathcal{E}[\,]$ with the term M. Then we have $\mathcal{E}[raiseM] \triangleright^{v*}_{exc} raiseM$ and $\mathcal{E}[[y]raise(yM)] \triangleright^{v*}_{exc} [y]raise(y\mathcal{E}[M])$ where $y \notin FV(M)$. Here, the continuation $\mathcal{E}$ with respect to $[y]raise(yM)$ is accumulated as an argument of y.

Example 4 (Exit Mechanism by a Proof of Peirce's Law)
Let $\mathcal{P}_1$ be $\lambda x_1.[y]x_1(\lambda x_2.raise(yx_2))$ of the type $((A \rightarrow B) \rightarrow A) \rightarrow A$. We consider the following two cases. The first case is called a normal case, and the second is an exceptional case.
(1) Case of $k \notin FV(M)$:
$\mathcal{P}_1\lambda k.M = (\lambda x_1.[y]x_1(\lambda x_2.raise(yx_2)))\lambda k.M \triangleright^{v*}_{exc} [y]M \triangleright^{v}_{exc} M$.
(2) Case of $k \notin FV(\mathcal{E}[V])$:
$\mathcal{P}_1\lambda k.\mathcal{E}[kV] \triangleright^{v*}_{exc} [y]\mathcal{E}[raise(yV)] \triangleright^{v*}_{exc} [y]raise(yV) \triangleright^{v}_{exc} [y]V \triangleright^{v}_{exc} V$.

In the second case, the context $\mathcal{E}[\,]$ is abandoned, and the value V to be passed on has the same type as that of the exceptional parameter of $\mathcal{P}_1$, which can be applied to implement a simple exit mechanism. This is the reason why type A in the definition of (exc) is a type of exceptional parameter. In terms of ML [MTH90], informally $[y\!:\!\neg A]M$ may be read as `let exception y of A in M handle (yx) => x end`, based on the correspondence of $\bot$ with exn (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) $\mathcal{L} \equiv \lambda xg.[y]g(x(\lambda k.raise(yk))) : ((A \rightarrow B) \rightarrow C) \rightarrow (C \rightarrow A) \rightarrow A$

$\mathcal{L}V_1V_2$ provides the following computation: If V_1 returns a normal value V, then the result of $\mathcal{L}V_1V_2$ is V_2V. If V_1 raises an exception with a value V', then the entire result becomes V'. That is, $\mathcal{L}$ computes a composition of V_2 and V_1 of a normal case. This type is a substitution instance of Łukasiewicz's formula.

(2) $\mathcal{H} \equiv \lambda xf.[y]x(\lambda k.raise(y(fk))) : ((A \rightarrow B) \rightarrow C) \rightarrow (A \rightarrow C) \rightarrow C$

[7] See also section 8.

$\mathcal{H}V_1V_2$ gives the following computation: If V_1 returns a normal value V, then the whole result is V. If V_1 raises an exception with a value V', then the result of $\mathcal{H}V_1V_2$ becomes V_2V'. Namely, $\mathcal{H}$ can be regarded as a handler of an exceptional case.

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

$\mathcal{G}$ 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:
`if true then` M `else` $N \rhd M$, `if false then` M `else` $N \rhd N$;
`fix` $f.M \rhd M[f := \texttt{fix}f.M]$;

$$\frac{\Gamma, f{:}A \to B, x{:}A \vdash M : B}{\Gamma \vdash \texttt{fix}f.\lambda x.M : A \to B}\ (fix)$$

(i) Let `prod` be
$\lambda l'.\lambda exit.$`(fix`$f.\lambda l.$ `if` $l = nil$ `then 1`
`else if car(`l`)=0 then` $exit$ `0` `else * (car(`l`)) (`f`(cdr(`l`))))`l'
with the type `int list` $\to$ `(int` $\to$ `int)` $\to$ `int`.
To compute the product of all integers in the integer list l, using Example 4 we define `Prod` as $\lambda l.\mathcal{P}_1$`(prod` l`)` with the type `int list` $\to$ `int`. `Prod(`l`)` makes it possible to return 0 immediately as an exception if l contains 0. For instance, we compute neither `* 1 2` nor `* 0 3` in the following:
`Prod [1,2,0,3]` $\rhd^* [y]$ `(`$fix\, f.\cdots$`)[1,2,0,3]` $\rhd^* [y]$ `* 1 ((fix`$f.\cdots$`)[2,0,3])` $\rhd^* [y]$ `* 1 (* 2 (fix`$f.\cdots$`)[0,3])` $\rhd^* [y]$ `* 1 (* 2 (raise (`y `0)))` $\rhd^* [y]$ `raise (`y `0)` $\rhd^*$ `0`.
Instead of $\mathcal{P}_1$, when we use $\mathcal{G}$ in the above, the program $\mathcal{G}$`(prod` l`) f g` computes `g 0` if l contains `0`, otherwise `f n` where `n` is the product of l.

(ii) Let `quot m n : int` be
`(fix`$f.\lambda ab.$ `if` a`<`b `then 0 else + 1 (`f `(-` a b`)` b`)) m n`
where `m,n : int`. Using Example 3, define `g a b : int + string` by
`if b=0 then inr(''error'') else inl(quot a b)`.
To compute the quotient, `Quot m n : string` is defined as `when(g m n,` $[x_1]$`makestring(`x_1`),` $[x_2]x_2$`)`.

6 Church-Rosser Property of λ^v_{exc}

In this section, we prove that λ^v_{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) *If $M \rhd^{v*}_{exc} N_1$ and $M \rhd^{v*}_{exc} N_2$, then $N_1 \rhd^{v*}_{exc} M'$ and $N_2 \rhd^{v*}_{exc} M'$ for some M'.*

To prove this proposition, define two parallel reductions, $\gg_1$ and $\gg_2$, on λ_{exc}-terms, for technical reasons (commutativity of the two parallel reductions).

(1) $x \gg_1 x$;
(2) if $M \gg_1 N$, then $\lambda x.M \gg_1 \lambda x.N$;
(3) if $M \gg_1 N$, then $\mathit{raise}\ M \gg_1 \mathit{raise}\ N$;
(4) if $M_i \gg_1 N_i$ $(i = 1, 2)$, then $M_1 M_2 \gg_1 N_1 N_2$;
(5) if $M \gg_1 N_1$ and $V \gg_1 N_2$ then $(\lambda x.M)V \gg_1 N_1[x := N_2]$;
(6) if $M_1 \gg_1 N_1$, then $(\mathit{raise}\ M_1)M_2 \gg_1 \mathit{raise}\ N_1$ for any M_2;
(7) if $M_1 \gg_1 N_1$, then $V(\mathit{raise}\ M_1) \gg_1 \mathit{raise}\ N_1$ for any V;
(8) if $M_i \gg_1 N_i$ $(i = 1, 2)$, then $([y]M_1)M_2 \gg_1 [y]((N_1[y \Leftarrow N_2])N_2)$;
(9) if $V \gg_1 N_1$ and $M \gg_1 N_2$, then $V([y]M) \gg_1 [y](N_1(N_2[N_1 \Rightarrow y]))$;
(10) if $M \gg_1 N$, then $[y]M \gg_1 [y]N$;
(11) if $M \gg_1 N$, then $yM \gg_1 yN$;
(12) if $M \gg_1 N$, then $y(\mathit{raise}\ M) \gg_1 N$;
(13) if $M \gg_1 N$, then $y[y_1]M \gg_1 yN[y_1 := y]$.

Lemma 11 *If $V \gg_1 M$, then M is a value.*
If $M \gg_1 N_1$ and $V \gg_1 N_2$, then $M[x := V] \gg_1 N_1[x := N_2]$.
If $M_i \gg_1 N_i$ $(i = 1, 2)$, then $M_1[y \Leftarrow M_2] \gg_1 N_1[y \Leftarrow N_2]$.
If $M \gg_1 N_1$ and $V \gg_1 N_2$, then $M[V \Rightarrow y] \gg_1 N_1[N_2 \Rightarrow y]$.

Lemma 12 *For any N, if we have $M \gg_1 N$, then $N \gg_1 M^{*1}$ for some M^{*1}.*

Proof. By induction on the derivation of $\gg_1$. Here, M^{*1} can be inductively given as follows:
(1) $x^{*1} = x$;;
(2) $(\lambda x.M)^{*1} = \lambda x.M^{*1}$;;
(3) $(\mathit{raise}\ M)^{*1} = \mathit{raise}\ M^{*1}$;;
(4-1) $((\lambda x.M)V)^{*1} = M^{*1}[x := V^{*1}]$,
(4-2) $((\mathit{raise}\ M)N)^{*1} = \mathit{raise}\ M^{*1}$,
(4-3) $(V(\mathit{raise}\ M))^{*1} = \mathit{raise}\ M^{*1}$,
(4-4) $(([y]M)N)^{*1} = [y]((M^{*1}[y \Leftarrow N^{*1}])N^{*1})$,
(4-4) $(V([y]M))^{*1} = [y](V^{*1}(M^{*1}[V^{*1} \Rightarrow y]))$,
(4-6) $(MN)^{*1} = M^{*1}N^{*1}$;;
(5) $([y]M)^{*1} = [y]M^{*1}$;;
(6-1) $(y(\mathit{raise}\ M))^{*1} = M^{*1}$,
(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 $\gg_2$ inductively as follows:
(1) $x \gg_2 x$;
(2) if $M \gg_2 N$, then $\lambda x.M \gg_2 \lambda x.N$;
(3) if $M \gg_2 N$, then $\mathit{raise}\ M \gg_2 \mathit{raise}\ N$;
(4) if $M_i \gg_2 N_i$ $(i = 1, 2)$, then $M_1 M_2 \gg_2 N_1 N_2$;
(5) if $M_1 \gg_2 N_1$, then $(\mathit{raise}\ M_1)M_2 \gg_2 \mathit{raise}\ N_1$ for any M_2;
(6) if $M_1 \gg_2 N_1$, then $V(\mathit{raise}\ M_1) \gg_2 \mathit{raise}\ N_1$ for any V;
(6) if $M \gg_2 N$, then $[y]M \gg_2 [y]N$;
(7) if $M \gg_2 N$, then $[y]M \gg_2 N$ where $y \notin FV(M)$;
(8) if $M \gg_2 N$, then $[y](\mathit{raise}\ yM) \gg_2 [y]N$;
(9) if $M \gg_2 N$, then $yM \gg_2 yN$;
(10) if $M \gg_2 N$, then $y(\mathit{raise}\ M) \gg_2 N$.

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) $(raise\ M)^{*2} = raise\ M^{*2}$;;
(4-1) $((raise\ M)N)^{*2} = raise\ M^{*2}$,
(4-2) $(V(raise\ M))^{*2} = raise\ M^{*2}$,
(4-3) $(MN)^{*2} = M^{*2}N^{*2}$;;
(5-1) $([y]M)^{*2} = M^{*2}$ if $y \notin FV(M)$,
(5-2) $([y](raise\ yM))^{*2} = [y]M^{*2}$,
(5-3) $([y]M)^{*2} = [y]M^{*2}$;;
(6-1) $(y(raise\ M))^{*2} = M^{*2}$,
(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 $([y](raise\ yM))N \gg_1 [y](raise\ y(M[y \Leftarrow N]N))N$, and $([y](raise\ yM))N \gg_2 ([y]M)N$:
$([y]M)N \gg_1 [y]M[y \Leftarrow N]N$, and $[y](raise\ y(M[y \Leftarrow N]N))N \gg_2 [y](raise\ y(M[y \Leftarrow N]N)) \gg_2 [y]M[y \Leftarrow N]N$.
Case of $V([y](raise\ yM)) \gg_1 [y]V(raise\ y(VM[V \Rightarrow y]))$, and $V([y](raise\ yM)) \gg_2 V([y]M)$:
$V([y]M) \gg_1 [y]VM[V \Rightarrow y]$, and $[y]V(raise\ y(VM[V \Rightarrow y])) \gg_2 [y](raise\ y(VM[V \Rightarrow y])) \gg_2 [y]VM[V \Rightarrow y]$.
Case of $y[y_1](raise\ y_1M) \gg_1 y(raise\ yM[y_1 := y])$, and $y[y_1](raise\ y_1M) \gg_2 y[y_1]M$:
$y[y_1]M \gg_1 yM[y_1 := y]$, and $y(raise\ yM[y_1 := y]) \gg_2 yM[y_1 := y]$. □

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 $(\gg_1 \cup \gg_2)^* = \rhd_{exc}^{v*}$, Proposition 18 (Church-Rosser) is confirmed.

7 CPS-Translation of λ_{exc}-Terms

We provide the translation from a variant of λ_{exc}^v to $\lambda^{\rightarrow}$, which logically induces Kuroda's translation and is applied to show the strong normalization property with respect to the strict fragment of λ_{exc}^v. This translation, with an auxiliary function Ψ 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 λ_{exc}^v to $\lambda^{\rightarrow}$)
$\overline{x} = \lambda k.kx$; $\overline{\lambda x.M} = \lambda k.k(\lambda x.\overline{M})$;

$\overline{yM} = \lambda k.k(\overline{M}y)$; $\overline{MN} = \lambda k.\overline{M}(\lambda m.\overline{N}(\lambda n.mnk))$;
$\overline{raise(M)} = \lambda k.\overline{M}\lambda x.x$; $\overline{[y]M} = \lambda y.\overline{M}y$.
$\Psi(x) = x$; $\Psi(\lambda x.M) = \lambda x.\overline{M}$; $\Psi(yV) = y\Psi(V)$. ◇

Lemma 15 *For any value* V, $\overline{V} \rhd^*_\beta \lambda k.k\Psi(V)$.

Lemma 16 *For any term* M *and value* V, $\overline{M[x := V]} \rhd^*_\beta \overline{M}[x := \Psi(V)]$.

Lemma 17 *For any term* M *where* $k \notin FV(M)$, $\lambda k.\overline{M}k \rhd_\beta \overline{M}$.

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:
$\overline{yM}[y := \lambda m.\overline{N}(\lambda n.mny)] = \lambda k.k(\overline{M}[y := \lambda m.\overline{N}(\lambda n.mny)]\lambda m.\overline{N}(\lambda n.mny))$
$=_\beta \lambda k.k(\overline{M[y \Leftarrow N]}\lambda m.\overline{N}(\lambda n.mny))$
$=_\beta \lambda k.k((\lambda k'.\overline{M[y \Leftarrow N]}\lambda m.\overline{N}(\lambda n.mnk'))y)$
$= \lambda k.k(\overline{M[y \Leftarrow N]N}y) = \overline{y(M[y \Leftarrow N]N)} = \overline{(yM)[y \Leftarrow N]}$. □

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:
$\overline{yM}[y := \lambda n.\Psi(V)ny] = \lambda k.k(\overline{M}y)[y := \lambda n.\Psi(V)ny]$
$= \lambda k.k(\overline{M}[y := \lambda n.\Psi(V)ny](\lambda n.\Psi(V)ny))$
$=_\beta \lambda k.k(\overline{M[V \Rightarrow y]}(\lambda n.\Psi(V)ny))$
$=_\beta \lambda k.k((\lambda k'.\overline{M[V \Rightarrow y]}(\lambda n.\Psi(V)nk'))y)$
$=_\beta \lambda k.k((\lambda k'.(\lambda m.\overline{M[V \Rightarrow y]}(\lambda n.mnk'))\Psi(V))y)$
$=_\beta \lambda k.k((\lambda k'.(\lambda k''.k''\Psi(V))(\lambda m.\overline{M[V \Rightarrow y]}(\lambda n.mnk')))y)$
$=_\beta \lambda k.k((\lambda k'.\overline{V}(\lambda m.\overline{M[V \Rightarrow y]}(\lambda n.mnk')))y)$
$= \lambda k.k(\overline{VM[V \Rightarrow y]}y) = \overline{y(VM[V \Rightarrow y])} = \overline{(yM)[V \Rightarrow y]}$. □

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

Lemma 20 *If* $M \rhd^v_{exc} N$, *then* $\overline{M} =_\beta \overline{N}$.

Proof. By induction on the derivation of $M \rhd_{exc} N$. We show some of the cases:
(ev5-1) $([y]M)N \rhd^v_{exc} ([y]M)[y \Leftarrow N]$:
$\overline{([y]M)N} = \lambda k.\overline{[y]M}(\lambda m.\overline{N}(\lambda n.mnk))$
$= \lambda k.(\lambda y.\overline{M}y)(\lambda m.\overline{N}(\lambda n.mnk))$
$\rhd_\beta \lambda k.\overline{M}[y := \lambda m.\overline{N}(\lambda n.mnk)](\lambda m.\overline{N}(\lambda n.mnk))$
$= \lambda y.\overline{M}[y := \lambda m.\overline{N}(\lambda n.mny)](\lambda m.\overline{N}(\lambda n.mny))$
$=_\beta \lambda y.(\lambda k.\overline{M}[y := \lambda m.\overline{N}(\lambda n.mny)](\lambda m.\overline{N}(\lambda n.mnk)))y$
$=_\beta \lambda y.(\lambda k.\overline{M[y \Leftarrow N]}(\lambda m.\overline{N}(\lambda n.mnk)))y$
$= \lambda y.\overline{(M[y \Leftarrow N])N}y = \overline{[y](M[y \Leftarrow N])N}$.

(ev5-2) $V([y]M) \rhd^v_{exc} ([y]M)[V \Rightarrow y]$:
$\overline{V([y]M)} = \lambda k.\overline{V}(\lambda m.\overline{[y]M}(\lambda n.mnk))$

$= \lambda k.\overline{V}(\lambda m.(\lambda y.\overline{M}y)(\lambda n.mnk))$
$\triangleright_\beta \lambda k.\overline{V}(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)$
$\triangleright_\beta^* \lambda k.(\lambda k_1.k_1\Psi(V))(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)$
$\triangleright_\beta \lambda k.(\lambda m.\overline{M}[y := \lambda n.mnk]\lambda n.mnk)\Psi(V)$
$\triangleright_\beta \lambda k.\overline{M}[y := \lambda n.\Psi(V)nk]\lambda n.\Psi(V)nk$
$= \lambda y.\overline{M}[y := \lambda n.\Psi(V)ny]\lambda n.\Psi(V)ny$
$=_\beta \lambda y.\overline{M[V \Rightarrow y]}\lambda n.\Psi(V)ny$
$=_\beta \lambda y.(\lambda m.\overline{M[V \Rightarrow y]}(\lambda n.mny))\Psi(V)$
$=_\beta \lambda y.(\lambda k.\overline{V}(\lambda m.\overline{M[V \Rightarrow y]}(\lambda n.mnk)))y$
$= \lambda y.\overline{V(M[V \Rightarrow y])}y = \overline{[y]V(M[V \Rightarrow y])}$. □

Now we have confirmed the soundness of the translation in the sense that equivalent λ^v_{exc}-terms are translated into equivalent λ-terms.

Proposition 18 (Soundness of the CPS-Translation)
If we have $M =^v_{exc} N$, 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 $\Gamma \vdash_{\lambda^v_{exc}} M : A$, then $\Gamma^q \vdash_{\lambda^{\to}} \overline{M} : \neg\neg A^q$.*

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

8 λ^v_{exc} with Signature

From the programming side we extend λ^v_{exc} 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 λ^v_{exc} with a certain signature can simulate computations of type-free λ-calculus.

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

$$\Gamma \vdash_\Sigma c : \Sigma(c) \qquad \Gamma \vdash_\Sigma x : \Gamma(x) \qquad \frac{\Gamma \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma yM : \bot}\ if\ \Gamma(y) \equiv \neg A \not\equiv \neg\bot$$

$$\frac{\Gamma, x{:}A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x.M : A \to B}\,(\to I) \qquad \frac{\Gamma \vdash_\Sigma M_1 : A \to B \quad \Gamma \vdash_\Sigma M_2 : A}{\Gamma \vdash_\Sigma M_1 M_2 : B}\,(\to E)$$

$$\frac{\Gamma \vdash_\Sigma M : \bot}{\Gamma \vdash_\Sigma raise(M) : A}\ if\ A \not\equiv \bot \qquad \frac{\Gamma, y{:}\neg A \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma [y]M : A}\,(exc)$$

$$\frac{\Gamma \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma [c]M : A}\,(Exc)\ if\ \Sigma(c) \equiv \neg A$$

$(\lambda x.M)V \ \triangleright^v_{exc}\ M[x := V]$;
$V(raise\ M) \ \triangleright^v_{exc}\ (raise\ M)$; $\quad (raise\ M)N \ \triangleright^v_{exc}\ (raise\ M)$;
$y(raise\ M) \ \triangleright^v_{exc}\ M$; $\quad y([y_1]M) \ \triangleright^v_{exc}\ yM[y_1 := y]$;
$[y]M \ \triangleright^v_{exc}\ M$ if $y \notin FV(M)$; $\quad [y](raise\ yM) \ \triangleright^v_{exc}\ [y]M$;
$([y]M)N \ \triangleright^v_{exc}\ [y]((M[y \Leftarrow N])N)$; $\quad V([y]M) \ \triangleright^v_{exc}\ [y](V(M[V \Rightarrow y]))$;
(ev6-1) $[c](raise\ cV) \ \triangleright^v_{exc}\ V$;
(ev6-2) $[c](raise\ c'V) \ \triangleright^v_{exc}\ raise\ c'V$ if $c \not\equiv c'$; $\quad$ (ev6-3) $[c]V \ \triangleright^v_{exc}\ V$.

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) => x end` may be regarded as the term $[c : \neg A]M$ rather than $[y : \neg A]M$[8]. Among the reduction rules, (ev6-1) is essentially used for encoding type-free λ-calculus in the next subsection.

8.1 Computational Use of "Inconsistency"

We show that the computation of $\triangleright_{\beta_V}$ in type free λ-calculus can be simulated in λ^v_{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 $(\alpha \to \alpha) \to \alpha \to \alpha$. Let Σ_e be $E : \neg(\star \to \star)$. Let F be $\lambda x_1 x_2.x_2$ and id be $\lambda x.x$.
$lam = \lambda xv.raise(Ex) : (\star \to \star) \to \star$; $\quad app = \lambda x_1 x_2.([E]F(x_1\ id))x_2 : \star \to \star \to \star$. $\diamond$

For a term of type free λ-calculus:
$M ::= x \mid \lambda x.M \mid MM$
the following encoding into $\star$ is defined by using *lam* and *app*.

Definition 15 (Encoding of Type-Free λ-Calculus in $\lambda^v_{exc}+\Sigma_e$)
$\lceil\ \rceil : Terms \to \star$ *is defined as follows:*
$\lceil x \rceil = x$; $\quad \lceil \lambda x.M \rceil = lam(\lambda x.\lceil M \rceil)$; $\quad \lceil MN \rceil = app\lceil M \rceil \lceil N \rceil$. $\diamond$

Proposition 20 *Let V be a value, i.e., a variable or a λ-abstraction.*
(1) $\lceil M[x := N] \rceil \equiv \lceil M \rceil[x := \lceil N \rceil]$.
(2) $\lceil V \rceil$ *is also a value.*
(3) $app(lam(V)) \ \triangleright^{v*}_{exc}\ \lambda v.Vv$ *where v is fresh.*
(4) If we have $M \triangleright_{\beta_V} N$ in the type-free λ_v-calculus à la [Plot75], then $\lceil M \rceil \ \triangleright^{v*}_{exc}\ \lceil N \rceil$ in $\lambda^v_{exc}+\Sigma_e$.

Proof. We verify only (4):
$\lceil(\lambda x.M)V\rceil \equiv app(lam(\lambda x.\lceil M \rceil))\lceil V \rceil \equiv app(\lambda v.raise(E(\lambda x.\lceil M \rceil)))\lceil V \rceil \triangleright^v_{exc} ([E]F(raise(E(\lambda x.\lceil M \rceil)))$
$\triangleright^v_{exc} ([E]raise(E(\lambda x.\lceil M \rceil)))\lceil V \rceil \triangleright^v_{exc} (\lambda x.\lceil M \rceil)\lceil V \rceil \triangleright^v_{exc} \lceil M \rceil[x := \lceil V \rceil] \equiv \lceil M[x := V] \rceil$. $\square$

In the above proof, (ev6-1) with the call-by-value computation is essentially necessary. For instance, Turing's fixed point combinator $Y \equiv (\lambda xf.f(xxf))\lambda xf.f(xxf)$ can be simulated as $\lceil YV \rceil \triangleright^{v*}_{exc} \lceil V(YV) \rceil$ for any V. This encoding would be regarded as a counterpart of [Lill95] that simulates recursive types with exceptions of ML.

Now the system λ^v_{exc} with the signature becomes logically inconsistent, so that $\lambda*$ with Girard's paradox [Coq86][Howe87] can also be interpreted in this system by a similar method. Of course, this encoding is impossible in λ^v_{exc} with empty signatures, which is logically consistent.

[8] See also footnote 4.

9 Comparison with Related Work

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

9.1 Relation to $\lambda^{\rightarrow}_{exn}$ of de Groote

Based on classical propositional logic, P.de Groote [Groo95] introduced the simply typed λ-calculus $\lambda^{\rightarrow}_{exn}$ for formalizing the exception-handling mechanism as in ML. At first appearance, λ^{v}_{exc} is a small subsystem of $\lambda^{\rightarrow}_{exn}$, 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 $\lambda^{\rightarrow}_{exn}$ [Groo95]. The term is defined by two distinct variables, x (λ-variables) and y (exception variables only with negation type):
$M ::= x \mid y \mid \lambda x.M \mid MM \mid (raise\ M) \mid \langle y.M|x.M\rangle$.
The value is defined as follows:
$V ::= x \mid \lambda x.M \mid yV$.
The typing rules are $(\rightarrow I)$, $(\rightarrow E)$, $(\bot E)$, 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 $\triangleright_{\beta_V}$, ($\mathtt{raise}_{\mathrm{left}}$) (i.e., ev2-2), ($\mathtt{raise}_{\mathrm{right}}$) (i.e., ev2-1), and
($\mathtt{handle}_{\mathrm{simple}}$) : $\langle y.V|x.N\rangle \triangleright_{exn} V$ if $y \notin FV(V)$;
($\mathtt{handle/raise}$) : $\langle y.(raise\ yV)|x.N\rangle \triangleright_{exn} \langle y.N[x := V]|x.N\rangle$;
($\mathtt{handle}_{\mathrm{left}}$) : $V\langle y.M|x.N\rangle \triangleright_{exn} \langle y.VM|x.VN\rangle$;
($\mathtt{handle}_{\mathrm{right}}$) : $\langle y.M|x.N\rangle O \triangleright_{exn} \langle y.MO|x.NO\rangle$.

Now we have the following natural translation from $\lambda^{\rightarrow}_{exn}$ to λ^{v}_{exc}.

Definition 16 (Translation from $\lambda^{\rightarrow}_{exn}$ to λ^{v}_{exc})
$(x)^\circ = x;\quad (y)^\circ = \lambda k.yk;\quad (\lambda x.M)^\circ = \lambda x.M^\circ;$
$(MN)^\circ = M^\circ N^\circ;\quad (raise\ M)^\circ = raise\ M^\circ;\quad (\langle y.M \mid x.N\rangle)^\circ = [y'](\lambda y.M^\circ)(\lambda x.y'N^\circ).$ ◇

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

Lemma 21 *If we have $M \triangleright_{exn} N$ in $\lambda^{\rightarrow}_{exn}$, then $M^\circ =^{v}_{exc} N^\circ$.*

The above proposition and lemma can be proved by straightforward induction. In terms of the inverse translation, λ^{v}_{exc} can be regarded as a fragment of $\lambda^{\rightarrow}_{exn}$. However, (ev5-1) and (ev5-2) could not be interpreted in $\lambda^{\rightarrow}_{exn}$. ($\mathtt{handle}_{\mathrm{left}}$) and ($\mathtt{handle}_{\mathrm{right}}$) 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 both the immediately higher one and the separated ones.

Definition 17 (Translation from λ^{v}_{exc} to $\lambda^{\rightarrow}_{exn}$)
$(x)^+ = x;\quad (\lambda x.M)^+ = \lambda x.M^+;\quad (MN)^+ = M^+N^+;$
$(yM)^+ = yM^+;\quad (raise\ M)^+ = raise\ M^+;\quad ([y]M)^+ = \langle y.M^+ \mid x.x\rangle.$ ◇

[9] Here, we take an important subset of the reduction rules from the original $\lambda^{\rightarrow}_{exn}$ to discuss the relation.

Proposition 22 *If we have* $\Gamma \vdash_{\lambda^v_{exc}} M : A$, *then* $\Gamma \vdash_{\lambda^{\rightarrow}_{exn}} M^+ : A$.

Comparing with (ev4-1), (ev4-2), and ($\mathtt{handle}_{\mathrm{simple}}$), ($\mathtt{handle/raise}$), the latter rules are restricted to a value[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 $[y]x_1x_2$ in λ^v_{exc} under the restriction, and similarly in $\lambda^{\rightarrow}_{exn}$. In contrast, the value restriction makes it possible to simulate (ev4-1) and (ev4-2) by the rules of Felleisen's λ_c as described in the next subsection.

9.2 Relation to Felleisen's λ_c

We compare λ^v_{exc} with a variant of λ_c of Felleisen[11]. We observe that the computation rules in λ_c are necessary to simulate some of the compatible rules in λ^v_{exc}[12].

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

$M ::= x \mid \lambda x.M \mid MM \mid \mathcal{F}M$

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

(F_L): $(\mathcal{F}M)N \triangleright_c \mathcal{F}(\lambda k.M(\lambda f.k(fN)))$; (F_R): $V(\mathcal{F}M) \triangleright_c \mathcal{F}(\lambda k.M(\lambda f.k(Vf)))$;

(F_{top}): $\mathcal{F}M \triangleright_c \mathcal{F}(\lambda k.M(\lambda f.kf))$.

The operator $\mathcal{F}$ has the type $\neg\neg A \rightarrow A$, which is a variant of and can be defined by Felleisen's $\mathcal{C}$, see [RS94]. In addition, the computation rule is (F_T): $\mathcal{F}M \triangleright_T M\lambda x.x$, which is applied only at the top-level.

Definition 18 (Translation from λ_c to λ^v_{exc})

$\underline{x} = x$; $\underline{\lambda x.M} = \lambda x.\underline{M}$; $\underline{M_1M_2} = \underline{M_1}\ \underline{M_2}$; $\underline{\mathcal{F}M} = [y]raise(\underline{M}(\lambda x.yx))$. $\diamond$

Proposition 23 *If* $\Gamma \vdash_{\lambda_c} M : A$, *then* $\Gamma \vdash_{\lambda^v_{exc}} \underline{M} : A$.

With regard to the reduction rules, (F_{top}) can be translated such that $\underline{\mathcal{F}(\lambda k.M(\lambda f.kf))} = [y]raise((\lambda k.\underline{M}(\lambda f.kf))\lambda x.yx) \triangleright^{v*}_{exc} [y]raise(\underline{M}(\lambda f.yf)) = \underline{\mathcal{F}M}$. However, (F_L) and (F_R) could not be simulated in λ^v_{exc}. 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 λ^v_{exc}. In a call-by-name system, the above replacement gives no mismatch, since we have $M[y := \lambda x.y(xN)] \triangleright^*_\beta M[y \Rightarrow N]$ and $M[y := \lambda x.y(Nx)] \triangleright^*_\beta M[N \Rightarrow y]$. 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 $\mathcal{F}$ in the above and $\mathcal{C}$ in [Groo94]. We often adopt the following operational semantics [FFKD86][FH92]: $\mathcal{E}[\mathcal{C}M] \triangleright M(\lambda x.\mathcal{A}(\mathcal{E}[x]))$. 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 $k \notin FV(\mathcal{E}'[V])$. Then $\mathcal{E}[\mathcal{C}\lambda k.\mathcal{E}'[kV]] \triangleright^* \mathcal{E}'[\mathcal{A}(\mathcal{E}[V])] \triangleright^* \mathcal{E}[V]$, and $\mathcal{E}[\mathcal{P}_1\lambda k.\mathcal{E}'[kV]] \triangleright^{v*}_{exc} \mathcal{E}[[y]\mathcal{E}'[raise(yV)]] \triangleright^{v*}_{exc} \mathcal{E}[[y]raise(yV)] \triangleright^{v*}_{exc} [y]raise(y\mathcal{E}[V]) \triangleright^{v*}_{exc} \mathcal{E}[V]$. When $k \notin FV(M)$, we have that $\mathcal{E}[\mathcal{C}\lambda k.M] \triangleright^* M$, and $\mathcal{E}[\mathcal{P}_1\lambda k.M] \triangleright^{v*}_{exc} \mathcal{E}[M]$. In this sense, $\mathcal{P}_1$ behaves like call/cc, for instance see [HDM93], rather than $\mathcal{C}$.

Definition 19 (Translation from λ^v_{exc} to λ_c)

$\langle x \rangle = x$; $\langle \lambda x.M \rangle = \lambda x.\langle M \rangle$; $\langle M_1M_2 \rangle = \langle M_1 \rangle \langle M_2 \rangle$;

$\langle yM \rangle = y\langle M \rangle$; $\langle raiseM \rangle = \mathcal{F}(\lambda v.\langle M \rangle)$; $\langle [y]M \rangle = \mathcal{F}(\lambda y.y\langle M \rangle)$. $\diamond$

[10] This restriction seems to be not essential in $\lambda^{\rightarrow}_{exn}$, by personal communication from P.de Groote.

[11] See also 4.3.3. with respect to a call-by-name version of λ_c.

[12] Of course, any reduction rule in λ^v_{exc} is compatible.

Proposition 24 *If* $\Gamma \vdash_{\lambda^v_{exc}} M : A$, *then* $\Gamma \vdash_{\lambda_c} \langle M \rangle : A$.

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_T). λ_c can simulate (ev5-1') and (ev5-2'), but with a value restriction such that the term before the reduction has the form $([y]V)N$ and $V([y]V')$, respectively. Finally, the remaining rules (ev3-1) and (ev3-2) with the value restriction of $(y[y_1]V)$ can be simulated in λ_c by using the following reduction rule (F''_R): $y(\mathcal{F}M) \triangleright M(\lambda x.yx)$, where the type of y is of the form $\neg A$. This rule is a special form of C''_R in Barbanera and Berardi [BB93]. Here, $\underline{y(\mathcal{F}M)} \triangleright^{v*}_{exc} \underline{M}\lambda x.yx$. Moreover, using (F_{top}) and (F''_R), we have that $\langle \underline{\mathcal{F}M} \rangle = \langle \underline{M} \rangle$. We also have that $\langle \underline{[y]V} \rangle \triangleright^{v*}_{exc} [y]\langle \underline{V} \rangle$, and $\underline{\langle raise\ M \rangle} \triangleright^{v*}_{exc} raise\ \langle \underline{M} \rangle$. From the above observations, λ^v_{exc} with the value restrictions and λ_c with (F''_R) have, in some sense, an isomorphism with respect to conversions.

10 Concluding Remarks

We have shown a simple natural deduction system λ_{exc} of classical propositional logic according to our observations of LJK proofs in sequent calculus. We have proved proof theoretical and computational properties of λ_{exc}. The Church-Rosser property and the Strong Normalization hold in the calculus, and there is an isomorphism between λ_{exc} and $\lambda\mu$ with respect to conversions. We have shown that from the existence of LJK proofs there is a strict fragment of λ_{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 $\lambda\mu$, $\lambda^{\rightarrow}_{exc}$ [Groo95], 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, λ^v_{exc}, based on classical propositional logic. There is a strict fragment of the form M_C in λ^v_{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 λ^v_{exc} with a signature, so that the computation of type-free λ-calculus can be simulated in it.

To find similarity between λ^v_{exc} and λ_c, we placed a value restriction on λ^v_{exc}. The notion of values has to be reconsidered. The term of the form yV is regarded as a value following de Groote[Groo95], which is based on some analogy of exceptions in ML. However, the mechanism of exception handling in $\lambda^{\rightarrow}_{exn}$ and λ^v_{exc} is different from that in ML, which has great resemblance to the global exception in $\lambda^v_{exc} + \Sigma$. A simple exit mechanism can be implemented mainly by (ev4-1) and (ev4-2). Here, in (ev4-2): $[y](raise\ yM) \triangleright [y]M$, 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.[13] 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 λ_{exc}-terms is also obtained more

---

[13]Instead of (ev4-2), if we had $[y](raise\ yV) \triangleright [y]V$, then $([y](raise\ yx_1))x_2 \triangleright^* [y](raise\ y(x_1x_2))$ and x_1x_2 (not confluent).

easily with a minor modification, and moreover, we can obtain that $\overline{M} \triangleright^*_\beta \overline{N}$ if $M \triangleright^v_{exc} N$ without (ev5-1) and (ev5-2)[14].

Besides the Strong Normalization of λ^v_{exc}, there are other problems to be considered. $\lambda^v_{exc}+\Sigma_e$ 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 $\lambda^v_{exc}+\Sigma_e$, then $\Gamma^q \vdash M' : \neg\neg A^q$ in $\lambda*$, where the constant E in Σ_e can be interpreted using the proof of Girard's paradox of the type $\perp \equiv \Pi x:*.x$. Here, is there a translation such that if $M \triangleright^v_{exc} N$ in $\lambda^v_{exc}+\Sigma_e$, 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 λ_{exc} 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

[Ando95] Y.Andou: A Normalization-Procedure for the First Order Classical Natural Deduction with Full Logical Symbols, *Tsukuba Journal of Mathematics*, Vol.19, No.1, pp.153-162, 1995.

[Bare84] H.P.Barendregt: *The Lambda Calculus, Its Syntax and Semantics* (revised edition), North-Holland, 1984.

[BB93] F.Barbanera and S.Berardi: Extracting Constructive Context from Classical Logic via Control-like Reductions, Lecture Notes in Computer Science, 664, pp.45-59, 1993.

[Coq86] T.Coquand: An Analysis of Girard's Paradox, *Proc. 1st Logic in Computer Science*, pp.227-236, 1986.

[Groo94] P.de Groote: On the Relation between the $\lambda\mu$-Calculus and the Syntactic Theory of Sequential Control, Lecture Notes in Artificial Intelligence, 822, pp.31-43, 1994.

[Groo95] P.de Groote: A Simple Calculus of Exception Handling, Lecture Notes in Computer Science, 902, pp.201-215, 1995.

[FFKD86] M.Felleisen, D.P.Friedman, E.Kohlbecker, B.Duba: Reasoning with Continuations, *Proc. Annual IEEE Symposium on Logic in Computer Science*, pp.131-141, 1986.

[FH92] M.Felleisen, R.Hieb: The Revised Report on the Syntactic Theories of Sequential Control and State, *Theoretical Computer Science*, 103, pp.131-141, 1992.

[Fuji94] K.Fujita: Extending NJ with Two Consequences, *The First Workshop on Non-Standard Logics and Logical Aspects of Computer Science (NSL '94)*, Kanazawa, Japan, December 5-8 1994.

[Fuji95] K.Fujita: On Embedding of Classical Substructural Logics[15], *Theory of Rewriting Systems and Its Applications*, Kyoto University, RIMS, Vol.918, pp.178-195, 1995.

[Fuji97-1] K.Fujita: μ-Head Form Proofs and its Application to Programming, *Computer Software*, Vol.14, No.2, pp.71-75, 1997.

[Fuji97-2] K.Fujita: μ-Head Form Proofs with at Most Two Formulas in the Succedent, *Transactions of Information Processing Society of Japan*, Vol.38, No.6, 1997.

[Gira91] J-Y.Girard: A New Constructive Logic: Classical Logic, *Math. Struct. in Comp. Science*, Vol.1, pp.255-296, 1991.

---

[14]This implies the Strong Normalization of the strict fragment M_C of λ^v_{exc}, with neither (ev5-1) nor (ev5-2).

[15]The revised version will appear in *Studia Logica*, under the title "On Proof Terms and Embeddings of Classical Substructural Logics".

[Gira93] J-Y.Girard: On the Unity of Logic, *Annals of Pure and Applied Logic*, 59, pp.201-217, 1993.

[Grif90] T.G.Griffin: A Formulae-as-Types Notion of Control, *Proc. 17th Annual ACM Symposium on Principles of Programming Languages*, pp.47-58, 1990.

[HDM93] R.Harper, B.F.Duba, D.MacQueen: Typing First-Class Continuations in ML, *J.Functional Programming*, 3 (4) pp.465-484, 1993.

[HN88] S.Hayashi, N.Nakano: *PX A Computational Logic*, The MIT Press, 1988.

[Howa80] W.Howard: *The Formulae-as-Types Notion of Constructions, To H.B.Curry: Essays on combinatory logic, lambda-calculus, and formalism*, Academic Press, pp.479-490, 1980.

[Howe87] D.J.Howe: The Computational Behaviour of Girard's Paradox, *Proc. 2nd Logic in Computer Science*, pp.205-214, 1987.

[Koba93] S.Kobayashi: Monads, Modality and Curry-Howard Principle, *Proc. 10th Japan Society for Software Science and Technology*, pp.225-228, 1993.

[Lill95] M.Lillibridge: *Exceptions Are Strictly More Powerful Than Call/CC*, Carnegie Mellon University, CMU-CS-95-178, 1995.

[Mae54] S.Maehara: Eine Darstellung Der Intuitionistischen Logik In Der Klassischen, *Nagoya mathematical journal*, pp.45-64, 1954.

[MTH90] R.Milner, M.Tofte, and R.Harper: *The Definition of Standard ML*, The MIT Press, 1990.

[Murt91] C.R.Murthy: An Evaluation Semantics for Classical Proofs, *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, pp.96-107, 1991.

[Naka95] H.Nakano: *Logical Structures of the Catch and Throw Mechanism*, PhD thesis, University of Tokyo, 1995.

[NPS90] B.Nordström, K.Petersson and J.M.Smith: *Programming in Martin-Löf's Type Theory, An Introduction*, Clarendon Press, 1990.

[NS93] A.Nerode and R.A.Shore: *Logic for Applications*, Springer-Verlag, 1993.

[Ong96] C.-H.L.Ong: A Semantic View of Classical Proofs: Type-Theoretic, Categorical, and Denotational Characterizations, *Linear Logic '96 Tokyo Meeting*, 1996.

[OS97] C.-H.L.Ong and C.A.Stewart: A Curry-Howard Foundation for Functional Computation with Control, *Proc. 24th Annual ACM SIGPLAN-SIGACT Symposium of Principles of Programming Languages*, 1997.

[Pari92] M.Parigot: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science, 624, pp.190-201, 1992.

[Pari93-1] M.Parigot: Classical Proofs as Programs, Lecture Notes in Computer Science, 713, pp.263-276, 1993.

[Pari93-2] M.Parigot: Strong Normalization for Second Order Classical Natural deduction, *Proc. 8th Annual IEEE Symposium on Logic in Computer Science*, 1993.

[Plot75] G.Plotkin: Call-by-Name, Call-by-Value and the λ-Calculus, *Theoretical Computer Science*, 1, pp. 125-159, 1975.

[Praw65] D.Prawitz: *Natural Deduction: A Proof-Theoretical Study*, Almqvist&Wiksell, 1965.

[Praw71] D.Prawitz: Ideas and Results in Proof Theory, *Proc. 2nd Scandinavian Logic Symposium*, edited by N.E.Fenstad, North-Holland, pp.235-307, 1971.

[RS94] N.J.Rehof and M.H.Sørensen: The λ_Δ-Calculus, Lecture Notes in Computer Scicence, 789, pp.516-542, 1994.

[Sche91] H.Schellinx: Some Syntactical Observations on Linear Logic, *J.Logic Computat.*, Vol.1, No.4, pp.537-559, 1991.

[Seld89] J.P.Seldin: Normalization and Excluded Middle.I, *Studia Logica XLVIII*, 2, pp.193-217, 1989.

[Szab69] M.E.Szabo: *The Collected Papers of Gerhard Gentzen*, North-Holland, 1969.

[Taka89] M.Takahashi: Parallel Reductions in λ-Calculus, *J.Symbolic Computation*, 7, pp.113-123, 1989.

[Take87] G.Takeuti: *Proof Theory* (second edition), North-Holland, 1987.

[TD88] A.S.Troelstra and D.van Dalen: *Constructivism in Mathematics, An Introduction*, North-Holland, 1988.