# Proofs, Lambda Terms and Control Operators

Helmut Schwichtenberg

Mathematisches Institut, Universität München,
Theresienstraße 39, D-80333 München, Germany.
email schwicht@rz.mathematik.uni-muenchen.de

The so-called continuation-passing-style translation (cps-translation) has been introduced by Fischer [8] for the $\lambda$-calculus and extended to the $\lambda$-calculus with control operators $\mathcal{C}$ and $\mathcal{A}$ by Felleisen et al. in [7]. By giving a typing a connection with implicational propositional logic has been established by Meyer and Wand [13] for the $\lambda$-calculus and extended to the $\lambda$-calculus with control operators $\mathcal{C}$ and $\mathcal{A}$ by Griffin [10]. Griffin has shown that all evaluations with respect to call-by-value $\beta$-conversion and the standard conversion rules for $\mathcal{C}$ and $\mathcal{A}$ terminate. More precisely Griffin extends the Meyer/Wand typing of Fischers cps-translation $\overline{M}$ of a term $M$ to the language involving the control operators $\mathcal{C}$ and $\mathcal{A}$. It still holds that if $M$ has type $A$, then $\overline{M}$ has type $\neg\neg A^\circ$, where $A^\circ$ is defined as $P^\circ := P$ and $(A \to B)^\circ := A^\circ \to \neg B^\circ \to \mathrm{F}$ (which is equivalent to $\neg B^\circ \to \neg A^\circ$). Griffin's proof of termination of evaluation is based on Plotkin's [16] technique of the so-called colon-translation (denoted $M : V$ and typed by $M^A : V^{\neg A^\circ}$) and context unwrapping (denoted $V^E$ and typed by requiring $V$ to be of type $\neg B^\circ$ and the evaluation context $E[]$ to be of type $B$ with the 'hole' of type $A$).

Here we essentially give an exposition of Griffin's result, with some simplifications and extensions based on work of Sabry and Felleisen [18]. In particular we stress its connection with questions of termination of different normalization strategies for minimal, intuitionistic and classical logic, or more precisely their fragments in implicational propositional logic. We also give some examples (due to Hirokawa) of derivations in minimal and classical logic which reproduce themselves under certain reasonable conversion rules.

This work clearly owes a lot to other people. Robert Constable and Chet Murthy have made me aware of the significance of Griffin's paper. In his thesis [14] Murthy has explored the relevance of this approach to the problem of "Extracting Constructive Content from Classical Proofs"; I will also have to say something about this in the introduction below. Thanks are due to Stefano Berardi, from whom I have learned the counterexamples for classical logic due to Sachio Hirokawa. I also would like to thank Ulrich Berger, Ralph Matthes and in particular Felix Joachimski from München, who have contributed significantly to the present notes.

# 1 Introduction

It is well known that from a classical proof of $\forall x \exists y\, B(x,y)$, $B$ quantifier-free, one can extract a program $t$ such that $\forall x\, B(x,tx)$ holds. There are two possibilities to do this (cf. [1, 2]): 1. A direct method, which uses the classical proof and proof normalization directly as an algorithm. 2. A translation of the classical proof into an intuitionistic one from which via a realizability interpretation a program can be extracted. It has been shown that both methods yield the same algorithm.

Furthermore one can try to answer the question if "programs from classical proofs" is a useful device practically. In [1, 2] the proof translation has been applied to a simple but informative example, namely a classical proof that w.r.t. an unbounded function $f{:}\mathbb{N} \to \mathbb{N}$ such that $f(0) = 0$ each $n$ has a root $m$, i.e., $f(m) \leq n < f(m+1)$ holds. This proof can be translated and a program $\mathrm{root}{:}\mathbb{N} \to \mathbb{N}$ (depending on $f$) can be extracted such that $f(\mathrm{root}(n)) \leq n < f(\mathrm{root}(n)+1)$ holds for all $n$. It's interesting that the classical proof is extremely easy and short (even if fully formalized); considerably shorter than the intuitionistic proof one would give intuitively. However the extracted program is unnecessarily complicated. This can be taken as a motivation to study refinements of the proof translation yielding simpler programs.

Program extraction can be messy for mainly two reasons: 1. A completely formalized proof, using the basic axioms of arithmetic only, will in general be extremely long. This can be remedied by introducing additional global assumptions which are of such a form that they do not spoil the extraction. 2. When translating a classical derivation into an intuitionistic one, each atomic formula $P$ is replaced by $(P \to A) \to A$, where $A$ is the existential formula we want to prove. Thus existential formulas are spread all over the derivation and therefore each subderivation gets computational content. This means that the extracted program will be at least as long and complicated as the proof. Furthermore one has to pay for the additional assumptions introduced in 1., since their translations have to be proved. In general, these proofs use case splittings which later show up in the program.

In [1, 2] a refined proof translation has been proposed which does *not* replace *all* atoms $P$ by $(P \to A) \to A$. By a simple syntactical analysis of the assumptions used in the derivation one can determine a set of *critical* atoms which suffice to be replaced in order to make the translation work. This refined translation simplifies the resulting programs drastically.

Another interesting example of a classical existence proof (for a formula with a quantifier-free kernel) which does not obviously contain an algorithm has been suggested by Yiannis Moschovakis: the gcd of two natural numbers $a_1$ and $a_2$ is a linear combination of the two. The usual classical proof is quite short and elegant: The ideal $(a_1, a_2)$ generated from $a_1, a_2$ has a least positive element, since $0 < a_2$. This element has a representation $\mathrm{abs}(k_1 a_1 - k_2 a_2)$

with $k_1, k_2 \in \mathbb{N}$. It is a common divisor of $a_1$ and $a_2$ (since otherwise the remainder of its division by $a_i$ would be a smaller positive element of the ideal), and it is the greatest common divisor (since any common divisor of $a_1$ and $a_2$ must also be a divisor of $\mathrm{abs}(k_1 a_1 - k_2 a_2)$). – It turns out that in this example only the divisibility relation $\cdot|\cdot$ will be critical in the sense above. We have actually carried out the program extraction first by hand and then by the machine in the interactive prover MINLOG under development in Munich. The extracted program is quite short, but somewhat different from Euklid's algorithm; see [3].

It would be interesting to see if this refined method can be applied successfully to larger examples too. A candidate might be the classical proof of Higman's Lemma by Higman [11] and Nash-Williams [15]. This proof has been translated and implemented in the Nuprl system by Murthy [14]. It is not known how the translated proof (which is extremely big) is related to the known constructive proofs of Higman's Lemma by Schütte and Simpson [19] and Coquand [5]. A refined translation might help answering this.

In these notes we will not go any further into these matters, but rather explore another aspect of the computational content of classical proofs, namely its relation to non-local control operators in functional languages (like the well-known `call/cc` in SCHEME). We will try to bring out the close connection to standard axiom schemes for classical logic like the stability scheme or the Peirce scheme. We will prove in detail that any evaluation sequence of a simply typed scheme program terminates.

To make the material accessible to a mixed audience we start from scratch. We begin with the simply typed $\lambda$-calculus, which is the simplest setting for proofs with 'computational content'. The logical system corresponding to it is minimal propositional logic with implication $\to$ as the only logical connective, and the computational content is given by $\beta$-conversion. We present Tait's proof of strong normalization, in a form which avoids informal arguments concerning reduction sequences. This makes it easy to formalize the proof, and to generalize it to more complex systems.

We then show how intuitionistic and classical logic can be embedded into minimal logic, again for simplicity in the implicational fragment of propositional logic. Intuitionistic logic is obtained by adding the ex-falso-quodlibet scheme $\mathrm{F} \to A$, and classical logic by adding the stability scheme $\neg\neg A \to A$, where $\neg A$ abbreviates $A \to \mathrm{F}$. Here falsity F is just a special propositional symbol. Writing $\vdash$ ($\vdash_i$, $\vdash_c$) for derivability in minimal (intuitionistic, classical) logic one can prove the following simple facts.

A formula $A$ is called negative if any propositional symbol in $A$ (different from F) occurs negated. For negative $A$ we have $\vdash \neg\neg A \to A$. For arbitrary formulas this does not hold, since e.g. $\not\vdash \neg\neg p \to p$ ($p$ a propositional symbol). We consider different translations $A \mapsto A'$ such that $\vdash_c A \leftrightarrow A'$ and $\vdash_c A$ iff $\vdash A$. The Gödel-Gentzen negative translation $^g$ is defined by

$$p^g \quad := \quad \neg\neg p,$$

$$\begin{aligned} \mathrm{F}^g &:= \mathrm{F}, \\ (A \to B)^g &:= A^g \to B^g. \end{aligned}$$

Furthermore we define $p^* := p$, $(A \to B)^* := \neg B^* \to \neg A^*$ and $p^\circ := p$, $(A \to B)^\circ := A^\circ \to \neg B^\circ \to \mathrm{F}$. Then $\vdash_c A \leftrightarrow A^g \leftrightarrow A^* \leftrightarrow A^\circ$ and $\vdash_c A \iff \vdash A^g \iff \vdash A^* \iff \vdash A^\circ$.

It has been observed by Meyer and Wand [13] that if $M\colon A$ is a simply typed $\lambda$-term, then Fischers cps-translation $\overline{M}$ of $M$ can be typed by $\overline{M}\colon \neg\neg A^\circ$. Griffin observed that this corresponds to a logical embedding. To see this note that $\vdash_c A \iff \vdash A^\circ$, hence $\vdash_c A \iff \vdash \neg\neg A^\circ$ since $A^\circ$ is negative (if composed) and hence $\vdash \neg\neg A^\circ \leftrightarrow A^\circ$. So we see that from the logical point of view this embedding is something like an overkill, since it introduces more negations than necessary.

We also have $\vdash_c \neg A$ iff $\vdash_i \neg A$ (Kuroda [12]). The latter is false if $\vdash_i$ is replaced by $\vdash$, since e.g. $\not\vdash \neg\neg(\neg\neg p \to p)$.

We then discuss the status of a special case of the Peirce scheme, namely

$$\mathcal{P}\colon (\neg A \to A) \to A.$$

The general Peirce scheme $((A \to B) \to A) \to A$ can be obtained from it by replacing falsity F in $\neg A$ by an arbitrary formula $B$. On the basis of minimal logic it is weaker than stability, for we have

$$\begin{aligned} &\vdash (\neg\neg p \to p) \to (\neg p \to p) \to p, \\ &\not\vdash [(\neg p \to p) \to p] \to \neg\neg p \to p. \end{aligned}$$

If, however, we add ex-falso-quodlibet, then the converse holds:

$$\vdash (\mathrm{F} \to p) \to [(\neg p \to p) \to p] \to \neg\neg p \to p.$$

We then show that some 'reasonable' simplification rules for derivations involving the Peirce scheme and the ex-falso-quodlibet scheme or else the stability scheme lead – together with $\beta$-conversion – to non-termination. These counterexamples are due to Hirokawa.

In order to find computational content in simply typed $\lambda$-terms/proofs involving $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{C}$ we discuss global control operators in functional languages. First we fix a deterministic strategy for normalizing (or evaluating) $\lambda$-terms, the so-called call-by-value strategy. To formulate it we use evaluation contexts of Felleisen et al. [7]. We then explain the call-with-current-continuation operator $\mathcal{P}$ (well-known from the LISP dialect SCHEME), the abort operator $\mathcal{A}$ and Felleisen's control operator $\mathcal{C}$. For $\mathcal{P}$ we give some programming examples. These operators also make sense in a type-free setting. Following Griffin [10] we then show that the operators $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$ can be typed by the Peirce-, ex-falso-quodlibet- and stability-schemes, respectively. So we have a computational meaning of these schemes, and we may try to use

that as a criterium to decide which simplification rules for proofs we should accept.

Based on Griffin's work we then show that the simply typed $\lambda$-calculus extended by the $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$ operators enjoys termination with respect to the call-by-value strategy. The proof involves a cps-translation into the simply typed $\lambda$-calculus and uses strong normalization for the latter. We first introduce a cps-translation for the simply typed $\lambda$-calculus and then extend it to the language involving $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$.

# 2 The simply typed lambda calculus

We first recall Gentzen's natural deduction system, for simplicity restricted to the implicational fragment of propositional logic. We present Tait's proof of strong normalization, in a form which avoids informal arguments concerning reduction sequences. This makes it easy to formalize the proof, and to generalize it to more complex systems.

**2.1.** *Natural deduction.* As our deductive formalism we use the system of natural deduction introduced by Gerhard Gentzen in [9]. In our implicational fragment of propositional logic it consists of the following introduction and elimination rules for $\to$.

For any formula $A$ let countably many *assumption variables* of type $A$ be given. We use $u^A$, $v^A$, $w^A$ to denote assumption variables of type $A$.

The notions of a *derivation term* $d^A$ in minimal logic and its set $\mathrm{FA}(d^A)$ of *free assumption variables* are defined inductively by

**(A)** $u^A$ is a derivation term with $\mathrm{FA}(u^A) = \{u^A\}$.

$(\to^+)$ If $d^B$ is a derivation term, then

$$(\lambda u^A\, d^B)^{A\to B}$$

is a derivation term with $\mathrm{FA}(\lambda u^A\, d^B) = \mathrm{FA}(d^B) \setminus \{u^A\}$.

$(\to^-)$ If $d^{A\to B}$ and $e^A$ are derivation terms, then

$$(d^{A\to B} e^A)^B$$

is a derivation term with $\mathrm{FA}(d^{A\to B} e^A) = \mathrm{FA}(d^{A\to B}) \cup \mathrm{FA}(e^A)$.

It is sometimes useful to display derivation terms in the following graphical fashion. $(\to^+)$:

$$\begin{array}{c} u\colon A \\ | \\ B \\ \hline A \to B \end{array} \to^+ u$$

$(\to^-)$:

$$\frac{\begin{array}{cc} | & | \\ A \to B & A \end{array}}{B}\ \to^-$$

A derivation term $d^A$ is called *closed*, if $\mathrm{FA}(d^A) = \emptyset$. We write

$$d^B[u_1^{A_1}, \ldots, u_n^{A_n}]$$

if the assumption variables free in $d^B$ are in the list $u_1^{A_1}, \ldots, u_n^{A_n}$. We also use the notation $d\colon A$ instead of $d^A$.

**Definition.** *A formula $A$ is called derivable from assumptions $A_1, \ldots, A_n$, if there is a derivation term $d^B[u_1^{A_1}, \ldots, u_n^{A_n}]$ with different assumption variables $u_1^{A_1}, \ldots, u_n^{A_n}$.*

Let $\Gamma$ be a (finite or infinite) set of formulas. We write $\Gamma \vdash B$, if the formula $B$ is derivable from finitely many assumptions $A_1, \ldots, A_n \in \Gamma$. The '$m$' here stands for minimal logic.

**2.2.** *Strong normalization.* We show in this subsection that any derivation $d$ can be transformed by appropriate conversion steps into a normal form. A derivation in normal form has the property that it does not make "detours", or more precisely, that it cannot occur that an elimination rule immediately follows an introduction rule. Derivations in normal form have many pleasant properties, and can be used for a variety of results.

The arguments in this subsection apply to derivations as well as to terms of the simply typed $\lambda$-calculus, which are essentially the same. So let us first introduce the latter.

Let $G$ be a set of ground types (e.g. nat and boole). *Types* (also called object types or simple types) are formed from $G$ by the operation $\rho \to \sigma$. For any type $\rho$ let a countable infinite set of *variables* of type $\rho$ be given. We denote variables of type $\rho$ by $x^\rho, y^\rho, \ldots$.

We define inductively *terms* $M^\rho$ of type $\rho$ and the set $\mathrm{FV}(M^\rho)$ of variables free in $M^\rho$.

- $x^\rho$ is a term of type $\rho$, $\mathrm{FV}(x^\rho) = \{x^\rho\}$.
- If $M$ is a term of type $\sigma$, then $\lambda x^\rho\, M$ is a term of type $\rho \to \sigma$. $\mathrm{FV}(\lambda x^\rho\, M) = \mathrm{FV}(M) \setminus \{x^\rho\}$.
- If $M$ is a term of type $\rho \to \sigma$ and $N$ is a term of type $\rho$, then $MN$ is a term of type $\sigma$. $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$.

For the arguments in this subsection it is convenient to use the following notation.

- Terms are denoted by $M, N, K, \ldots$.
- $x, y, z \ldots$ denote assumption variables as well as object variables.
- We identify terms differing only by the names of their bound variables.
- $\rho, \sigma, \tau \ldots$ denote formulas as well as types, and $\iota$ denotes atomic formulas or F as well as ground types. $\rho \to \sigma$ denotes function types as well as implications. It is also useful to require that $\to$ associates to the right.

We use $M_x^N$ to denote the usual substitution operation.

We now define a *conversion relation* $M \to_0 M'$ between terms.

**Definition.** *$M \to_0 M'$ is defined by*

$$(\lambda x\, M)N \to_0 M_x^N. \qquad (\beta)$$

*A term $M$ is called $\beta$-convertible if it has the form of a left hand side of $(\beta)$. Such terms are also called $\beta$-redex (for reducible expression).*

From $\to_0$ one derives a one-step reduction relation $\to_\beta$ as follows. Intuitively $M \to_\beta M'$ means that $M'$ is obtained from $M$ by converting exactly one subterm.

**Definition.** *$M \to_\beta M'$ is defined inductively by*

$$\begin{aligned} M \to_0 M' &\implies M \to_\beta M', \\ M \to_\beta M' &\implies \lambda x\, M \to_\beta \lambda x\, M', \\ M \to_\beta M' &\implies MN \to_\beta M'N, \\ N \to_\beta N' &\implies MN \to_\beta MN'. \end{aligned}$$

**Definition.** *A term $M$ is $\beta$-normal if $M$ has no $\beta$-convertible subterm.*

Hence a term $M$ is $\beta$-normal if and only if $M$ contains no subterm $M'$ such that $M \to_\beta M'$. We now show that $\to_\beta$ is terminating, i.e. that any reduction sequence starting with $M$ terminates after finitely many steps. By a *reduction sequence* we mean a (finite or infinite) sequence

$$M_1, M_2, \ldots, M_n, \ldots$$

such that $M_{i+1}$ arises from $M_i$ by a $\beta$-conversion of a subterm, i.e. $M_i \to_\beta M_{i+1}$. We write $M \to_\beta^* M'$ (or $M \to_\beta^+ M'$) if $M'$ is a member of a reduction sequence (a reduction sequence with at least two elements) starting with $M$. Hence $\to_\beta^*$ is the reflexive transitive closure of $\to_\beta$.

To prove termination of $\to_\beta$ we make use of a method due to W.W. Tait and define so-called *strong computability predicates*. We present the proof

here in a form which avoids intuitive arguments concerning reduction sequences and therefore is suitable for formalization in a theory of inductive definitions.

We begin with a definition of strongly normalizable terms, by a strictly positive induction.

**Definition.** *If all $M'$ such that $M \to_\beta M'$ are strongly normalizable (sn), then so is $M$.*

Obviously $M$ is sn if and only if any reduction sequence starting with $M$ terminates after a finite number of steps. This can be seen as follows. $\Longrightarrow$. Induction on the definition of 'strongly normalizable'. Consider a reduction sequence starting with $M$ and therein the first reduct $M'$. The IH for $M'$ yields the assertion. $\Longleftarrow$. By induction on the length of the longest reduction sequence (König's Lemma).

We note a number of the properties of the notion 'strongly normalizable', to be used below.

$$\text{If all terms } \vec{M} \text{ are sn, then so is } x\vec{M}. \tag{1}$$

*Proof.* Induction on the definition of sn for $\vec{M}$. Let $x\vec{M} \to_\beta N$ be given. It suffices to show that $N$ is sn. From $x\vec{M} \to_\beta N$ it follows that $N = x\vec{M}'$, where $\vec{M}'$ arises by substitution of $M_i$ by $M_i'$ with $M_i \to_\beta M_i'$. It is to be proved that $x\vec{M}'$ is sn. This follows from the IH for $\vec{M}'$.

$$\text{If } Mx \text{ is sn, then so is } M. \tag{2}$$

*Proof.* Induction on the definition of sn for $Mx$. Let $M \to_\beta M'$ be given. It suffices to show that $M'$ is sn. From $M \to_\beta M'$ we get $Mx \to_\beta M'x$. The IH for $M'x$ then yields that $M'$ is sn.

We now define when a term $M^\rho$ is strongly computable (sc), by induction on the type $\rho$.

**Definition.**

- *$M^\iota$ is sc if $M^\iota$ is sn.*
- *$M^{\rho\to\sigma}$ is sc if for any sc $N^\rho$ also $(MN)^\sigma$ is sc.*

*A term $M$ is called strongly computable under substitution if for any sc terms $\vec{N}$ also $M[\vec{N}/\vec{x}]$ is sc.*

We note a property of the notion 'strongly computable' which will be used below.

$$M \text{ is sc if and only if } M\vec{N} \text{ is sc for all sc } \vec{N}. \tag{3}$$

*Proof.* Induction on the length of $\vec{N}$.

**2.3.** LEMMA.

a. *Any sc term* $M^\rho$ *is sn.*

b. *If* $\vec{M}$ *are sn, then* $(x\vec{M})^\rho$ *is sc.*

*Proof* by simultaneous induction on the type $\rho$. *Case* $\iota$. a. By definition. b. By (1).

*Case* $\rho \to \sigma$ a. Let $M^{\rho\to\sigma}$ be sc. By IHb (with $\vec{M}$ empty) and the definition of strong computability $(Mx)^\sigma$ is sc. By IHa $Mx$ then is sn. By (2) $M$ is sn too. b. Consider $(x\vec{M})^{\rho\to\sigma}$ with $\vec{M}$ sn. Let $N^\rho$ be sc. We have to show that $(x\vec{M}N)^\sigma$ is sc. By IHa $N$ is sn, hence $x\vec{M}N$ is sc by IHb. □

If $M \to_\beta M'$ and $M$ is sn, then by the definition of strong normalizability also $M'$ is sn. We now show that the corresponding assertion is also valid for strong computability.

**2.4.** LEMMA. *If* $M \to_\beta M'$ *and* $M$ *is sc, then so is* $M'$.

*Proof.* We use (3). Let $\vec{N}$ be a list of sc terms such that $M\vec{N}$ is of ground type. Then $M\vec{N}$ is sc by (3), hence also sn. Furthermore we have $M\vec{N} \to_\beta M'\vec{N}$. By definition of strong normalizablility $M'\vec{N}$ is sn. Thus by (3) $M'$ is sc. □

**2.5.** LEMMA. *Let* $N$ *be sn. If* $M_x^N\vec{L}$ *is sn and of a ground type, then so is* $(\lambda x\, M)N\vec{L}$.

*Proof* by induction by the definition of strong normalizability for $N$ and $M_x^N\vec{L}$. So we consider all the reducts $K$ of $(\lambda x\, M)N\vec{L}$. Clearly it suffices to show that any such $K$ is sn.

*Case* $K = M_x^N\vec{L}$, i.e. we have an outer $\beta$-conversion. Hence $K$ is sn by assumption.

*Case* $K = (\lambda x\, M')N\vec{L}$ with $M \to_\beta M'$. Then we have $M_x^N\vec{L} \to_\beta {M'}_x^N\vec{L}$. By definition of strong normalizability ${M'}_x^N\vec{L}$ is sn. Hence by IH $K$ is sn.

*Case* $K = (\lambda x\, M)N'\vec{L}$ with $N \to_\beta N'$. Then we have $M_x^N\vec{L} \to_\beta^* M_x^{N'}\vec{L}$. By definition of strong normalizability $M_x^{N'}\vec{L}$ and $N'$ are sn. Hence by IH $K$ is sn.

*Case* $K = (\lambda x\, M)N\vec{L}'$ with $L_i \to_\beta L_i'$ for $i$ and $L_j = L_j'$ for $j \neq i$. Then we have $M_x^N\vec{L} \to_\beta M_x^N\vec{L}'$. By definition of strong normalizability $M_x^N\vec{L}'$ is sn. Hence by IH $K$ is sn. □

**2.6.** COROLLARY. *If* $M_x^N$ *is sc for all sc* $N$, *then also* $\lambda x\, M$ *is sc.*

*Proof.* Let $M_x^N$ be sc for all sc $N$. We have to show that $\lambda x\, M$ is sc. So let $N$ and $\vec{L}$ be sc such that $(\lambda x\, M)N\vec{L}$ is of ground type. We must show that

$(\lambda x\, M)N\vec{L}$ is sc. Since $M^N_x$ is sc by assumption, $M^N_x\vec{L}$ is sc too and hence also sn. Since by Lemma 2.3a $N$ is also sn, by Lemma 2.5 it follows that $(\lambda x\, M)N\vec{L}$ is sn and hence also sc. □

**2.7.** LEMMA. *Any term $M$ is sc under substitution.*

*Proof* by induction on $M$. *Case* $x$. The claim follows from Lemma 2.3b or by assumption.

*Case* $MN$. $\vec{K}$ be sc. We have to show that $M^{\vec{K}}_{\vec{x}}N^{\vec{K}}_{\vec{x}}$ are sc. This clearly holds, since by IH $M^{\vec{K}}_{\vec{x}}$ as well as $N^{\vec{K}}_{\vec{x}}$ are sc.

*Case* $\lambda x\, M$. $\vec{K}$ be sc. We have to show that $\lambda x\, M^{\vec{K}}_{\vec{x}}$ is sc. We now apply Corollary 2.6. Let $N$ be sc. By IH for $M$ also $M^{N,\vec{K}}_{x,\vec{x}}$ is sc. Hence by Corollary 2.6 the claim follows. □

From Lemma 2.7 and Lemma 2.3 we directly get

**2.8.** THEOREM. *$\to_\beta$ is terminating, i.e. any term $M$ is sn.* □

This proof can easily be extended to terms with pairing $\langle M_0, M_1\rangle$ and projections $\pi_0(M)$, $\pi_1(M)$. It can also be extended to terms that are built with primitive recursion operators (see for example Troelstra [6, 25] or Schwichtenberg [20]), the general recursion operator or the fixed point operator (see Plotkin [17]) or the bounded fixed point operator (see Schwichtenberg/Wainer [21]).

One can also show that the normal form is uniquely determined. A simple proof which uses the technique of parallel reduction (originating from W.W. Tait) has recently been published by Takahashi in [23].

# 3 Logical embeddings

We now show how intuitionistic and classical logic can be embedded into minimal logic, again for simplicity in the implicational fragment of propositional logic; a more complete treatment can be found in [24]. Intuitionistic logic is obtained by adding the ex-falso-quodlibet scheme $\mathrm{F} \to A$, and classical logic by adding the stability scheme $\neg\neg A \to A$, where $\neg A$ abbreviates $A \to \mathrm{F}$. Alternatively one can also obtain classical logic by adding the Peirce schema $(\neg A \to A) \to A$ plus the ex-falso-quodlibet scheme $\mathrm{F} \to A$ to minimal logic. We then show that some 'reasonable' simplification rules for derivations involving the Peirce scheme and the ex-falso-quodlibet scheme or else the stability scheme lead – together with $\beta$-conversion – to non-termination. These counterexamples are due to Hirokawa.

**3.1.** *Embedding classical and intuitionistic logic into minimal logic.* Derivation terms in *intuitionistic* and in *classical* logic are obtained by adding to the first (assumption-) clause of the definition

- in the case of intuitionistic logic: For any propositional symbol $p$ we let

$$\mathcal{A}_p : \mathrm{F} \to p$$

be a derivation term with $\mathrm{FA}(\mathcal{A}_p) = \emptyset$ (Ex-falso-quodlibet axiom).

- in the case of classical logic: For any propositional symbol $p$ we let

$$\mathcal{C}_p : \neg\neg p \to p$$

be a derivation term with $\mathrm{FA}(\mathcal{C}_p) = \emptyset$ (Stability axiom).

Here falsity F is just a special propositional symbol, and $\neg A$ abbreviates $A \to \mathrm{F}$. We write $\Gamma \vdash A$ ($\Gamma \vdash_i A$, $\Gamma \vdash_c A$), if there is a derivation term $d^A$ in minimal (intuitionistic, classical) logic such that for any $u^B \in \mathrm{FA}(d)$ we have $B \in \Gamma$.

By obvious reasons the stability axiom is also called the *principle of indirect proof* for the propositional symbol $p$. We now want to show that from our stability axioms we can derive the principle of indirect proof for arbitrary formulas (in our $\to$-language).

**3.2.** LEMMA. *(Stability Lemma). From stability assumptions $\mathcal{C}_p$ for any propositional symbol $p$ occurring in a formula $A$ we can derive $\neg\neg A \to A$.*

*Proof* by induction on $A$. *Case* $p$. Use $\mathcal{C}_p$.

*Case* F. $\neg\neg\mathrm{F} \to \mathrm{F} = ((\mathrm{F} \to \mathrm{F}) \to \mathrm{F}) \to \mathrm{F}$

$$\frac{(\mathrm{F} \to \mathrm{F}) \to \mathrm{F} \qquad \mathrm{F} \to \mathrm{F}}{\mathrm{F}}$$

*Case* $A \to B$. Use

$$\vdash (\neg\neg B \to B) \to \neg\neg(A \to B) \to A \to B.$$

It can be derived by

$$\dfrac{\dfrac{\neg\neg B \to B \qquad \dfrac{\dfrac{\neg\neg(A \to B) \qquad \dfrac{\dfrac{u\colon \neg B \qquad \dfrac{v\colon A \to B \qquad A}{B}}{\mathrm{F}}\, v}{\neg(A \to B)}}{\mathrm{F}}\, u}{\neg\neg B}}{B}$$

□

Similarly we can show that from our ex-falso-quodlibet axioms we can derive ex-falso-quodlibet for arbitrary formulas (again in our $\to$-language).

**3.3.** LEMMA. *(Ex-falso-quodlibet Lemma). From assumptions $\mathcal{A}_p$ for any propositional symbol $p$ occurring in a formula $A$ we can derive $\mathrm{F} \to A$ in intuitionistic logic.*

*Proof.* By induction on $A$. □

From $\neg\neg A \to A$ one can clearly derive $\mathrm{F} \to A$. Hence any formula derivable in intuitionistic logic is also derivable in classical logic.

Having defined classical and intuitionistic logic, we now want to derive some embedding results.

**Definition.** *A formula $A$ is negative if any propositional symbol in $A$ (different from* F*) occurs negated.*

If one extends this notion to first order logic, then one also has to require that $\vee, \exists$ do not occur in $A$.

**3.4.** LEMMA. *For negative $A$ we have $\vdash \neg\neg A \to A$.*

*Proof.* This follows from the Stability Lemma, using $\vdash \neg\neg\neg p \to \neg p$; here is a derivation of it:

$$\dfrac{\neg\neg\neg p \qquad \dfrac{\dfrac{u\colon \neg p \qquad p}{\mathrm{F}}}{\neg\neg p}\, u}{\mathrm{F}}$$

□

*N.B.* $\not\vdash \neg\neg p \to p$.

**Definition.** *(Gödel-Gentzen negative translation $^g$).*

$$\begin{aligned} p^g &:= \neg\neg p, \\ \mathrm{F}^g &:= \mathrm{F}, \\ (A \to B)^g &:= A^g \to B^g. \end{aligned}$$

**3.5.** THEOREM. *For all $A$,*

a. $\vdash_c A \leftrightarrow A^g$,

b. $\Gamma \vdash_c A$ *iff* $\Gamma^g \vdash A^g$.

*Proof.* a. Clear. b. $\Longleftarrow$. Clear by a. $\Longrightarrow$. By induction on the classical derivation. *Case* $\mathcal{C}_p$. $(\neg\neg p \to p)^g = \neg\neg\neg\neg p \to \neg\neg p$, which is easily derivable. *Case* $\to^+$. Assume

$$\dfrac{\begin{array}{c} u\colon A \\ | \\ B \end{array}}{A \to B}\, \to^+ u$$

Then by IH

$$\begin{array}{c} u\colon A^g \\ | \\ B^g \end{array} \quad \text{and hence} \quad \frac{\begin{array}{c} u\colon A^g \\ | \\ B^g \end{array}}{A^g \to B^g}\,{\to}^+ u$$

*Case* $\to^-$. Assume

$$\frac{\begin{array}{cc} | & | \\ A \to B & A \end{array}}{B}$$

Then by IH

$$\begin{array}{cc} | & | \\ A^g \to B^g & A^g \end{array} \quad \text{and hence} \quad \frac{\begin{array}{cc} | & | \\ A^g \to B^g & A^g \end{array}}{B^g} \qquad \square$$

**3.6.** COROLLARY. *For negative $A$ we have $\vdash_c A$ iff $\vdash A$.*

*Proof.* By the theorem $\vdash_c A$ iff $\vdash A^g$. Since $A$ is negative, any atom $\neq \mathrm{F}$ in $A$ is negated and hence appears in $A^g$ under a triple negation (as $\neg\neg\neg p$), and $\vdash \neg\neg\neg p \leftrightarrow \neg p$. $\square$

*N.B.* $\not\vdash \neg\neg p \to p$, hence the corollary does not hold for all formulas $A$.

**3.7.** COROLLARY. *Let $A^*$ be defined by $p^* := p$ and $(A \to B)^* := \neg B^* \to \neg A^*$, and $A^\circ$ be defined by $p^\circ := p$ and $(A \to B)^\circ := A^\circ \to \neg B^\circ \to \mathrm{F}$. Then*

$$\vdash_c A \iff \vdash A^* \iff \vdash A^\circ.$$

*Proof. Case* $*$. For an atom $p$ both sides are false, and for $A$ composed $A^*$ is negative, and clearly $\vdash_c A \leftrightarrow A^*$. *Case* $\circ$. Clearly $\vdash A^* \leftrightarrow A^\circ$. $\square$

*Remark.* It has been observed by Meyer and Wand [13] that if $M\colon A$ is a simply typed $\lambda$-term, then Fischers cps-translation $\overline{M}$ of $M$ can be typed by $\overline{M}\colon \neg\neg A^\circ$. Griffin observed that this corresponds to a logical embedding. To see this note that by what we just proved $\vdash_c A$ iff $\vdash A^\circ$, hence $\vdash_c A$ iff $\vdash \neg\neg A^\circ$ since $A^\circ$ is negative (if composed) and hence $\vdash \neg\neg A^\circ \leftrightarrow A^\circ$. So we see that from the logical point of view this embedding is something like an overkill, since by what we have proved it introduces more negations than necessary.

We now give the Kuroda interpretation. First note that

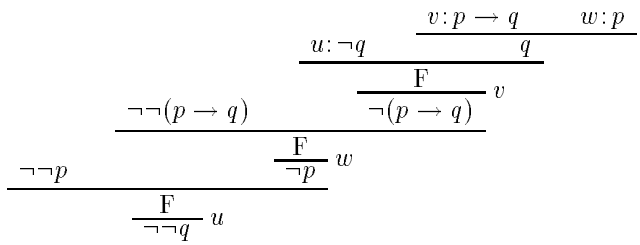
$$\vdash \neg\neg(p \to q) \to \neg\neg p \to \neg\neg q, \tag{4}$$

$$\vdash (\mathrm{F} \to q) \to (\neg\neg p \to \neg\neg q) \to \neg\neg(p \to q), \tag{5}$$

but

$$\not\vdash (\neg\neg p \to \neg\neg q) \to \neg\neg(p \to q).$$

For example, a derivation for (4) is




A derivation for (5) can be obtained similarly. Hence we have

$$\vdash_i \neg\neg(p \to q) \leftrightarrow (\neg\neg p \to \neg\neg q). \tag{6}$$

**3.8.** COROLLARY. *(Kuroda [12]). For all $A$ we have $\vdash_c \neg A$ iff $\vdash_i \neg A$.*

*Proof.* $\Longleftarrow$. Clear. $\Longrightarrow$. Assume $\vdash_c \neg A$. Then $\vdash_c \neg A^g$. Since $A^g$ is negative, by the corollary above $\vdash \neg A^g$. Using (6) we can push the $\neg\neg$ outside (we need $\vdash \neg\neg \mathrm{F} \leftrightarrow \mathrm{F}$ here; cf. the proof of the Stability Lemma) and obtain $\vdash_i \neg\neg\neg A$, hence $\vdash_i \neg A$. □

*N.B.* $\not\vdash \neg\neg(\neg\neg p \to p)$, hence Corollary 3.8 does not hold for $\vdash$ instead of $\vdash_i$.

Finally we discuss the status of a special case of the Peirce scheme, namely

$$\mathcal{P}\colon (\neg A \to A) \to A.$$

The general Peirce scheme $((A \to B) \to A) \to A$ can be obtained from it by replacing falsity F in $\neg A$ by an arbitrary formula $B$. On the basis of minimal logic it is weaker than stability, for we have

$$\vdash (\neg\neg p \to p) \to (\neg p \to p) \to p,$$
$$\not\vdash [(\neg p \to p) \to p] \to \neg\neg p \to p.$$

If, however, we add ex-falso-quodlibet, then the converse holds:

$$\vdash (\mathrm{F} \to p) \to [(\neg p \to p) \to p] \to \neg\neg p \to p.$$

Hence we could also define classical logic by adding to minimal logic the ex-falso-quodlibet axioms as well as the Peirce scheme. In fact, it suffices to add the Peirce axioms $(\neg p \to p) \to p$ for all propositional symbols $p \neq \mathrm{F}$.

**3.9.** *Counterexamples for classical logic.* Counterexamples for classical logic have been given by Sachio Hirokawa; I have learned them form Stefano Berardi (personal communication). To obtain the first example we use our special case of the Peirce formula, i.e. $\mathcal{P}\colon (\neg A \to A) \to A$. For derivations

involving $\mathcal{P}$, the conversion rule ($\mathcal{P}$-rule) mapping

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg A}\,\mathrm{M} \qquad \dfrac{}{(\neg A \to A) \to A}\,\mathcal{P}}{A}}{\mathrm{F}} \qquad \dfrac{}{\neg A \to A}\,\mathrm{N}}$$

into

$$\dfrac{\dfrac{\dfrac{}{\neg A}\,\mathrm{M} \qquad \dfrac{\dfrac{}{\neg A \to A}\,\mathrm{N} \qquad \dfrac{}{\neg A}\,\mathrm{M}}{A}}{\mathrm{F}}$$

is correct. The corresponding derivation term is

$$M(\mathcal{P}N) \to_{\mathcal{P}} M(NM).$$

To formulate the counterexample we introduce some abbreviations. Let $K := \lambda u \lambda v\, u =: \lambda u, v.u$ and $N := \lambda z.zw$ with a free parameter $w$. The counterexample is $K(\mathcal{P}N)(\mathcal{P}N)$, which can be typed by

$$\begin{aligned}
&\mathcal{P}\colon ((\mathrm{F} \to \mathrm{F}) \to \mathrm{F}) \to \mathrm{F},\\
&K\colon \mathrm{F} \to \mathrm{F} \to \mathrm{F},\\
&N\colon (\mathrm{F} \to \mathrm{F}) \to \mathrm{F},\\
&w\colon \mathrm{F}.
\end{aligned}$$

It is a counterexample, for we have

$$\begin{aligned}
& K(\mathcal{P}N)(\mathcal{P}N)\\
\to_{\mathcal{P}}\ & K(\mathcal{P}N)[N\{K(\mathcal{P}N)\}] \quad \text{by the } \mathcal{P}\text{-rule with } M := K(\mathcal{P}N)\\
\to_{\beta}\ & K(\mathcal{P}N)[K(\mathcal{P}N)w]\\
\to_{\beta}\ & K(\mathcal{P}N)(\mathcal{P}N)
\end{aligned}$$

or more explicitely

$$\begin{aligned}
& (\lambda u, v.u)(\mathcal{P}(\lambda z.zw))(\mathcal{P}(\lambda z.zw))\\
\to_{\mathcal{P}}\ & (\lambda u, v.u)(\mathcal{P}(\lambda z.zw))[(\lambda z.zw)\{(\lambda u, v.u)(\mathcal{P}(\lambda z.zw))\}]\\
\to_{\beta}\ & (\lambda u, v.u)(\mathcal{P}(\lambda z.zw))[(\lambda u, v.u)(\mathcal{P}(\lambda z.zw))w]\\
\to_{\beta}\ & (\lambda u, v.u)(\mathcal{P}(\lambda z.zw))[\mathcal{P}(\lambda z.zw)].
\end{aligned}$$

Hence we have reproduced the original term.

From this we can also obtain a counterexample for the language involving stability $\mathcal{C}\colon \neg\neg A \to A$ instead of the Peirce formula $\mathcal{P}$. For derivations involving $\mathcal{C}$ the following conversion rule is correct.

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}\colon \neg\neg A \to A} \qquad \dfrac{}{\neg\neg A}\,\mathrm{M}}{A}}{\mathrm{F}}\,\mathrm{G} \qquad \to_{\mathcal{C}} \qquad \dfrac{\dfrac{}{\neg\neg A}\,\mathrm{M} \qquad \dfrac{\dfrac{}{\mathrm{F}}\,\mathrm{G}}{\neg A}\,u}{\mathrm{F}} \qquad (\mathcal{C}\text{-rule})$$

The corresponding derivation terms are

$$G[\mathcal{C}M] \to_{\mathcal{C}} M(\lambda u\, G[u]).$$

Here $G$ is an 'application context'; in particular, free assumption variables should not be bound in $G$, for otherwise the resulting derivation would contain these assumption variables free.

First note that it is easy to derive $\mathcal{P}$ form $\mathcal{C}$:

$$\dfrac{\mathcal{C}\colon \neg\neg A \to A \qquad \dfrac{\dfrac{\dfrac{}{u\colon \neg A}\qquad \dfrac{\dfrac{}{v\colon \neg A \to A}\qquad \dfrac{}{u\colon \neg A}}{A}}{\mathrm{F}}}{\neg\neg A}\, u}{A}\, v$$
$$(\neg A \to A) \to A$$

The corresponding derivation term is

$$\mathcal{P} := \lambda v.\mathcal{C}\lambda u.u(vu).$$

It turns out that with this definition we can derive the $\mathcal{P}$-rule from the $\mathcal{C}$-rule:

$$\begin{array}{rcl} M(\mathcal{P}N) & = & M([\lambda v.\mathcal{C}\lambda u.u(vu)]N) \\ & \to_\beta & M(\mathcal{C}\lambda u.u(Nu)) \\ & \to_{\mathcal{C}} & (\lambda u.u(Nu))(\lambda x.Mx) \\ & \to_\beta & (\lambda x.Mx)(N(\lambda x.Mx)) \\ & \to_\beta & M(N(\lambda x.Mx)), \end{array}$$

which is the right hand side of the $\mathcal{P}$-rule up to $\eta$-expansion of the final $M$. Hence it is not surprising that we obtain the following counterexample for derivations involving $\mathcal{C}$ instead of $\mathcal{P}$. We use the abbreviations

$$\begin{array}{rcl} \mathcal{P} & := & \lambda v.\mathcal{C}\lambda u.u(vu), \\ K & := & \lambda u\lambda v\, u, \\ N & := & \lambda z.zw \quad \text{with a free parameter } w. \end{array}$$

Then the counterexample again is $K(\mathcal{P}N)(\mathcal{P}N)$, which can be typed as before. It is a counterexample, for we have

$$\begin{array}{rcl} K(\mathcal{P}N)(\mathcal{P}N) & = & K(\mathcal{P}N)([\lambda v.\mathcal{C}\lambda u.u(vu)]N) \\ & \to_\beta & K(\mathcal{P}N)(\mathcal{C}\lambda u.u(Nu)) \\ & = & G[\mathcal{C}M] \quad \text{with } G := K(\mathcal{P}N)[],\ M := \lambda u.u(Nu), \\ & \to_{\mathcal{C}} & M(\lambda x\, G[x]) \\ & = & (\lambda u.u(Nu))(\lambda x.K(\mathcal{P}N)x) \\ & \to_\beta & (\lambda x.K(\mathcal{P}N)x)[N\lambda x.K(\mathcal{P}N)x] \end{array}$$

$$
\begin{aligned}
&\to_\beta \quad K(\mathcal{P}N)[N\lambda x.K(\mathcal{P}N)x] \quad \text{using call-by-name}\\
&\to_\beta \quad K(\mathcal{P}N)[(\lambda x.K(\mathcal{P}N)x)w]\\
&\to_\beta \quad K(\mathcal{P}N)[K(\mathcal{P}N)w]\\
&\to_\beta \quad K(\mathcal{P}N)(\mathcal{P}N).
\end{aligned}
$$

Hence we have seen that a reasonable conversion rule for $\mathcal{C}$ together with $\beta$-conversion (allowing call-by-name) leads to non-termination of the classical implicational calculus.

Note that the $\mathcal{C}$-rule is applied here in a context $M_1(\mathcal{C}M_2)$. This seems to be responsible for non-termination; cf. Gunnar Stålmark [22].

# 4 Global control operators

In order to find computational content in simply typed $\lambda$-terms/proofs involving $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{C}$ we now discuss global control operators in functional languages. We fix the call-by-value strategy for normalizing (or evaluating) $\lambda$-terms. To formulate it we use evaluation contexts of Felleisen et al. [7]. We then explain the call-with-current-continuation operator `call/cc` or $\mathcal{P}$ (well-known from the LISP dialect SCHEME), the abort operator $\mathcal{A}$ and Felleisen's control operator $\mathcal{C}$. For $\mathcal{P}$ we give some programming examples. These operators also make sense in a type-free setting. Following Griffin [10] we then show that the $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$ operators can be typed by the Peirce-, ex-falso-quodlibet- and stability-schemes, respectively. So we have a computational meaning of these schemes, and we may try to use that as a criterium to decide which simplification rules for proofs we should accept.

**4.1.** *Evaluation contexts.* Type-free $\lambda$-terms are defined by

$$M ::= x \mid \lambda x\, M \mid M\, M.$$

Terms of the form $\lambda x\, M$ are called *abstractions*; variables and abstractions are called *values* and will be denoted by $U, V, W$. A term of the form $(\lambda x\, M)V$ with a value $V$ is called a $\beta_v$-redex. A natural call-by-value strategy to evaluate closed terms takes the leftmost $\beta_v$-redex not within an abstractions and converts it. Felleisen et al. [7] have introduced the notion of an *evaluation context* to deal conveniently with this situation. In this subsection we study evaluation contexts in some detail.

First note that any term $M$ is either a value $V$, or it has a uniquely determined leftmost position not under a lambda-abstraction of the form $V_1V_2$. Examples:

$$
\begin{aligned}
&x,\\
&x(\lambda y\, y),
\end{aligned}
$$

$$x(\underline{y}(\underline{\lambda z\, z})),$$
$$x((\underline{\lambda y\, y})\underline{z}),$$
$$x((\underline{\lambda y\, y})\underline{z})(z_1(\lambda y_1\, y_1)),$$
$$x((\lambda y\, y)(\underline{z_1}(\underline{\lambda y_1\, y_1}))).$$

The 'rest' of the term is an *evaluation context* which can be defined inductively by

**Definition.** *(Evaluation context).*

$$E ::= [] \mid VE \mid EM.$$

If $E$ is an evaluation context, then $E[M]$ denotes the result of replacing the 'hole' $[]$ in $E$ by the term $M$.

**4.2.** LEMMA. *Any term $M$ can be written uniquely in the from $V$ or $E[V_1V_2]$.*

*Proof.* Existence. By induction on $M$. It clearly suffices to consider the case $MN$. In case $M = V$ use the IH for $N$. If $N = V_1$ let $E = []$. If $N = E_1[V_1V_2]$ let $E = VE_1$. In case $M = E_1[V_1V_2]$ let $E = E_1N$.

Uniqueness. By induction on $M$. It again suffices to consider the case $MN$. So assume

$$MN = E[V_1V_2] = E'[V_1'V_2'].$$

*Case* $E = []$. If $E' = []$, then $V_1V_2 = V_1'V_2'$ and hence $V_1 = V_1'$ and $V_2 = V_2'$. If $E' = V'E_1'$, then $V_1V_2 = V'E_1'[V_1'V_2']$. But then $V_2 = E_1'[V_1'V_2']$, which is impossible. If $E' = E_1'N$, then $V_1V_2 = E_1'[V_1'V_2']N$. But then $V_1 = E_1'[V_1'V_2']$, which is impossible.

*Case* $E = VE_1$. By symmetry reasons we may assume $E' \neq []$. If $E' = V'E_1'$, then $MN = VE_1[V_1V_2] = V'E_1'[V_1'V_2']$ and hence $V = V'$ and $E_1[V_1V_2] = E_1'[V_1'V_2'] = N$. The IH for $N$ yields $E_1 = E_1'$, $V_1 = V_1'$ and $V_2 = V_2'$. If $E' = E_1'N'$, then $VE_1[V_1V_2] = E_1'[V_1'V_2']N'$ and hence $V = E_1'[V_1'V_2']$, which is impossible.

*Case* $E = E_1N$. By symmetry reasons we may assume $E' \neq []$ and $E' \neq V'E_1'$. So let $E' = E_1'N'$. We obtain $MN = E_1[V_1V_2]N = E_1'[V_1'V_2']N'$. But then $N = N'$, hence $M = E_1[V_1V_2] = E_1'[V_1'V_2']$, and the IH for $M$ yields the claim. □

**4.3.** LEMMA. *$E[E']$ is an evaluation context.*

*Proof* by induction on $E$. *Case* $[]$. Clear. *Case* $VE_1$. Then $(VE_1)[E'] = VE_1[E']$ and the claim follows from the IH. *Case* $E_1M$. Then $(E_1M)[E'] = E_1[E']M$ and the claim again follows from the IH. □

Sabry and Felleisen [18, p. 16] give a different but equivalent definition of evaluation contexts, which will be useful later.

**Definition.** *(Sabry/Felleisen evaluation context).*

$$F ::= [] \mid F[V[]] \mid F[[]M].$$

**4.4.** LEMMA. *$F[F']$ is an evaluation context in the sense of Sabry/Felleisen.*

*Proof* by induction on $F$. *Case* []. Clear. *Case* $F[V[]]$. We must show that $F[V[F']]$ is again an evaluation context (in the sense of Sabry/Felleisen). The IH for $V[]$ yields that $V[F']$ is an evaluation context. The IH for $F$ yields that $F[V[F']]$ is an evaluation context. *Case* $F[[]M]$. We must show that $F[[F']M]$ is again an evaluation. The IH for $[]M$ yields that $[F']M$ is an evaluation context. The IH for $F$ yields that $F[[F']M]$ is an evaluation context. □

**4.5.** LEMMA.

a. *Any $F$ is an $E$.*

b. *Any $E$ is an $F$.*

*Proof.* a. By induction on $F$. *Case* []. Clear. *Case* $F[V[]]$. By IH $F$ is an $E$. We must show that $E[V[]]$ is an $E'$. But this follows from the Lemma 4.3 concerning substitution of $E$'s. *Case* $F[[]M]$. By IH $F$ is an $E$. We must show that $E[[]M]$ is an $E'$. But this again follows from Lemma 4.3.

b. By induction on $E$. *Case* []. Clear. *Case* $VE$. By IH $E$ is an $F$. We must show that $VF$ is an $F'$. But this follows from Lemma 4.4 concerning substitution of $F$'s. *Case* $EM$. By IH $E$ is an $F$. We must show that $FM$ is an $F'$. But this again follows from Lemma 4.4. □

Because of this coincidence we will use $E$ also for evaluation contexts in the sense of Sabry/Felleisen.

**4.6.** LEMMA. *Any term $M$ can be written uniquely in the form $E[V]$ with $E \neq E'[[]N]$ (i.e. the hole [] in $E$ is not the left hand side of an application).*

*Proof.* Clearly we may assume that $M$ is not a value.

Existence. By Lemma 4.2 $M$ is of the form $M = E'[V_1V_2]$. Take $E := E'[V_1[]]$.

Uniqueness. Assume $M = E[V] = E'[V']$. Again $E = E_1[V_1[]]$ and $E' = E_1'[V_1'[]]$, hence $E[V] = E_1[V_1V]$ and $E'[V'] = E_1'[V_1'V']$. So we have

$$E_1[V_1V] = E_1'[V_1'V']$$

and therefore by Lemma 4.2 $E_1 = E_1'$, $V_1 = V_1'$ and $V = V'$. But then $E = E'$ follows. □

Later we will also need

**4.7.** LEMMA. *Any evaluation context $E$ can be written uniquely in one of the forms*

$$[], \quad E[x[]], \quad E[(\lambda x\, M)[]], \quad E[[]M].$$

*Proof.* Existence. This follows immediately from Sabry/Felleisen's characterization of evaluation contexts.

Uniqueness. This is clear, since there is a unique hole [] in $E$, and [] (in case $E \neq []$) is either the right hand side or else the left hand side of an application. □

Since by Lemma 4.6 any *closed* non-normal term can be written uniquely in the form $E[(\lambda x\, M)V]$, we can express call-by-value $\beta$-conversion conveniently as

$$E[(\lambda x\, M)V] \mapsto E[M_x^V].$$

If $M_x^V$ is not a value, then any further computation step can only take place within $M_x^V$. Hence, in a precise sense, $E$ represents the rest of the computation (or continuation) still to be done after the evaluation of $(\lambda x\, M)V$ is complete. So the notion of an evaluation context makes it possible to formulate precisely the effect of program constructs that deal with global control, such as `call/cc` in SCHEME. To these we turn next.

**4.8.** *Manipulating continuations.* In the programming language SCHEME the `call/cc` construct (`call/cc` *proc*) is informally described as follows (in the Revised$^4$ Report [4]). "The procedure `call/cc` packages up the current continuation as an 'escape procedure' and passes it as an argument to *proc*. The escape procedure is a SCHEME procedure of one argument that, if later passed a value, will ignore whatever continuation is in effect at that later time and will give the value instead to the continuation that was in effect when the escape procedure was created".

We now give some simple example programs involving `call/cc`.

```
(define (occurs? var term)
  (call/cc
   (lambda (return) ; return is the continuation; calling
     ; return with one argument has the effect that the result
     ; will be returned immediately as the result of the
     ; call/cc expression
     (let occurs-help ((term term))
       (cond ((variable? term)
              (if (eq? var term)
                  (return #t)))
                    ; if found, return #t immediately
             ((fct-app-form? term)
              (for-each occurs-help
```

```
                    (fct-app-form-to-args term)))
            ((app-form? term)
             (occurs-help (app-form-to-op term))
             (occurs-help (app-form-to-arg term)))
            ((lambda-form? term)
             (if
              (not (eq? var (lambda-form-to-symbol term)))
              (occurs-help (lambda-form-to-kernel term)))))
      #f))))

(define (unify term1 term2)
  (call/cc
   (lambda (return)
     (do ((x (list empty-subst term1 term2)
             (let* ((subst (car x))
                    (t1 (cadr x))
                    (t2 (caddr x))
                    (p (disagreement-pair t1 t2)))
               (if (not p)
                   (return subst)
                   (let ((l (car p)) (r (cadr p)))
                     (cond
                      ((and (variable? l)
                            (not (occurs? l r)))
                       (list (extend subst l r)
                             (term-substitute
                               t1 (list p))
                             (term-substitute
                               t2 (list p))))
                      ((and (variable? r)
                            (not (occurs? r l)))
                       (list (extend subst r l)
                             (term-substitute
                               t1 (list (reverse p)))
                             (term-substitute
                               t2 (list (reverse p)))))
                      (else (return 'no))))))))
         (#f)))))
```

The procedure `prod` expects a list argument and forms the product of the elements of the list. If one of the elements is 0, it returns 0 immediately.

```
(define (prod l)
  (call/cc
   (lambda (return)
     (let aux ((x l))
```

```
      (cond ((null? x) 1)
            ((zero? (car x)) (return 0))
            (else (* (car x) (aux (cdr x))))))))))
```

A continuation-passing-style version of `prod` is

```
(define (prod-cps l)
  (let aux ((x l)
            (k (lambda (y) y)))
    (cond ((null? x) (k 1))
          ((zero? (car x)) 0)
          (else (aux (cdr x)
                     (lambda (y) (* (car x) (k y))))))))))
```

We will also deal with an abort operator (e.g. `error` in most implementations of SCHEME), which throws away the current continuation and immediately returns its argument. Finally, we will also discuss a control operator introduced by Felleisen. Its effect is very similar to that of `call/cc`. However, it also throws away the continuation in effect when it is called, and returns immediately.

To deal with these constructs in our present setting, we extend our notion of a term by adding the global control operators $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$. We view these operators not as separate values but rather as term constructors. Hence we extend our definition of a term as follows.

$$M ::= x \mid \lambda x\, M \mid M\, M \mid \mathcal{P}M \mid \mathcal{C}M \mid \mathcal{A}M.$$

The notion of a value remains unchanged

$$V ::= x \mid \lambda x\, M,$$

and also the definition of evaluation contexts stays the same.

We now have to extend our lemmata concerning unique representations of terms to the extended language. To Lemma 4.2 there corresponds

**4.9.** LEMMA. *Any term $M$ can be written uniquely in one of the forms $V$, $E[V_1V_2]$, $E[\mathcal{P}M]$, $E[\mathcal{C}M]$ or $E[\mathcal{A}M]$.*

*Proof.* Existence. By induction on $M$. It clearly suffices to consider the case $MN$. In case $M = V$ use the IH for $N$. If $N = V_1$ let $E = []$. If $N$ has one of the forms $E_1[V_1V_2]$, $E_1[\mathcal{P}M]$, $E_1[\mathcal{C}M]$ or $E_1[\mathcal{A}M]$ let $E = VE_1$. In case $M$ has one of the forms $E_1[V_1V_2]$, $E_1[\mathcal{P}M]$, $E_1[\mathcal{C}M]$ or $E_1[\mathcal{A}M]$ let $E = E_1N$.

Uniqueness. By induction on $M$. It again suffices to consider the case $MN$. The proof proceeds just as for Lemma 4.2; one only has to consider some more cases. □

To Lemma 4.6 there corresponds

**4.10.** LEMMA. *Any term $M$ can be written uniquely in one of the forms $E[V]$, $E[\mathcal{P}M]$, $E[\mathcal{C}M]$ or $E[\mathcal{A}M]$ with $E \neq E'[]N]$ (i.e. the hole [] in $E$ is not the left hand side of an application).*

*Proof.* Just as for Lemma 4.6; one again only has to consider some more cases. □

The conversion rules for $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$ can now be formulated:

$$E[\mathcal{P}M] \to_{\mathcal{P}} E[M\lambda z.\mathcal{A}E[z]],$$
$$E[\mathcal{C}M] \to_{\mathcal{C}} M\lambda z.\mathcal{A}E[z],$$
$$E[\mathcal{A}M] \to_{\mathcal{A}} M.$$

Clearly these conversion rules express what we informally have said before.

**4.11.** *Typing the conversion rules for control operators.* The conversion rules for $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$ are

$$E[\mathcal{P}M] \to_{\mathcal{P}} E[M\lambda z.\mathcal{A}E[z]],$$
$$E[\mathcal{C}M] \to_{\mathcal{C}} M\lambda z.\mathcal{A}E[z],$$
$$E[\mathcal{A}M] \to_{\mathcal{A}} M.$$

We now want to type these conversion rules. It can be seen easily that the only possible way to do this is as follows. For the $\mathcal{P}$-rule

$$\dfrac{\mathcal{P}\colon (\neg A \to A) \to A \qquad \begin{array}{c}\mid \mathrm{M}\\ \neg A \to A\end{array}}{\begin{array}{c}A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}}$$

is transformed into

$$\dfrac{\neg A \to A \qquad \begin{array}{c}\mid \mathrm{M}\\ \dfrac{\mathcal{A}\colon \mathrm{F} \to \mathrm{F} \qquad \begin{array}{c}z\colon A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}}{\dfrac{\mathrm{F}}{\neg A}\, z}\end{array}}{\begin{array}{c}A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}}$$

and for the $\mathcal{C}$-rule we have

$$\dfrac{\mathcal{C}\colon \neg\neg A \to A \qquad \begin{array}{c}\mid \mathrm{M}\\ \neg\neg A\end{array}}{\begin{array}{c}A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}} \quad \to_{\mathcal{C}} \quad \dfrac{\begin{array}{c}\mid \mathrm{M}\\ \neg\neg A\end{array} \qquad \dfrac{\mathcal{A}\colon \mathrm{F} \to \mathrm{F} \qquad \begin{array}{c}z\colon A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}}{\dfrac{\mathrm{F}}{\neg A}\, z}}{\mathrm{F}}$$

Note that the appearance of the $\mathcal{A}$-axioms $\mathrm{F} \to \mathrm{F}$ in these derivations clearly is unneccessary from the logical point of view. However, as will be apparent soon, they are essential from the operational point of view for the derivation terms.

$$\dfrac{\mathcal{A}\colon \mathrm{F} \to A \qquad \begin{array}{c}\mid \mathrm{M}\\ \mathrm{F}\end{array}}{\begin{array}{c}A\\ \mid \mathrm{E}\\ \mathrm{F}\end{array}} \quad \to_{\mathcal{A}} \quad \begin{array}{c}\mid \mathrm{M}\\ \mathrm{F}\end{array} \qquad (\mathcal{A}\text{-rule})$$

At this point the connection of the global control operators to classical logic becomes apparent. Starting from the conversion rules which have been motivated solely by operational considerations we arrive necessarily at the typing

$$\begin{aligned}&\mathcal{P}\colon (\neg A \to A) \to A,\\ &\mathcal{C}\colon \neg\neg A \to A,\\ &\mathcal{A}\colon \mathrm{F} \to A\end{aligned}$$

for $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$, which make them correspond to the Peirce, stability and ex-falso-quodlibet schemes, respectively.

However, there is one problem with this typing. Assume that we have a closed term possibly involving $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$ to which we wish to apply the conversion rules above. As we just have seen, this is only possible if the whole term has type F. But there is no closed term of type F, for this would mean that $\vdash_c \mathrm{F}$.

To solve this problem, recall that in the presence of $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$ we essentially are in the realm of classical logic. Hence instead of deriving $A$ we may as well derive $\neg\neg A$, i.e. F from an additional assumption $\neg A$.

A slightly different way to formulate this has been used by Griffin. He defines the conversion rules by

$$\begin{aligned}&\mathcal{C}(\lambda k\, E[(\lambda x\, M)V]) \to_{\beta v} \mathcal{C}(\lambda k\, E[M^V_x]),\\ &\mathcal{C}(\lambda k\, E[\mathcal{P}M]) \to_{\beta \mathcal{P}} \mathcal{C}(\lambda k\, E[M\lambda z.\mathcal{A}E[z]]),\\ &\mathcal{C}(\lambda k\, E[\mathcal{C}M]) \to_{\beta \mathcal{C}} \mathcal{C}(\lambda k\, M\lambda z.\mathcal{A}E[z]),\\ &\mathcal{C}(\lambda k\, E[\mathcal{A}M]) \to_{\beta \mathcal{A}} \mathcal{C}(\lambda k\, M).\end{aligned}$$

# 5 Termination of evaluation

Based on Griffin's work [10] we now show that the simply typed $\lambda$-calculus extended by the $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$ operators enjoys termination with respect to the call-by-value strategy. The proof involves a cps-translation into the simply typed $\lambda$-calculus and uses strong normalization for the latter. We first introduce a cps-translation for the simply typed $\lambda$-calculus and then extend it to the language involving $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$.

**5.1.** *A cps-translation of the simply typed lambda calculus.* We define simultaneously a 'term transformation' $\mathcal{T}_W(M)$ and an 'evaluation context transformation' $\mathcal{K}_W(E)$. $\mathcal{T}_W(M)$ is closely related to Plotkin's [16] and Griffin's [10] 'colon translation' $M\colon W$, and $\mathcal{K}_W(E)$ is closely related to their $W^E$. However, our definition – which is based on Sabry/Felleisen [18] – is more 'compacting' in the sense that it avoids 'administrative redexes'.

**Definition.** *a. For any value $W\colon \neg A^*$ and term $M\colon A$ we define a term $\mathcal{T}_W(M)\colon \mathrm{F}$ by*

$$\begin{array}{rcl}
\mathcal{T}_W(V) & := & W\Phi(V), \\
\mathcal{T}_W(E[xV]) & := & x^*\mathcal{K}_W(E)\Phi(V), \ \textit{where } x\colon B \to C,\ x^*\colon \neg C^* \to \neg B^*, \\
\mathcal{T}_W(E[(\lambda x\, M)V]) & := & (\lambda x^*\mathcal{T}_W(E[M]))\Phi(V), \quad \textit{where } x\colon A,\ M\colon B,\ x^*\colon A^*.
\end{array}$$

*b. For any value $V\colon A$ we define a value $\Phi(V)\colon A^*$ by*

$$\begin{array}{rcl}
\Phi(x) & := & x^*, \\
\Phi(\lambda x\, M) & := & \lambda \ell, x^*.\mathcal{T}_\ell(M),
\end{array}$$

*where $x\colon A$, $M\colon B$, and $\ell\colon \neg B^*$ is a newly created variable.*
*c. For any value $W\colon \neg A^*$ and evaluation context $E[\cdot^B]\colon A$ we define a value $\mathcal{K}_W(E)\colon \neg B^*$ by*

$$\begin{array}{rcl}
\mathcal{K}_W([]) & := & W, \\
\mathcal{K}_W(E[x[]]) & := & x^*\mathcal{K}_W(E), \quad \textit{where } x\colon B \to C,\ x^*\colon \neg C^* \to \neg B^*, \\
\mathcal{K}_W(E[(\lambda x\, M)[]]) & := & \lambda x^*\mathcal{T}_W(E[M]), \quad \textit{where } x\colon B,\ x^*\colon B^*, \\
\mathcal{K}_W(E[[]M]) & := & \lambda x^*\mathcal{T}_W(E[xM]).
\end{array}$$

*where in the last clause $[]\colon B \to C$, $x\colon B \to C$, $x^*\colon \neg C^* \to \neg B^*$ and $x$ is a newly created variable.*

We now show that this is a good definition. First note that $\Phi$ can be eliminated by replacing each $\mathcal{T}()$-clause by two for the two possible forms of $V$ and inserting there the definition of $\Phi$. We define a measure $|M|$ for terms $M$ and $|E|$ for evaluation contexts $E$ such that in each clause of the definition the measure decreases.

**Definition.** *(Measure).*

$$\begin{array}{rcl}
|M| & := & 2 \cdot \mathrm{vars}(M), \\
|E| & := & 2 \cdot \mathrm{vars}(E) + 3,
\end{array}$$

*where* $\mathrm{vars}(\cdot)$ *is the total number of variable occurrences in the term or evaluation context $\cdot$, including the bound occurrences.*

Now the well-definedness of $\mathcal{T}()$ and $\mathcal{K}()$ follows from

$$|E[xV]| \ > \ |E|,$$

$$
\begin{aligned}
|E[(\lambda x\, M)V]| &> |E[M]|,\\
|E[x\,[\,]]| &> |E|,\\
|E[(\lambda x\, M)[\,]]| &> |E[M]|,\\
|E[[\,]M]| &> |E[xM]|.
\end{aligned}
$$

**5.2.** LEMMA. *(Context unwrapping).*

a. $\mathcal{T}_{\mathcal{K}_W(E)}(M) \to_\beta^* \mathcal{T}_W(E[M])$, *where* $E[\cdot^B]\colon A$, $W\colon \neg A^*$, $M\colon B$,

b. $\mathcal{K}_{\mathcal{K}_W(E)}(E') \to_\beta^* \mathcal{K}_W(E[E'])$, *where* $E[\cdot^B]\colon A$, $W\colon \neg A^*$, $E'[\cdot^C]\colon B$.

*Proof.* To be able to give a proof by induction we have to prove simultaneously with parts a and b the following special cases of the substitution lemma (to be proved in general form as our next lemma):

c. $\mathcal{T}_W(M)^{\Phi(V)}_{x^*} \to_\beta^* \mathcal{T}_W(M^V_x)$ with $M = E[xN]$, where $x$ is not free in $E$, $N$ and $W$.

d. $\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \to_\beta^* \mathcal{K}_W(E^V_x)$ with $E = E_1[xE_2]$, where $x$ is not free in $E_1$, $E_2$ and $W$.

We prove parts a-d simultaneously by induction on the following measure.

- For part a: $|E| + |M|$.
- For part b: $|E| + |E'|$.
- For part c: $|M| + |V|$.
- For part d: $|E| + |V|$.

a. We first prove part a for values $V$, by cases on $E$. If $E$ does not have the hole $[\,]$ on the left hand side of an application, then

$$
\begin{aligned}
\mathcal{T}_{\mathcal{K}_W(E)}(V) &= \mathcal{K}_W(E)\Phi(V)\\
&= \mathcal{T}_W(E[V]).
\end{aligned}
$$

The last equation follows from the fact that the clauses in the definition of $\mathcal{T}_W(V)$ and $\mathcal{K}_W(E)$ look very similar. It can be proved easily by considering the cases $[\,]$, $E[x[\,]]$ and $E[(\lambda x\, M)[\,]]$ for $E$.

So it remains to consider the case $E[[\,]M]$.

$$
\begin{aligned}
\mathcal{T}_{\mathcal{K}_W(E[[\,]M])}(V) &= \mathcal{K}_W(E[[\,]M])\Phi(V)\\
&= (\lambda x^*\mathcal{T}_W(E[xM]))\Phi(V)\\
&\to_\beta \mathcal{T}_W(E[xM])^{\Phi(V)}_{x^*}\\
&\to_\beta^* \mathcal{T}_W(E[VM]) \quad \text{by IHc for } E[xM],\, V.
\end{aligned}
$$

Note here that the IHc is applicable since $|E[xM]|+|V| < |E[[]M]|+|V|$. – We now prove part a for the case where $M$ is not a value. *Case* $E'[xV]$.

$$\begin{array}{rcl}\mathcal{T}_{\mathcal{K}_W(E)}(E'[xV]) & = & x^*\mathcal{K}_{\mathcal{K}_W(E)}(E')\Phi(V) \\ & \to_\beta^* & x^*\mathcal{K}_W(E[E'])\Phi(V) \quad \text{by IHb for } E,\, E' \\ & = & \mathcal{T}_W(E[E'[xV]]).\end{array}$$

Note that the IHb is applicable since $|E| + |E'| < |E| + |E'[xV]|$. *Case* $E'[(\lambda x\, M)V]$.

$$\begin{array}{rl} & \mathcal{T}_{\mathcal{K}_W(E)}(E'[(\lambda x\, M)V]) \\ = & (\lambda x^*\mathcal{T}_{\mathcal{K}_W(E)}(E'[M]))\Phi(V) \\ \to_\beta^* & (\lambda x^*\mathcal{T}_W(E[E'[M]]))\Phi(V) \quad \text{by IHa for } E,\, E'[M] \\ = & \mathcal{T}_W(E[E'[(\lambda x\, M)V]]).\end{array}$$

Note that the IHa is applicable since $|E| + |E'[M]| < |E| + |E'[(\lambda x\, M)V]|$.

b. In case $E' = []$ we have $\mathcal{K}_{\mathcal{K}_W(E)}([]) = \mathcal{K}_W(E)$ by definition. *Case* $E'[x[]]$.

$$\begin{array}{rcl}\mathcal{K}_{\mathcal{K}_W(E)}(E'[x[]]) & = & x^*\mathcal{K}_{\mathcal{K}_W(E)}(E') \\ & \to_\beta^* & x^*\mathcal{K}_W(E[E']) \quad \text{by IHb for } E,\, E' \\ & = & \mathcal{K}_W(E[E'[x[]]]).\end{array}$$

Note that the IHb is applicable since $|E| + |E'| < |E| + |E'[x[]]|$. *Case* $E'[(\lambda x\, M)[]]$.

$$\begin{array}{rcl}\mathcal{K}_{\mathcal{K}_W(E)}(E'[(\lambda x\, M)[]]) & = & \lambda x^*\mathcal{T}_{\mathcal{K}_W(E)}(E'[M]) \\ & \to_\beta^* & \lambda x^*\mathcal{T}_W(E[E'[M]]) \quad \text{by IHa for } E,\, E'[M] \\ & = & \mathcal{K}_W(E[E'[(\lambda x\, M)[]]]).\end{array}$$

Note that the IHa is applicable since $|E| + |E'[M]| < |E| + |E'[(\lambda x\, M)[]]|$. *Case* $E'[[]M]$.

$$\begin{array}{rcl}\mathcal{K}_{\mathcal{K}_W(E)}(E'[[]M]) & = & \lambda x^*\mathcal{T}_{\mathcal{K}_W(E)}(E'[xM]) \\ & \to_\beta^* & \lambda x^*\mathcal{T}_W(E[E'[xM]]) \quad \text{by IHa for } E,\, E'[xM] \\ & = & \mathcal{K}_W(E[E'[[]M]]).\end{array}$$

Note that the IHa is applicable since $|E| + |E'[xM]| < |E| + |E'[[]M]|$.

c. First note that we can only have the following cases for $M$: $E[xU]$, $E[xE'[yU]]$ and $E[xE'[(\lambda y\, M)U]]$. *Case* $E[xU]$.

$$\begin{array}{rcl}\mathcal{T}_W(E[xU])_{x^*}^{\Phi(V)} & = & (x^*\mathcal{K}_W(E)\Phi(U)_{x^*}^{\Phi(V)} \\ & = & \Phi(V)\mathcal{K}_W(E)\Phi(U).\end{array}$$

In case $V = y$ this is

$$\begin{array}{rl} = & y^*\mathcal{K}_W(E)\Phi(U) \\ = & \mathcal{T}_W(E[yU]), \end{array}$$

and in case $V = \lambda y\, N$ this is

$$\begin{array}{rl} = & (\lambda \ell, y^*.\mathcal{T}_\ell(N))\mathcal{K}_W(E)\Phi(U) \\ \to_\beta & (\lambda y^*\mathcal{T}_{\mathcal{K}_W(E)}(N))\Phi(U) \\ \to_\beta^* & (\lambda y^*\mathcal{T}_W(E[N]))\Phi(U) \quad \text{by IHa for } E,\, N \\ = & \mathcal{T}_W(E[(\lambda y\, N)U]). \end{array}$$

Note that the IHa is applicable since $|E| + |N| < |E[xy]| + |\lambda z\, N|$. *Case* $E[xE'[yU]]$.

$$\begin{array}{rll} \mathcal{T}_W(E[xE'[yU]])_{x^*}^{\Phi(V)} & = & (y^*\mathcal{K}_W(E[xE'])\Phi(U))_{x^*}^{\Phi(V)} \\ & = & y^*\mathcal{K}_W(E[xE'])_{x^*}^{\Phi(V)}\Phi(U) \\ & \to_\beta^* & y^*\mathcal{T}_W(E[VE'])\Phi(U) \quad \text{by IHd for } E[xE'],\, V \\ & = & \mathcal{T}_W(E[VE'[yU]]). \end{array}$$

Note that the IHd is applicable since $|E[xE']| + |V| < |E[yU]| + |V|$. *Case* $E[xE'[(\lambda y\, M)U]]$.

$$\begin{array}{rll} & \mathcal{T}_W(E[xE'[(\lambda y\, M)U]])_{x^*}^{\Phi(V)} & \\ = & ((\lambda y^*\mathcal{T}_W(E[xE'[M]]))\Phi(U))_{x^*}^{\Phi(V)} & \\ = & (\lambda y^*\mathcal{T}_W(E[xE'[M]])_{x^*}^{\Phi(V)})\Phi(U) & \\ \to_\beta^* & (\lambda y^*\mathcal{T}_W(E[VE'[M]]))\Phi(U) & \text{by IHc for } E[xE'[M]],\, V \\ = & \mathcal{T}_W(E[VE'[(\lambda y\, M)U]]). & \end{array}$$

Note that the IHc is applicable since $|E[xE'[M]]| + |V| < |E[(\lambda y\, M)U]| + |V|$.

d. First note that we can only have the following cases for $E$: $E[x[]]$, $E[xE'[y[]]]$, $E[xE'[(\lambda y\, M)[]]]$ and $E[xE'[[]M]]$. *Case* $E[x[]]$.

$$\begin{array}{rll} \mathcal{K}_W(E[x[]])_{x^*}^{\Phi(V)} & = & (x^*\mathcal{K}_W(E))_{x^*}^{\Phi(V)} \\ & = & \Phi(V)\mathcal{K}_W(E). \end{array}$$

In case $V = y$ this is

$$\begin{array}{rl} = & y^*\mathcal{K}_W(E) \\ = & \mathcal{K}_W(E[y[]]), \end{array}$$

and in case $V = \lambda y\, N$ this is

$$\begin{array}{rl} = & (\lambda \ell, y^*.\mathcal{T}_\ell(N))\mathcal{K}_W(E) \\ \to_\beta & \lambda y^*\mathcal{T}_{\mathcal{K}_W(E)}(N) \\ \to_\beta^* & \lambda y^*\mathcal{T}_W(E[N]) \quad \text{by IHa for } E,\, N \\ = & \mathcal{K}_W(E[(\lambda y\, N)[]]). \end{array}$$

Note that the IHa is applicable since $|E| + |N| < |E[x[]]| + |V|$. *Case* $E[xE'[y[]]]$.

$$\begin{array}{rcl}
\mathcal{K}_W(E[xE'[y[]]])^{\Phi(V)}_{x^*} & = & (y^*\mathcal{K}_W(E[xE']))^{\Phi(V)}_{x^*} \\
& = & y^*\mathcal{K}_W(E[xE'])^{\Phi(V)}_{x^*} \\
& \to^*_\beta & y^*\mathcal{K}_W(E[VE']) \quad \text{by IHd for } E[xE'], V \\
& = & \mathcal{K}_W(E[VE'[y[]]]).
\end{array}$$

Note that the IHd is applicable since $|E[xE']| + |V| < |E[xE'[y[]]]| + |V|$.

*Case* $E[xE'[(\lambda y\, M)[]]]$.

$$\begin{array}{rcl}
 & & \mathcal{K}_W(E[xE'[(\lambda y\, M)[]]])^{\Phi(V)}_{x^*} \\
& = & (\lambda y^*\mathcal{T}_W(E[xE'[M]]))^{\Phi(V)}_{x^*} \\
& = & \lambda y^*\mathcal{T}_W(E[xE'[M]])^{\Phi(V)}_{x^*} \\
& \to^*_\beta & \lambda y^*\mathcal{T}_W(E[VE'[M]]) \quad \text{by IHc for } E[xE'[M]], V \\
& = & \mathcal{K}_W(E[VE'[(\lambda y\, M)[]]]).
\end{array}$$

Note that the IHc is applicable since $|E[xE'[M]]| + |V| < |E[xE'[(\lambda y\, M)[]]]| + |V|$. *Case* $E[xE'[[]M]]$.

$$\begin{array}{rcl}
 & & \mathcal{K}_W(E[xE'[[]M]])^{\Phi(V)}_{x^*} \\
& = & (\lambda y^*\mathcal{T}_W(E[xE'[yM]]))^{\Phi(V)}_{x^*} \\
& \to^*_\beta & \lambda y^*\mathcal{T}_W(E[VE'[yM]]) \quad \text{by IHc for } E[xE'[yM]], V \\
& = & \mathcal{K}_W(E[VE'[[]M]]).
\end{array}$$

Note that the IHc is applicable since $|E[yM_x]| + |V| < |E[[]M_x]| + |V|$. □

**5.3.** LEMMA. *(Substitution). Let $x$ be not free in $V$. Then*

a. $\mathcal{T}_W(M)^{\Phi(V)}_{x^*} \to^*_\beta \mathcal{T}_{W^{\Phi(V)}_{x^*}}(M^V_x)$.

b. $\Phi(W)^{\Phi(V)}_{x^*} \to^*_\beta \Phi(W^V_x)$.

c. $\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \to^*_\beta \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)$.

*Proof.* We prove parts a-c simultaneously by induction on the following measure, using the previous lemma on context unwrapping.

- For part a: $|M|^+$.
- For part b: $|W|^+$.

- For part c: $|E|^+$.

Here $|M|^+$ and $|E|^+$ are defined similarly to $|M|$, $|E|$ exept that binding occurrences of variables now count.

a. First note that we can have the following cases for $M$: $y$, $x$, $\lambda y\, M$, $E[yU]$, $E[xU]$ and $E[(\lambda y\, M)U]$. *Case* $y$.

$$\begin{aligned}\mathcal{T}_W(y)^{\Phi(V)}_{x^*} &= (Wy^*)^{\Phi(V)}_{x^*}\\ &= W^{\Phi(V)}_{x^*}y^*\\ &= \mathcal{T}_{W^{\Phi(V)}_{x^*}}(y).\end{aligned}$$

*Case* $x$.

$$\begin{aligned}\mathcal{T}_W(x)^{\Phi(V)}_{x^*} &= (Wx^*)^{\Phi(V)}_{x^*}\\ &= W^{\Phi(V)}_{x^*}\Phi(V)\\ &= \mathcal{T}_{W^{\Phi(V)}_{x^*}}(V).\end{aligned}$$

*Case* $\lambda y\, M$.

$$\begin{aligned}\mathcal{T}_W(\lambda y\, M)^{\Phi(V)}_{x^*} &= (W(\lambda\ell, y^*.\mathcal{T}_\ell(M)))^{\Phi(V)}_{x^*}\\ &= W^{\Phi(V)}_{x^*}(\lambda\ell, y^*.\mathcal{T}_\ell(M)^{\Phi(V)}_{x^*})\\ &\to^*_\beta W^{\Phi(V)}_{x^*}(\lambda\ell, y^*.\mathcal{T}_\ell(M^V_x)) \quad \text{by IHa for } M\\ &= \mathcal{T}_{W^{\Phi(V)}_{x^*}}(\lambda y\, M^V_x).\end{aligned}$$

Note that the IHa is applicable since by the definition of $|\cdot|^+$ we have $|M|^+ < |\lambda y\, M|^+$. *Case* $E[yU]$.

$$\begin{aligned}\mathcal{T}_W(E[yU])^{\Phi(V)}_{x^*} &= (y^*\mathcal{K}_W(E)\Phi(U))^{\Phi(V)}_{x^*}\\ &= y^*\mathcal{K}_W(E)^{\Phi(V)}_{x^*}\Phi(U)^{\Phi(V)}_{x^*}\\ &\to^*_\beta y^*\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)\Phi(U)^{\Phi(V)}_{x^*} \quad \text{by IHc for } E\\ &\to^*_\beta y^*\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)\Phi(U^V_x) \quad \text{by IHb for } U\\ &= \mathcal{T}_{W^{\Phi(V)}_{x^*}}(E[yU]^V_x).\end{aligned}$$

Note that the IHc is applicable since $|E|^+ < |E[yU]|^+$, and the IHb is applicable since $|U|^+ < |E[yU]|^+$. *Case* $E[xU]$.

$$\begin{aligned}&\mathcal{T}_W(E[xU])^{\Phi(V)}_{x^*}\\ = \;&(x^*\mathcal{K}_W(E)\Phi(U))^{\Phi(V)}_{x^*}\\ = \;&\Phi(V)\mathcal{K}_W(E)^{\Phi(V)}_{x^*}\Phi(U)^{\Phi(V)}_{x^*}\end{aligned}$$

$$
\begin{array}{lll}
\rightarrow_\beta^* & \Phi(V)\mathcal{K}_{W_{x^*}^{\Phi(V)}}(E_x^V)\Phi(U)_{x^*}^{\Phi(V)} & \text{by IHc for } E \\
\rightarrow_\beta^* & \Phi(V)\mathcal{K}_{W_{x^*}^{\Phi(V)}}(E_x^V)\Phi(U_x^V) & \text{by IHb for } U.
\end{array}
$$

In case $V = y$ this is

$$
\begin{array}{ll}
= & y^*\mathcal{K}_{W_{x^*}^{\Phi(V)}}(E_x^V)\Phi(U_x^V) \\
= & \mathcal{T}_{W_{x^*}^{\Phi(V)}}(E[xU]_x^V),
\end{array}
$$

and in case $V = \lambda y\, N$ this is

$$
\begin{array}{lll}
= & (\lambda y^*.\mathcal{T}_{\mathcal{K}_{W_{x^*}^{\Phi(V)}}(E_x^V)}(N))\Phi(U_x^V) & \\
\rightarrow_\beta^* & (\lambda y^*.\mathcal{T}_{W_{x^*}^{\Phi(V)}}(E_x^V[N]))\Phi(U_x^V) & \text{by context unwrapping} \\
= & \mathcal{T}_{W_{x^*}^{\Phi(V)}}(E[xU]_x^V). &
\end{array}
$$

Note that the IHc is applicable since $|E|^+ < |E[xU]|^+$, and the IHb is applicable since $|U|^+ < |E[xU]|^+$. *Case* $E[(\lambda y\, M)U]$.

$$
\begin{array}{lll}
 & \mathcal{T}_W(E[(\lambda y\, M)U])_{x^*}^{\Phi(V)} & \\
= & ((\lambda y^*\mathcal{T}_W(E[M]))\Phi(U))_{x^*}^{\Phi(V)} & \\
= & (\lambda y^*\mathcal{T}_W(E[M])_{x^*}^{\Phi(V)})\Phi(U)_{x^*}^{\Phi(V)} & \\
\rightarrow_\beta^* & (\lambda y^*\mathcal{T}_{W_{x^*}^{\Phi(V)}}(E_x^V[M_x^V]))\Phi(U)_{x^*}^{\Phi(V)} & \text{by IHa for } E[M] \\
\rightarrow_\beta^* & (\lambda y^*\mathcal{T}_{W_{x^*}^{\Phi(V)}}(E[M]_x^V))\Phi(U_x^V) & \text{by IHb for } U \\
= & \mathcal{T}_{W_{x^*}^{\Phi(V)}}(E[(\lambda y\, M)U]_x^V). &
\end{array}
$$

Note that the IHa is applicable since $|E[M]|^+ < |E[(\lambda y\, M)U]|^+$, and the IHb is applicable since $|U|^+ < |E[(\lambda y\, M)U]|^+$.

b. *Case* $x$. Clear. *Case* $y \neq x$. Clear. *Case* $\lambda y\, M$.

$$
\begin{array}{lll}
\Phi(\lambda y\, M)_{x^*}^{\Phi(V)} & = & (\lambda \ell, y^*.\mathcal{T}_\ell(M))_{x^*}^{\Phi(V)} \\
 & = & \lambda \ell, y^*.\mathcal{T}_\ell(M)_{x^*}^{\Phi(V)} \\
 & \rightarrow_\beta^* & \lambda \ell, y^*.\mathcal{T}_\ell(M_x^V) \quad \text{by IHa for } M.
\end{array}
$$

Note that the IHa is applicable since by the definition of $|\cdot|^+$ we have $|M|^+ < |\lambda y\, M|^+$.

c. *Case* $E[x[]]$.

$$
\begin{array}{lll}
\mathcal{K}_W(E[x[]])_{x^*}^{\Phi(V)} & = & (x^*\mathcal{K}_W(E))_{x^*}^{\Phi(V)} \\
 & = & \Phi(V)\mathcal{K}_W(E)_{x^*}^{\Phi(V)}.
\end{array}
$$

In case $V = y$ this is

$$\begin{array}{rcl} & = & y^*\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \\ & \to^*_\beta & \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E[y[]]) \quad \text{by IHc for } E, \end{array}$$

and in case $V = \lambda y\, N$ this is

$$\begin{array}{rcl} & = & (\lambda\ell, y^*.\mathcal{T}_\ell(N))\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \\ & \to^*_\beta & (\lambda\ell, y^*.\mathcal{T}_\ell(N))\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x) \quad \text{by IHc for } E \\ & \to_\beta & \lambda y^*\mathcal{T}_{\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)}(N) \\ & \to^*_\beta & \lambda y^*\mathcal{T}_{W^{\Phi(V)}_{x^*}}(E[N]^V_x) \quad \text{by context unwrapping} \\ & = & \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E[x[]]^V_x). \end{array}$$

Note that the IHc is applicable since $|E|^+ < |E[x[]]|^+$. *Case* $E[y[]]$.

$$\begin{array}{rcl} \mathcal{K}_W(E[y[]])^{\Phi(V)}_{x^*} & = & (y^*\mathcal{K}_W(E))^{\Phi(V)}_{x^*} \\ & = & y^*\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \\ & \to^*_\beta & y^*\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E) \quad \text{by IHc for } E \\ & = & \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E[y[]]). \end{array}$$

Note that the IHc is applicable since $|E|^+ < |E[y[]]|^+$. *Case* $E[(\lambda y\, M)[]]$.

$$\begin{array}{rcl} \mathcal{K}_W(E[(\lambda y\, M)[]])^{\Phi(V)}_{x^*} & = & (\lambda y^*\mathcal{T}_W(E[M]))^{\Phi(V)}_{x^*} \\ & \to^*_\beta & \lambda y^*\mathcal{T}_{W^{\Phi(V)}_{x^*}}(E^V_x[M^V_x]) \quad \text{by IHa for } E[M] \\ & = & \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x[(\lambda y\, M^V_x)[]]). \end{array}$$

Note that the IHa is applicable since $|E[M]|^+ < |E[(\lambda y\, M)[]]|^+$. *Case* $E[[]M]$.

$$\begin{array}{rcl} \mathcal{K}_W(E[[]M])^{\Phi(V)}_{x^*} & = & (\lambda y^*\mathcal{T}_W(E[yM]))^{\Phi(V)}_{x^*} \\ & \to^*_\beta & \lambda y^*\mathcal{T}_{W^{\Phi(V)}_{x^*}}(E^V_x[yM^V_x]) \quad \text{by IHa for } E[yM] \\ & = & \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)[[]M^V_x]. \end{array}$$

Note that the IHa is applicable since $|E[yM]|^+ < |E[[]M]|^+$. □

We now show that to any $\beta$-conversion step there corresponds a nonempty finite list of $\beta$-conversion steps in the cps-translation.

**5.4.** LEMMA. *(Simulation). If* $M \to_\beta M'$, *then* $\mathcal{T}_W(M) \to^+_\beta \mathcal{T}_W(M')$.

*Proof.* Consider $E[(\lambda x\, M)V] \to_\beta E[M^V_x]$. Then we have

$$\begin{array}{rcl} \mathcal{T}_W(E[(\lambda x\, M)V]) & = & (\lambda x^*\mathcal{T}_W(E[M]))\Phi(V) \\ & \to_\beta & \mathcal{T}_W(E[M])^{\Phi(V)}_{x^*} \\ & \to^*_\beta & \mathcal{T}_W(E[M^V_x]) \quad \text{by the substitution lemma 5.3.} \end{array}$$

Note that the original redex in $M = E[(\lambda x\, M)V]$, which may have been deep inside the term, is transported to the top level by the cps-translation $\mathcal{T}_W(M)$ of $M$.

It follows from the simulation lemma that to any $\beta$-conversion there corresponds a finite nonempty sequence of $\beta$-conversions in the cps-translation.

**5.5.** *Addition of global control operators.* The simultaneous definition of $\mathcal{T}_W(M)$, $\Phi(V)$ and $\mathcal{K}_W(E)$ has to be extended by three additional clauses for $\mathcal{T}()$:

$$\begin{array}{rcl}
\mathcal{T}_W(E[\mathcal{P}M]) & := & \mathcal{T}_{\lambda x^*.\, x^*\mathcal{K}_W(E)(\lambda \ell, z^*.\, \mathcal{T}_W(E[z]))}(M), \\
\mathcal{T}_W(E[\mathcal{C}M]) & := & \mathcal{T}_{\lambda x^*.\, x^* W(\lambda \ell, z^*.\, \mathcal{T}_W(E[z]))}(M), \\
\mathcal{T}_W(E[\mathcal{A}M]) & := & \mathcal{T}_k(M), \quad \text{where } k := \lambda x.\mathcal{A}x\colon \mathrm{F} \to \mathrm{F}.
\end{array}$$

To see that this is a good definition we have to change our measure to also count the control constants:

**Definition.** *(Measure).*

$$\begin{array}{rcl}
|M| & := & 2 \cdot \mathrm{vars}(M) + \mathrm{cconsts}(M), \\
|E| & := & 2 \cdot \mathrm{vars}(E) + 3 + \mathrm{cconsts}(M),
\end{array}$$

*where* $\mathrm{vars}(\cdot)$ *is the total number of variable occurrences in the term or evaluation context* $\cdot$*, including the bound occurrences, and* $\mathrm{cconsts}(M)$ *is the total number of control constants in* $M$*.*

Then the well-definedness of $\mathcal{T}()$ and $\mathcal{K}()$ follows from

$$\begin{array}{rcl}
|E[xV]| & > & |E|, \\
|E[(\lambda x\, M)V]| & > & |E[M]|, \\
|E[\mathcal{P}M]| & > & |E|, |E[z]|, |M|, \\
|E[\mathcal{C}M]| & > & |E[z]|, |M|, \\
|E[\mathcal{A}M]| & > & |M|, \\
|E[x\,[\,]]| & > & |E|, \\
|E[(\lambda x\, M)[\,]]| & > & |E[M]|, \\
|E[[\,]M]| & > & |E[xM]|.
\end{array}$$

For the extended definition we will need an additional lemma:

**5.6.** Lemma. *If* $W \to_\beta W'$*, then* $\mathcal{T}_W(M) \to_\beta \mathcal{T}_{W'}(M)$*,* $\Phi(W) \to_\beta \Phi(W')$ *and* $\mathcal{K}_W(E) \to_\beta \mathcal{K}_{W'}(E)$*.*

*Proof* by induction on the simultaneous definition of $\mathcal{T}_W(V)$, $\Phi(W)$ and $\mathcal{K}_W(E)$. □

Again we have

**5.7.** LEMMA. *(Context unwrapping).*

a. $\mathcal{T}_{\mathcal{K}_W(E)}(M) \to_\beta^* \mathcal{T}_W(E[M])$, *where* $E[\cdot^B]\colon A$, $W\colon \neg A^*$, $M\colon B$,

b. $\mathcal{K}_{\mathcal{K}_W(E)}(E') \to_\beta^* \mathcal{K}_W(E[E'])$, *where* $E[\cdot^B]\colon A$, $W\colon \neg A^*$, $E'[\cdot^C]\colon B$.

*Proof* as before. To be able to give a proof by induction we have prove simultaneously with parts a and b the following special cases of the substitution lemma (to be proved in general form as our next lemma):

c. $\mathcal{T}_W(M)_{x^*}^{\Phi(V)} \to_\beta^* \mathcal{T}_W(M_x^V)$ with $M = E[xN]$, where $x$ is not free in $E$, $N$ and $W$.

d. $\mathcal{K}_W(E)_{x^*}^{\Phi(V)} \to_\beta^* \mathcal{K}_W(E_x^V)$ with $E = E_1[xE_2]$, where $x$ is not free in $E_1$, $E_2$ and $W$.

We prove parts a-d simultaneously by induction on the following measure.

- For part a: $|E| + |M|$.
- For part b: $|E| + |E'|$.
- For part c: $|M| + |V|$.
- For part d: $|E| + |V|$.

We just have to add two additional clauses at the end of parts a and c.

a. *Case* $E'[\mathcal{P}M]$.

$$\begin{array}{rcl}
\mathcal{T}_{\mathcal{K}_W(E)}(E'[\mathcal{P}M]) & = & \mathcal{T}_{\lambda x^*.\, x^* \mathcal{K}_{\mathcal{K}_W(E)}(E')(\lambda \ell, z^*.\, \mathcal{T}_{\mathcal{K}_W(E)}(E'[z]))}(M), \\
& \to_\beta^* & \mathcal{T}_{\lambda x^*.\, x^* \mathcal{K}_W(E[E'])(\lambda \ell, z^*.\, \mathcal{T}_W(E[E'[z]]))}(M), \\
& = & \mathcal{T}_W(E[E'[\mathcal{P}M]]).
\end{array}$$

Note that we have used the above Lemma 5.6 here. *Case* $E'[\mathcal{A}M]$.

$$\begin{array}{rcl}
\mathcal{T}_{\mathcal{K}_W(E)}(E'[\mathcal{A}M]) & = & \mathcal{T}_k(M) \\
& = & \mathcal{T}_W(E[E'[\mathcal{A}M]]).
\end{array}$$

c. Here we have to add three more cases for $M$: $E[xE'[\mathcal{P}M]]$, $E[xE'[\mathcal{C}M]]$ and $E[xE'[\mathcal{A}M]]$. *Case* $E[xE'[\mathcal{P}M]]$.

$$\begin{array}{rcl}
 & \mathcal{T}_W(E[xE'[\mathcal{P}M]])_{x^*}^{\Phi(V)} & \\
= & \mathcal{T}_{\lambda u^*.\, u^* \mathcal{K}_W(E[xE'])(\lambda \ell, z^*.\, \mathcal{T}_W(E[xE'[z]]))}(M)_{x^*}^{\Phi(V)} & \\
\to_\beta^* & \mathcal{T}_{\lambda u^*.\, u^* \mathcal{K}_W(E[xE'])_{x^*}^{\Phi(V)}(\lambda \ell, z^*.\, \mathcal{T}_W(E[xE'[z]])_{x^*}^{\Phi(V)})}(M) & \\
\to_\beta^* & \mathcal{T}_{\lambda u^*.\, u^* \mathcal{K}_W(E[VE'])(\lambda \ell, z^*.\, \mathcal{T}_W(E[VE'[z]]))}(M) & \\
 & \qquad \text{by IHd for } E[xE'],\ V \text{ and IHc for } E[xE'[z]],\ V & \\
= & \mathcal{T}_W(E[xE']_x^V[\mathcal{P}M]). &
\end{array}$$

Note that the IHd is applicable since $|E[xE']| + |V| < |E[xE'[\mathcal{P}M]]| + |V|$, and the IHc is applicable since $|E[xE'[z]]| + |V| < |E[xE'[\mathcal{P}M]]| + |V|$. *Case* $E[\mathcal{C}M]$. Similarly. *Case* $E[xE'[\mathcal{A}M]]$.

$$\begin{aligned}\mathcal{T}_W(E[xE'[\mathcal{A}M]])^{\Phi(V)}_{x^*} &= \mathcal{T}_k(M)^{\Phi(V)}_{x^*}\\ &= \mathcal{T}_k(M), \quad \text{since } x \text{ is not in } M\\ &= \mathcal{T}_W(E[xE']^V_x[\mathcal{A}M]).\end{aligned}$$

**5.8.** LEMMA. *(Substitution).*

a. $\mathcal{T}_W(M)^{\Phi(V)}_{x^*} \to^*_\beta \mathcal{T}_{W^{\Phi(V)}_{x^*}}(M^V_x)$.

b. $\Phi(W)^{\Phi(V)}_{x^*} \to^*_\beta \Phi(W^V_x)$.

c. $\mathcal{K}_W(E)^{\Phi(V)}_{x^*} \to^*_\beta \mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)$.

*Proof.* We again prove parts a-c simultaneously by induction on the following measure, using the previous lemma on context unwrapping.

- For part a: $|M|^+$.
- For part b: $|W|^+$.
- For part c: $|E|^+$.

Here $|M|^+$ and $|E|^+$ are defined similarly to $|M|$, $|E|$ exept that binding occurrences of variables now count.

a. Here we have to add three more cases for $M$: $E[\mathcal{P}M]$, $E[\mathcal{C}M]$ and $E[\mathcal{A}M]$. *Case* $E[\mathcal{P}M]$.

$$\begin{aligned}& \mathcal{T}_W(E[\mathcal{P}M])^{\Phi(V)}_{x^*}\\ = \;& \mathcal{T}_{\lambda u^*.u^*\mathcal{K}_W(E)(\lambda\ell,z^*.\mathcal{T}_W(E[z]))}(M)^{\Phi(V)}_{x^*}\\ \to^*_\beta \;& \mathcal{T}_{\lambda u^*.u^*\mathcal{K}_W(E)^{\Phi(V)}_{x^*}(\lambda\ell,z^*.\mathcal{T}_W(E[z])^{\Phi(V)}_{x^*})}(M^V_x) \quad \text{by IHa for } M\\ \to^*_\beta \;& \mathcal{T}_{\lambda u^*.u^*\mathcal{K}_{W^{\Phi(V)}_{x^*}}(E^V_x)(\lambda\ell,z^*.\mathcal{T}_{W^{\Phi(V)}_{x^*}}(E^V_x[z]))}(M^V_x)\\ & \qquad \text{by IHd for } E \text{ and IHc for } E[z]\\ = \;& \mathcal{T}_{W^{\Phi(V)}_{x^*}}(E^V_x[\mathcal{P}M^V_x]).\end{aligned}$$

Note that the IHc is applicable since $|E|^+ < |E[\mathcal{P}M]|^+$, and the IHa is applicable since $|E[z]|^+ < |E[\mathcal{P}M]|^+$. *Case* $E[\mathcal{C}M]$. Similarly. *Case* $E[\mathcal{A}M]$.

$$\begin{aligned}\mathcal{T}_W(E[\mathcal{A}M])^{\Phi(V)}_{x^*} &= \mathcal{T}_k(M)^{\Phi(V)}_{x^*}\\ &= \mathcal{T}_k(M^V_x), \quad \text{by IHa for } M\\ &= \mathcal{T}_{W^{\Phi(V)}_{x^*}}(E^V_x[\mathcal{A}M^V_x]).\end{aligned}$$

We now show that in our extended setting with control operators to any $\beta$-conversion step there corresponds a nonempty finite list of $\beta$-conversion steps in the cps-translation, and to any control conversion step there corresponds a possibly empty finite list of $\beta$-conversion steps in the cps-translation. In b–d let $M$ be of type F and $k := \lambda x.\mathcal{A}x\colon \mathrm{F} \to \mathrm{F}$.

Recall again that $k$ is a special variable to be viewed as the top continuation. It has type $\neg A^*$ if $M$ has type $A$.

**5.9.** LEMMA. *(Simulation).*

a. *If* $M \to_\beta M'$, *then* $\mathcal{T}_W(M) \to_\beta^+ \mathcal{T}_W(M')$.

b. *If* $M \to_{\mathcal{P}} M'$, *then* $\mathcal{T}_k(M) \to_\beta^* \mathcal{T}_k(M')$.

c. *If* $M \to_{\mathcal{C}} M'$, *then* $\mathcal{T}_k(M) \to_\beta^* \mathcal{T}_k(M')$.

d. *If* $M \to_{\mathcal{A}} M'$, *then* $\mathcal{T}_k(M) = \mathcal{T}_k(M')$.

*Proof.* a. Unchanged.

b. We have to consider a $\mathcal{P}$-conversion $E[\mathcal{P}M] \to_{\mathcal{P}} E[M\lambda z.\mathcal{A}E[z]]$. Then we have

$$\begin{array}{rll}
 & \mathcal{T}_k(E[\mathcal{P}M]) & \\
= & \mathcal{T}_{\lambda x^*.x^*\mathcal{K}_k(E)(\lambda \ell, z^*.\mathcal{T}_k(E[z]))}(M) & \text{by definition of } \mathcal{T}() \text{ for } \mathcal{P} \\
= & \mathcal{T}_{\lambda x^*.x^*\mathcal{K}_k(E)(\lambda \ell, z^*.\mathcal{T}_\ell(\mathcal{A}E[z]))}(M) & \text{by definition of } \mathcal{T}() \text{ for } \mathcal{A} \\
= & \mathcal{T}_{\lambda x^*.x^*\mathcal{K}_k(E)\Phi(\lambda z.\mathcal{A}E[z])}(M) & \text{by definition of } \Phi() \\
= & \mathcal{T}_{\lambda x^*\mathcal{T}_k(E[x\lambda z.\mathcal{A}E[z]])}(M) & \text{by definition of } \mathcal{T}() \\
= & \mathcal{T}_{\mathcal{K}_k(E[[]\lambda z.\mathcal{A}E[z]])}(M) & \text{by definition of } \mathcal{K}() \\
\to_\beta^* & \mathcal{T}_k(E[M\lambda z.\mathcal{A}E[z]]) & \text{by context unwrapping.}
\end{array}$$

c. We have to consider a $\mathcal{C}$-conversion $E[\mathcal{C}M] \to_{\mathcal{C}} M\lambda z.\mathcal{A}E[z]$. Then we have

$$\begin{array}{rll}
 & \mathcal{T}_k(E[\mathcal{C}M]) & \\
= & \mathcal{T}_{\lambda x^*.x^*k(\lambda \ell, z^*.\mathcal{T}_k(E[z]))}(M) & \text{by definition of } \mathcal{T}() \text{ for } \mathcal{C} \\
= & \mathcal{T}_{\lambda x^*.x^*k(\lambda \ell, z^*.\mathcal{T}_\ell(\mathcal{A}E[z]))}(M) & \text{by definition of } \mathcal{T}() \text{ for } \mathcal{A} \\
= & \mathcal{T}_{\lambda x^*.x^*k\Phi(\lambda z.\mathcal{A}E[z])}(M) & \text{by definition of } \Phi() \\
= & \mathcal{T}_{\lambda x^*\mathcal{T}_k(x\lambda z.\mathcal{A}E[z])}(M) & \text{by definition of } \mathcal{T}() \text{ and } \mathcal{K}() \\
= & \mathcal{T}_{\mathcal{K}_k([]\lambda z.\mathcal{A}E[z])}(M) & \text{by definition of } \mathcal{K}() \\
\to_\beta^* & \mathcal{T}_k(M\lambda z.\mathcal{A}E[z]) & \text{by context unwrapping.}
\end{array}$$

d. We have to consider an $\mathcal{A}$-conversion $E[\mathcal{A}M] \to_{\mathcal{A}} M$. Then we have

$$\mathcal{T}_k(E[\mathcal{A}M]) \quad = \quad \mathcal{T}_k(M) \quad \text{by definition of } \mathcal{T}() \text{ for } \mathcal{A}.$$

**5.10.** THEOREM. *(Griffin). Let $M:A$ be a term of the simply typed $\lambda$-calculus extended by the $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{A}$ constructs. Any reduction (or evaluation) sequence starting with $M$, which uses call-by-value $\beta$-conversion and in addition $\mathcal{P}$-, $\mathcal{C}$- and $\mathcal{A}$-conversions, must terminate after finitely many steps.*

*Proof.* We may assume $M:\mathrm{F}$ (otherwise consider $u:M$ with a new variable $u:\neg A$). Let $k := \lambda x.\mathcal{A}x:\mathrm{F} \to \mathrm{F}$. After the cps-translation $M \mapsto \mathcal{T}_k(M)$ the reduction sequence gives rise to a reduction sequence in the simply typed $\lambda$-calculus, of the following form.

- To any call-by-value $\beta$-conversion there corresponds a nonempty sequence of $\beta$-conversions.
- To any $\mathcal{P}$- or $\mathcal{C}$-conversion there corresponds a (possibly empty) sequence of $\beta$-conversions.
- To any $\mathcal{A}$-conversion there corresponds no $\beta$-conversion.

Hence by the strong normalization result for the simply typed $\lambda$-calculus we are done if we can show that there cannot be an infinite sequence $\mathcal{P}$-, $\mathcal{C}$- or $\mathcal{A}$-conversions in the original sequence. But this is clear, since any such conversion reduces the total number of $\mathcal{P}$, $\mathcal{C}$ or $\mathcal{A}$ operators not under a $\lambda$-abstraction. □

# References


[1] Ulrich Berger and Helmut Schwichtenberg. Program development by proof transformation. In H. Schwichtenberg, editor, *Proof and Computation*, volume 139 of *Series F: Computer and Systems Sciences*, pages 1–45. NATO Advanced Study Institute, International Summer School held in Marktoberdorf, Germany, July 20 – August 1, 1993, Springer Verlag, Berlin, Heidelberg, New York, 1995.

[2] Ulrich Berger and Helmut Schwichtenberg. Program extraction from classical proofs. In D. Leivant, editor, *Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, IN, USA, October 1994*, volume 960 of *Lecture Notes in Computer Science*, pages 77–97. Springer Verlag, Berlin, Heidelberg, New York, 1995.

[3] Ulrich Berger and Helmut Schwichtenberg. The greatest common divisor: a case study for program extraction from classical proofs. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs. International Workshop TYPES '95, Torino, Italy, June 1995. Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 36–46. Springer Verlag, Berlin, Heidelberg, New York, 1996.

[4] William Clinger, Jonathan Rees (editors) H. Abelson, N.I. Adams IV, D.H. Bartley, G. Brooks, R.K. Dybvig, D.P. Friedman, R. Halstead, C. Hanson, C.T. Haynes, E. Kohlbecker, D. Oxley, K.M. Pitman, G.J. Rozas, G.L. Steele Jr., G.J. Sussman, and M. Wand. *Revised$^4$ Report on the Algorithmic Language Scheme*, 1991. Appeared in ACM Lisp Pointers IV, July-September 1991, and also as MIT AI Memo 848b. It can be obtained by anonymous ftp at the two Scheme Repositories, altdorf.ai.mit.edu and nexus.yorku.ca.

[5] Thierry Coquand. A proof of Higman's lemma by structural induction, April 1993.

[6] Anne S. Troelstra (editor). *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1973.

[7] Matthias Felleisen, Daniel P.Friedman, E. Kohlbecker, and B.F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987. Preliminary version: Reasoning with Continuations, in Proceedings of the 1st IEEE Symposium on Logic in Computer Science, 1986.

[8] M. Fischer. Lambda calculus schemata. *Sigplan Notices*, 7:104–109, 1972.

[9] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

[10] Timothy G. Griffin. A formulae–as–types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, 1990.

[11] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.

[12] Sigekatu Kuroda. Intuitionistische Untersuchungen der formalistischen Logik. *Nagoya Mathematical Journal*, 2:35–47, 1951.

[13] A.R. Meyer and M. Wand. Continuation semantics in typed lambda–calculi. In *Proceedings Workshop Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer Verlag, Berlin, Heidelberg, New York, 1985.

[14] Chetan Murthy. Extracting constructive content from classical proofs. Technical Report 90–1151, Dep.of Comp.Science, Cornell Univ., Ithaca, New York, 1990. PhD thesis.

[15] C. St. J. A. Nash-Williams. On well–quasi–ordering finite trees. *Proc. Cambridge Phil. Soc.*, 59:833–835, 1963.

[16] Gordon D. Plotkin. Call–by–name, call–by–value and the $\lambda$–calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[17] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[18] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation–passing style. *Lisp and Symbolic Computation*, 6:289–360, 1993.

[19] Kurt Schütte and Stephen G. Simpson. Ein in der reinen Zahlentheorie unbeweisbarer Satz über endliche Folgen von natürlichen Zahlen. *Archiv für Mathematische Logik und Grundlagenforschung*, 25:75–89, 1985.

[20] Helmut Schwichtenberg. Primitive recursion on the partial continuous functionals. In M. Broy, editor, *Informatik und Mathematik*, pages 251–269. Springer Verlag, Berlin, Heidelberg, New York, 1991.

[21] Helmut Schwichtenberg and Stanley S. Wainer. Ordinal bounds for programs. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 387–406. Birkhäuser, Boston, 1995.

[22] Gunnar Stålmark. Normalization theorems for full first order classical natural deduction. *The Journal of Symbolic Logic*, 56(1):129–149, 1991.

[23] Masako Takahashi. Parallel reductions in $\lambda$–calculus. *Information and Computation*, 118:120–127, 1995.

[24] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 1996.

[25] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics. An Introduction*, volume 121, 123 of *Studies in Logic and the Foundations of Mathematics.* North–Holland, Amsterdam, 1988.