A Type-Theoretic Foundation of Delimited Continuations[†]

Zena M. Ariola[‡]
University of Oregon

Hugo Herbelin
INRIA-Futurs

Amr Sabry[§]
Indiana University

Abstract. There is a correspondence between classical logic and programming language calculi with first-class continuations. With the addition of control delimiters, the continuations become composable and the calculi are believed to become more expressive. We formalise that the addition of control delimiters corresponds to the addition of a single dynamically-scoped variable modelling the special top-level continuation. From a type perspective, the dynamically-scoped variable requires effect annotations. From a logic perspective, the effect annotations can be understood in a standard logic extended with the dual of implication, namely subtraction.

Keywords: callcc, monad, prompt, reset, shift, subcontinuation, subtraction

1. Introduction

Programming practice suggests that control operators add expressive power to purely functional languages. For example, control operators permit the implementation of backtracking (Haynes, 1986), coroutines (Haynes et al., 1986), and lightweight processes (Wand, 1999), which go beyond pure functional programming. Of course, any complete program that uses these abstractions can be globally transformed using the continuation-passing style (CPS) transformation to a purely functional program, but this misses the point. As Felleisen (1990) formalises and proves, the additional expressiveness of control operators comes from the fact that no local transformation of program fragments using control operators is possible.

There is another, simpler, way to formalise the additional expressive power of control operators that is based on the Curry-Howard

[†] Extended version of the conference article "A Type-Theoretic Foundation of Continuations and Prompts." (Ariola et al., 2004)

[‡] Supported by National Science Foundation grant number CCR-0204389

[§] Supported by National Science Foundation grant number CCR-0204389, by a Visiting Researcher position at Microsoft Research, Cambridge, U.K., and by a Visiting Professor position at the University of Genova, Italy.

isomorphism (Howard, 1980). The pure λ-calculus corresponds to intuitionistic logic; extending it with the control operator $\mathcal{C}$ (Felleisen, 1988) makes it equivalent to classical logic (Griffin, 1990), which is evidently more expressive. Felleisen (1990) also showed that the control operator *callcc* is less expressive than $\mathcal{C}$. Ariola and Herbelin (2003) provided the logical explanation: *callcc* corresponds to *minimal classical logic* where it is possible to prove Peirce's law but not double negation elimination, whereas $\mathcal{C}$ corresponds to (non-minimal) classical logic which proves double negation elimination.

This logic-based expressiveness is arguably a simpler approach but it has so far only been applied to a small family of control operators, and the situation for the other control operators is much less understood. For example, the operators *shift* and *reset* (Danvy and Filinski, 1990) are widely believed to be more expressive than $\mathcal{C}$ but it is not clear how to formalise this belief for at least three reasons:

1. Several type systems and type-and-effect systems have been proposed for these operators (Danvy and Filinski, 1989; Murthy, 1992; Kameyama, 2000; Kameyama, 2001), and it is not apparent which of these systems is the "right" one even when moving to the worlds of CPS or monads (Wadler, 1994). Furthermore, for the type-and-effect systems, there are several possible interpretations of the effect annotations, which should be completely eliminated anyway to get a correspondence with a standard logic.

2. Under the Curry-Howard isomorphism, reductions rules correspond to proof normalisation steps and hence we expect the reductions rules for the control operators to be confluent and, in the simply-typed case, strongly normalising. But although the semantics of many control operators (including *shift* and *reset* (Kameyama and Hasegawa, 2003)) can be given using local reduction rules, many of the systems are not confluent, and under none of the type-and-effect systems are the typed terms known to be strongly normalising.

3. The operators *shift* and *reset* can simulate a large number of other computational effects like state and exceptions (Filinski, 1994). This seems to indicate that the understanding of the expressive power of such control operators must also include an understanding of the expressive power of other effects. Thielecke *et. al.* (1999, 2000, 2001) have formalised the expressive power of various combinations of continuations, exceptions and state but their results need to be generalised and explained using standard type systems and logics.

Based on the technical results in this paper, we propose a calculus $\lambda_{\mathcal{C}}^{\rightarrow -}$ which corresponds to classical *subtractive logic* (Rauszer, 1974;

Crolard, 2001), as a foundation in which to reason about "all" control operators and their relative expressive power. Restrictions of classical subtractive logic have been independently (and recently) shown to correspond to coroutines and exceptions (Crolard, 2004) but our connection to the more expressive world of first-class delimited continuations is novel. In more detail, we establish that the additional expressiveness of *shift* and *reset* comes from the fact that *reset* essentially corresponds to a *dynamically-scoped variable*, and in the presence of dynamic scope, continuations can model other effects like state and exceptions. Logically this is apparent from the following equivalences that hold *classically*: $A \wedge \neg T \to B = A \to B \vee T = A \wedge \neg T \to B \wedge \neg T$. In other words, in the presence of control operators, a function that takes a dynamically-scoped environment entry of type $\neg T$ can be expressed as a function that throws an exception of type T, which can be expressed as a function that manipulates a state variable of type $\neg T$. Each of these views leads to a different type-and-effect system for *shift* and *reset*, which can all be embedded in the type $A - T \to B - U$ that uses the subtraction connective. We also note that *shift* and *reset* only use a limited amount of the expressive power of classical subtractive logic. This leaves room for investigating other control operators that are believed to be even more expressive.

The next three sections review some background material and set up the stage for our technical development. Section 2 reviews some of the basic control operators and their connection to classical logic discovered by Griffin. Section 3 improves on Griffin's original formulation by using the $\lambda^{\to}_{\mathcal{C}^-\mathsf{tp}}$-calculus from our previous work (Ariola and Herbelin, 2003). Section 4 reviews the semantics of *shift* and *reset* and their equivalent formulation using $\mathcal{C}$ and # (*prompt*). It explains that an essential ingredient of the semantics is the dynamic nature of control delimiters.

Our contributions are explained in detail in the remaining sections. The main point of Section 5 is that it is possible to explain the semantics of *shift* and *reset* in the context of $\lambda^{\to}_{\mathcal{C}^-\mathsf{tp}}$ by generalising the calculus to include just one dynamically-scoped variable. The section formalises this point by giving the semantics and establishing the soundness and completeness of the calculus with respect to the original semantics. Section 6 investigates various ways to extend the type system of $\lambda^{\to}_{\mathcal{C}^-\mathsf{tp}}$ to accommodate the dynamically-scoped variable. We present three systems (which are closely related to existing type-and-effect systems) and reason about their properties. Sections 7 and 8 explain how to interpret the effect annotations in a standard logic. The first focuses on motivating the dual connective of implication, namely subtraction, and formally defining $\lambda^{\to -}_{\mathcal{C}^-}$. The second focuses on using the subtractive

$$
\begin{array}{lll}
x, a, v, f, c \in \mathit{Vars} & & \\
M, N \in \mathit{Terms} & ::= & x \mid \lambda x.M \mid MN \mid \mathcal{A}\ M \mid \mathcal{K}\ M \mid \mathcal{C}\ M \\
V \in \mathit{Values} & ::= & x \mid \lambda x.M \\
E \in \mathit{EvCtxt} & ::= & \Box \mid E\ M \mid V\ E
\end{array}
$$

Figure 1. Syntax of λ_c

type for explaining and managing the effect annotations. Section 9 develops a CPS transformation for $\lambda_{c^-}^{\rightarrow -}$ and shows that it is consistent with known CPS transformations for *shift* and *reset*. Section 10 concludes with a discussion of other control operators. Finally Appendix A gives the details of the proof of strong normalisation for $\lambda_{c^-}^{\rightarrow -}$.

2. Control and Classical Logic

We review the semantics of continuation-based control operators and their connection to classical logic.

2.1. Operational Semantics

Figure 1 introduces a call-by-value calculus extended with the operators *abort* ($\mathcal{A}$), *callcc* ($\mathcal{K}$), and $\mathcal{C}$. The semantics of the control operators can be described most concisely using the following three operational rules, which rewrite complete programs:

$$
\begin{array}{rcl}
E[\mathcal{A}\ M] & \mapsto & M \\
E[\mathcal{K}\ M] & \mapsto & E[M\ (\lambda x.\mathcal{A}\ E[x])] \\
E[\mathcal{C}\ M] & \mapsto & M\ (\lambda x.\mathcal{A}\ E[x])
\end{array}
$$

In each of the rules, the entire program is split into an evaluation context E representing the continuation, and a current redex to rewrite. The operator $\mathcal{A}$ aborts the continuation returning its subexpression to the top-level; the other two operators capture the evaluation context E and reify it as a function $(\lambda x.\mathcal{A}\ E[x])$. When invoked, this function aborts the evaluation context at the point of invocation, and installs the captured context instead.

The rules show that $\mathcal{C}$ differs from $\mathcal{K}$ in that it does not duplicate the evaluation context. This difference makes $\mathcal{C}$ at least as expressive as both $\mathcal{A}$ and $\mathcal{K}$; it can be used to define them as follows:

$$\mathcal{A}\ M \triangleq \mathcal{C}\ (\lambda\_.\ M) \qquad \text{(Abbrev. 1)}$$

$$\mathcal{K}\ M \triangleq \mathcal{C}\ (\lambda c.\ c\ (M\ c)) \qquad \text{(Abbrev. 2)}$$

We use _ to refer to an anonymous variable.

We will therefore focus on $\mathcal{C}$ in the remainder of the paper, but still occasionally treat $\mathcal{A}$ as a primitive control operator to provide more intuition.

EXAMPLE 1. (A λ_c-term and its evaluation) We have:

$$\mathcal{C}\ (\lambda c.\ 1\ +\ c\ 2) \mapsto (\lambda c.\ 1\ +\ c\ 2)\ (\lambda x.\mathcal{A}\ x) \mapsto\!\!\!\!\rightarrow 1\ +\ \mathcal{A}\ 2 \mapsto 2$$

The invocation of the continuation c abandons the context $1\ +\ [\ ]$.

2.2. Reduction Rules

β_v :	$(\lambda x.M)\ V$	$\rightarrow$	$M[V/x]$
$\mathcal{C}_L$:	$(\mathcal{C}M)\ N$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda f.\mathcal{A}\ (c\ (f\ N))))$
$\mathcal{C}_R$:	$V(\mathcal{C}M)$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ (c\ (V\ x))))$
$\mathcal{C}_{top}$:	$\mathcal{C}M$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ (c\ x)))$
$\mathcal{C}_{idem}$:	$\mathcal{C}\ (\lambda c.\mathcal{C}M)$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ x))$

Figure 2. Reductions of call-by-value $\lambda_{\mathcal{C}}$

Instead of presenting the semantics of $\lambda_{\mathcal{C}}$ as a relation on complete programs, it is possible to give local reduction rules that are applicable anywhere in a term and in arbitrary order. (See Figure 2.) Instead of capturing the entire evaluation context at once, the rules allow one to *lift* the control operation step-by-step until it reaches another control operator. At any point, it is possible to use $\mathcal{C}_{top}$ to start applying M to part of the captured context and then continue lifting the outer $\mathcal{C}$ to accumulate more of the context. The rules are in correspondence with the operational semantics in the sense that there is a standard reduction sequence that reaches an answer which is almost identical to the answer produced by the operational semantics (Felleisen and Hieb, 1992, Th. 3.17). For the term in Example 1, the standard reduction sequence produces $\mathcal{C}\ (\lambda c.c\ 2)$ instead of 2.

2.3. Griffin's Type System

Griffin introduced a type system for $\lambda_{\mathcal{C}}$ where types can also be read as propositions (Griffin, 1990). The set of basic types is assumed to include a special type $\bot$ which represents an empty type or the proposition "false." The type system is given in Figure 3. In the rule $\neg\neg_E$, a continuation accepting a value of type A is given the type $A \rightarrow\ \bot$ (which corresponds to the negation $\neg A$). Thus, the closed term $\lambda x.\mathcal{C}x$ provides a proof of the double negation elimination rule $\neg\neg A \rightarrow A$.

$$b \in \mathit{BaseType} \;=\; \{\ \bot, \ldots\ \}$$
$$A, B \in \mathit{TypeExp} \;::=\; b \mid A \to B$$

$$\frac{}{\Gamma, x : A \vdash x : A}\ Ax \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \to_i$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash M' : A}{\Gamma \vdash M M' : B}\ \to_e \qquad \frac{\Gamma \vdash M : (A \to \bot) \to \bot}{\Gamma \vdash \mathcal{C} M : A}\ \neg\neg_E$$

Figure 3. Type system of $\lambda_{\mathcal{C}}$ (Griffin)

PROPOSITION 2 (Griffin). *A formula A is provable in classical logic iff there exists a closed $\lambda_{\mathcal{C}}$ term M such that $\vdash M : A$.*

3. The $\lambda^{\to}_{\mathcal{C}\text{-}\mathsf{tp}}$-calculus

Our previous work (Ariola and Herbelin, 2003) introduced a refinement of the $\lambda_{\mathcal{C}}$-calculus called the $\lambda^{\to}_{\mathcal{C}\text{-}\mathsf{tp}}$-calculus, which is better behaved both as a reduction system and as a logical system. We introduce this calculus and use it in the remainder of the paper as the basis for developing a uniform framework for reasoning about control operators.

3.1. The Top-Level Continuation

When reasoning about continuations, the issues related to the "top-level" are often swept under the rug. In Section 2.1, we stated that the reduction rules are in correspondence with the operational semantics dismissing the spurious context $\mathcal{C}\ (\lambda c.c\ \Box)$ which surrounds the answer.

Griffin notes a similar situation in the type system of Figure 3. According to the operational semantics, a complete program $E[\mathcal{C}\ M]$ would rewrite to $(M\ (\lambda x.\mathcal{A}\ E[x]))$. For the right-hand side to typecheck $E[x]$ must have type $\bot$, which implies that the rule is only applicable when the program $E[\mathcal{C}\ M]$ has type $\bot$. In Griffin's words, "since there are no closed terms of this type, the rule is useless!" Griffin addressed this problem by requiring that every program M be wrapped in the context $\mathcal{C}\ (\lambda c.\ c\ \Box)$, which provides an explicit binding for the top-level continuation, and by modifying the operational rules to only apply in the context provided by the top-level continuation. In particular, the rule used above becomes:

$$\mathcal{C}\ (\lambda c.E[\mathcal{C}\ M]) \mapsto \mathcal{C}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ E[x]))$$

$$
\begin{array}{rcl}
x, a, v, f \in \mathit{Vars} & & \\
k \in \mathit{KVars} & & \\
\mathit{KConsts} & = & \{\ \mathsf{tp}\ \} \\
M, N \in \mathit{Terms} & ::= & x \mid \lambda x.M \mid MN \mid \mathcal{C}^{-}(\lambda k.J) \\
V \in \mathit{Values} & ::= & x \mid \lambda x.M \\
J \in \mathit{Jumps} & ::= & k\ M \mid \mathsf{tp}\ M
\end{array}
$$

Figure 4. Syntax of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$

$$
\begin{array}{lrcl}
\beta_v: & (\lambda x.M)V & \rightarrow & M[V/x] \\
\mathcal{C}^{-}_{L}: & (\mathcal{C}^{-}(\lambda k.J))\ N & \rightarrow & \mathcal{C}^{-}(\lambda k'.J\ [k'\ (MN)/k\ M]) \\
\mathcal{C}^{-}_{R}: & V\ (\mathcal{C}^{-}(\lambda k.J)) & \rightarrow & \mathcal{C}^{-}(\lambda k'.J\ [k'\ (VM)/k\ M]) \\
\mathcal{C}^{-}_{idem'}: & \mathcal{C}^{-}(\lambda k.k''\ (\mathcal{C}^{-}(\lambda k'.J))) & \rightarrow & \mathcal{C}^{-}(\lambda k.J\ [k''/k']) \\
\mathcal{C}^{-}_{idem}: & \mathcal{C}^{-}(\lambda k.\mathsf{tp}\ (\mathcal{C}^{-}(\lambda k'.J))) & \rightarrow & \mathcal{C}^{-}(\lambda k.J\ [\mathsf{tp}/k']) \\
\mathcal{C}^{-}_{elim}: & \mathcal{C}^{-}(\lambda k.k\ M) & \rightarrow & M \qquad \text{where } k \notin FV(M)
\end{array}
$$

Figure 5. Reductions of call-by-value $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$

Since the issue of the "top-level" becomes critical for functional continuations, it is well-worth an explicit and formal treatment. The $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus provides such a treatment by introducing a special constant tp denoting the top-level continuation.

3.2. Syntax and Semantics

The syntax of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$ is in Figure 4. We distinguish between regular variables, continuation variables, and the special continuation constant tp. We also restrict the use of $\mathcal{C}$ such that the argument is always a λ-abstraction which binds a continuation variable and immediately performs a jump. The restriction does not lose expressiveness but imposes a useful structure on the terms. Indeed the $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus is isomorphic to Parigot's $\lambda\mu$-calculus (1992) extended with a tp continuation constant.

With the presence of tp, the grammar distinguishes two kinds of jumps: a jump to the top-level (which aborts the program execution) and a jump to a previously defined point (which throws a value to a continuation). We abbreviate the terms which abstract over these jumps as follows:

$$\mathcal{A}^{-}M \triangleq \mathcal{C}^{-}(\lambda\_.\mathsf{tp}\ M) \qquad \text{(Abbrev. 3)}$$

$$\mathit{throw}\ k\ M \triangleq \mathcal{C}^{-}(\lambda\_.k\ M) \qquad \text{(Abbrev. 4)}$$

The reduction rules of the $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus are in Figure 5. In addition to the regular substitution operation $M[V/x]$, the rules use structural substitutions of the form $J[k\ (M\ N)/k\ M]$ from the $\lambda\mu$-calculus. Such substitutions can be read as: "for every free occurrence of k, replace the

jump $(k\ M)$ by the jump $(k\ (M\ N))$. The reductions rules are closely related to the ones of λ_c in Figure 2 with small differences: there is a new variant of $\mathcal{C}^-_{idem}$ dealing with a regular continuation variable, there is a new rule $\mathcal{C}^-_{elim}$ that eliminates a superfluous jump whose target is the current continuation, and the rule $\mathcal{C}_{top}$ is no longer needed.

$$\begin{array}{rcl}(\mathcal{C}\ M)^\circ & = & \mathcal{C}^-(\lambda k.\mathsf{tp}\ (M^\circ\ (\lambda x.throw\ k\ x))) \\ (\mathcal{A}\ M)^\circ & = & \mathcal{A}^-\ M^\circ\end{array}$$

Figure 6. Embedding of $\lambda_{\mathcal{C}}$ in $\lambda^{\rightarrow}_{\mathcal{C}^-\mathsf{tp}}$ (interesting clauses)

There is a natural embedding of the terms of the $\lambda_{\mathcal{C}}$-calculus into the $\lambda^{\rightarrow}_{\mathcal{C}^-\mathsf{tp}}$-calculus, under which the semantics of one system is consistent with the semantics of the other (Ariola and Herbelin, 2003). Figure 6 shows the interesting clauses of the embedding: for all other term constructors the embedding is homomorphic. The embedding of $\mathcal{C}$ looks similar to a $\mathcal{C}_{top}$-reduction. The embedding of $\mathcal{A}$ is calculated using Abbrev. 1 followed by a simplification rule. The term in Example 1 is embedded as $\mathcal{C}^-(\lambda k.\mathsf{tp}\ ((\lambda c.\ 1\ +\ c\ 2)\ (\lambda x.throw\ k\ x)))$ which reduces to $\mathcal{C}^-(\lambda k.\mathsf{tp}\ (1\ +\ throw\ k\ 2))$.

3.3. Type System

The set of base types in $\lambda^{\rightarrow}_{\mathcal{C}^-\mathsf{tp}}$ includes $\bot$ which represents an empty type or the proposition "false" as before but it no longer plays a special role in the judgements. Instead, the judgements refer to a distinct special type $\perp\!\!\!\perp$ which is used as the type of jumps, *i.e.*, the type of expressions in the syntactic category J. The type $\perp\!\!\!\perp$ is not a base type; it can never occur as the conclusion of any judgement for terms in the syntactic category M, but it may occur in the context Γ as the return type of a continuation variable.

It is possible (Crolard, 2001) with a bit of juggling to inject a continuation of type $T \rightarrow \perp\!\!\!\perp$ into the type of *functions* $T \rightarrow \bot$ used in the Griffin's system:

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma, k : T \rightarrow \perp\!\!\!\perp, x : T \vdash x : T; T}}{\Gamma, k : T \rightarrow \perp\!\!\!\perp, x : T \vdash k\ x : \perp\!\!\!\perp; T}}{\Gamma, k : T \rightarrow \perp\!\!\!\perp, x : T \vdash \mathcal{C}^-(\lambda\_.k\ x) : \bot; T}}{\Gamma, k : T \rightarrow \perp\!\!\!\perp \vdash \lambda x.\mathcal{C}^-(\lambda\_.k\ x) : T \rightarrow \bot; T}$$

Thus, it does no harm to informally think of $\perp\!\!\!\perp$ as $\bot$ remembering that an explicit coercion is required to move from one to the other. But the special nature of the type $\perp\!\!\!\perp$ can perhaps be best understood by examining the situation in the isomorphic $\lambda\mu$-calculus extended with tp.

$$
\begin{array}{rcl}
b \in \mathit{BaseType} & = & \{\ \bot, \ldots\ \} \\
A, B, T \in \mathit{TypeExp} & ::= & b \mid A \to B \\
\Gamma \in \mathit{Contexts} & ::= & \cdot \mid \Gamma, x : A \mid \Gamma, k : A \to \perp\!\!\!\perp
\end{array}
$$

$$
\frac{}{\Gamma, x : A \vdash x : A; T}\ Ax \qquad \frac{\Gamma, x : A \vdash M : B; T}{\Gamma \vdash \lambda x.M : A \to B; T}\ \to_i
$$

$$
\frac{\Gamma \vdash M : A \to B; T \quad \Gamma \vdash M' : A; T}{\Gamma \vdash M M' : B; T}\ \to_e
$$

$$
\frac{\Gamma, k : A \to \perp\!\!\!\perp \vdash J : \perp\!\!\!\perp; T}{\Gamma \vdash \mathcal{C}^{-}(\lambda k.J) : A; T}\ RAA
$$

$$
\frac{\Gamma, k : A \to \perp\!\!\!\perp \vdash M : A; T}{\Gamma, k : A \to \perp\!\!\!\perp \vdash k\ M : \perp\!\!\!\perp; T}\ \to_e^k \qquad \frac{\Gamma \vdash M : T; T}{\Gamma \vdash \mathsf{tp}\ M : \perp\!\!\!\perp; T}\ \to_e^{\mathsf{tp}}
$$

Figure 7. Type system of $\lambda^{\to}_{\mathcal{C}^{-}\mathsf{tp}}$

In that type system, there is no need for $\perp\!\!\!\perp$: the types of continuation variables are maintained on the right-hand side of the sequent and jumps have no type. In other words, a more accurate understanding of $\perp\!\!\!\perp$ is as a special symbol denoting "no type."

In the original presentation of the type system (Ariola and Herbelin, 2003), the continuation constant tp was given the type $\bot \to \perp\!\!\!\perp$. This is because the purpose there was to characterise the computational content of (non-minimal) classical logic. A common and useful generalisation is to give the top-level continuation the type $T \to \perp\!\!\!\perp$ for some arbitrary but fixed type T (Murthy, 1992). We adopt this generalisation and modify the system to keep track of the special type T. Instead of providing an explicit signature for the type of the constant tp, we keep the type T on the right-hand side of the judgements where it acts as a global parameter to the type system.

The axioms and inference rules of the type system are in Figure 7. In the new system, the rule *RAA* (*Reductio Ad Absurdum*) is similar to the double-negation rule $\neg\neg_E$ in Griffin's system of Figure 3 except that it uses the special type $\perp\!\!\!\perp$ instead of $\bot$. According to the $\to_e^{\mathsf{tp}}$ rule, the special top-level continuation can only be invoked with a value of the distinguished type T. We add the special rule $\to_e^k$ for applying a continuation variable since the axiom rule only applies to regular variables.

The presence of the top-level type may give more information about the type of a term. For example, this judgement:

$$\vdash \mathcal{A}^{-}\ 5 \ ==\ \texttt{"Hello"} : \mathsf{bool};\ \mathsf{int}$$

accurately predicts that the expression returns a bool or jumps to the top-level with an int. But in the following judgement:

$$\vdash \lambda x.x \ + \ \mathcal{A}^{-} \texttt{"Hello"} : \mathsf{int} \to \mathsf{int};\ \mathsf{string}$$

the presence of the type string is actually restrictive and worse, misleading as we explain in Sections 6.1 and 6.2 where we talk about generalisations of the system.

The $\lambda^{\to}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus refines the $\lambda_{\mathcal{C}}$-calculus and has all the right properties.

PROPOSITION 3 (Ariola and Herbelin).

(i) $\lambda^{\to}_{\mathcal{C}^{-}tp}$ *is confluent and typed terms are strongly normalising.*

(ii) Subject reduction: Given $\lambda^{\to}_{\mathcal{C}^{-}tp}$ *terms* M *and* N*, if* $\Gamma \vdash M : A; T$ *and* $M \to N$*, then* $\Gamma \vdash N : A; T$*.*

(iii) A formula A *is provable in minimal classical logic iff there exists a closed* $\lambda^{\to}_{\mathcal{C}^{-}tp}$ *term* M *such that* $\vdash M : A; B$ *is provable without using rule* $\to^{tp}_{e}$ *(in which case the exact form of* B *is irrelevant).*

(iv) A formula A *is provable in classical logic iff there exists a closed* $\lambda^{\to}_{\mathcal{C}^{-}tp}$ *term* M *such that* $\vdash M : A; \bot$*.*

4. Functional Continuations

We aim to extend the results for $\mathcal{C}$ to the more expressive control operators associated with delimited functional continuations. We focus on three of the basic operators: *shift* which captures a functional continuation (Danvy and Filinski, 1990), and the two identical control operators # and *reset* (the first introduced by Felleisen (1988) and the second introduced by Danvy and Filinski (1990)) which delimit continuations.

4.1. The Operators *shift* and *reset*

The operational semantics of *shift* ($\mathcal{S}$) and *reset* (#) in the context of a call-by-value calculus is given below:

$$\begin{array}{rcl} x, a, v, f, c \in \mathit{Vars} & & \\ M \in \mathit{Terms} & ::= & x \mid \lambda x.M \mid MM \mid \mathcal{S}\ M \mid \#\ M \\ V \in \mathit{Values} & ::= & x \mid \lambda x.M \\ E \in \mathit{EvCtxt} & ::= & \Box \mid E\ M \mid V\ E \mid \#\ E \end{array}$$

$$\begin{array}{rcl} E[\#\ V] & \mapsto & E[V] \\ E_{\downarrow}[\#\ (E_{\uparrow}[\mathcal{S}\ M])] & \mapsto & E_{\downarrow}[\#\ (M\ (\lambda x.\#\ E_{\uparrow}[x]))] \end{array}$$

$$
\begin{array}{lll}
x, a, v, f, c \in \mathit{Vars} & & \\
M, N \in \mathit{Terms} & ::= & x \mid \lambda x.M \mid MN \mid \mathcal{C}M \mid \#M \\
V \in \mathit{Values} & ::= & x \mid \lambda x.M
\end{array}
$$

Figure 8. Syntax of $\lambda_{\mathcal{C}\#}$

The first rule indicates that the role of a *reset* terminates when its subexpression is evaluated; indeed the point of *reset* is to *delimit* the continuation captured during the evaluation of its subexpression. When compared to the semantics of $\mathcal{C}$ or $\mathcal{K}$ given in Section 2.1, we note at least three points about the semantics of $\mathcal{S}$:

1. Unlike either $\mathcal{C}$ or $\mathcal{K}$, the control operator $\mathcal{S}$ only captures part of the surrounding evaluation context. This surrounding context is split into three parts: a part $E_\uparrow$ which extends to the *closest* occurrence of a *reset*, the occurrence of the *reset* itself, and the rest of the evaluation context $E_\downarrow$ surrounding the first *reset* and which may include other occurrences of *reset*. It is an error to use $\mathcal{S}$ in a context that is not surrounded by a *reset*.

2. Like $\mathcal{C}$ but unlike $\mathcal{K}$, the action of capturing the context by $\mathcal{S}$ also aborts it. In particular, the context $E_\uparrow$ is removed and "shifted" inside the captured continuation.

3. Unlike $\mathcal{C}$ or $\mathcal{K}$, the captured context is reified into a function which does not include an $\mathcal{A}$. Indeed, the reified continuation is a *functional continuation* whose invocation returns to the point of invocation just like an ordinary function application.

The term in Example 1 rewritten with *shift* and *reset* evaluates as follows:

$$
\begin{array}{rcl}
\#\ \mathcal{S}\ (\lambda c.\ 1\ +\ c\ 2) & \mapsto & \#\ ((\lambda c.\ 1\ +\ c\ 2)\ (\lambda x.\ \#\ x)) \\
& \mapsto\!\!\!\!\to & \#\ (1\ +\ \#\ 2) \\
& \mapsto\!\!\!\!\to & 3
\end{array}
$$

The invocation of the continuation c does not abort, but returns to the context $1\ +\ [\ ]$. In general, functional continuations are *composable*. For example, $\#\ (\mathcal{S}\ (\lambda c.\ c\ (c\ 1))\ +\ 2)$ returns 5 but the same term with a $\mathcal{C}$ returns 3.

4.2. The Operators $\mathcal{C}$ and

There is an equivalent formulation of *shift* and *reset* using $\mathcal{C}$ and # (#). This representation is more suitable for our purposes since we already have all the machinery to deal with $\mathcal{C}$.

β_v :	$(\lambda x.M)V$	$\rightarrow$	$M[V/x]$
$\mathcal{C}_L$:	$(\mathcal{C}M)N$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda f.\mathcal{A}\ (c\ (f\ N))))$
$\mathcal{C}_R$:	$V(\mathcal{C}M)$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ (c\ (V x))))$
$\mathcal{C}_{idem}$:	$\mathcal{C}\ (\lambda c.\mathcal{C}M)$	$\rightarrow$	$\mathcal{C}\ (\lambda c.M(\lambda x.\mathcal{A}\ x))$
$\#_{\mathcal{C}}$	$\#(\mathcal{C}M)$	$\rightarrow$	$\#(M\ (\lambda x.\mathcal{A}\ x))$
$\#_v$	$\#V$	$\rightarrow$	V

Figure 9. Reductions of call-by-value $\lambda_{\mathcal{C}\#}$

The syntax of $\lambda_{\mathcal{C}\#}$ is given in Figure 8: it consists of adding #-expressions to the syntax of $\lambda_{\mathcal{C}}$ in Figure 1. The intention is that the continuation captured by $\mathcal{C}$ will be delimited by the #. No other changes to the semantics of $\mathcal{C}$ is assumed. In particular, the continuation captured by $\mathcal{C}$ still includes an $\mathcal{A}$. Since $\mathcal{A}$ is an abbreviation for a particular use of $\mathcal{C}$, its action is also delimited by the #. This also means that it is an error to use $\mathcal{C}$ in a context that is not surrounded by a #.

Formally, we give the semantics using local reduction rules in Figure 9. The reduction rules extend those of Figure 2 with rules for the # but omit the $\mathcal{C}_{top}$ rule. All uses of the omitted rule can be simulated with the new $\#_{\mathcal{C}}$ rule, except for the erroneous occurrences not surrounded by a #.

The use of $\lambda_{\mathcal{C}\#}$ to study functional continuations defined by *shift* and *reset* is justified by the following facts. A # is just a synonym for *reset* and the operators $\mathcal{S}$ and $\mathcal{C}$ can be mutually simulated as follows (Filinski, 1994):

$$\begin{aligned}\mathcal{S}\ M &\triangleq \mathcal{C}\ (\lambda c.M\ (\lambda v.\#\ (c\ v)))\\ \mathcal{C}\ M &\triangleq \mathcal{S}\ (\lambda c.M\ (\lambda x.\mathcal{A}\ (c\ x)))\end{aligned}$$

The mutual simulation is based on the facts that $\mathcal{A}\ M = \mathcal{S}\ (\lambda\_.M)$ and that for any value V, $\mathcal{A}\ (\#\ V) = \mathcal{A}\ V$ and $\#\ (\mathcal{A}\ V) = \#\ V$ (Kameyama and Hasegawa, 2003).

4.3. Dynamic Scope of

One of the original motivations for introducing the notion of a control delimiter is that it provides an explicit "top-level" for its subexpression (Felleisen, 1988). It is therefore clear that the original $\lambda^{\rightarrow}_{\mathcal{C}^-\mathsf{tp}}$ which has a *constant* corresponding to the top-level continuation needs to be extended by making the top-level continuation a *variable*. In this section, we explain why this variable cannot be a statically-scoped variable.

Consider a possible embedding of the $\lambda_{\mathcal{C}\#}$ expression $\#M$ in a generalisation of $\lambda^{\rightarrow}_{\mathcal{C}^-\mathsf{tp}}$ where the constant tp is replaced by a class of regular

statically-scoped continuation variables $\{\ tp, tp', \ldots\ \}$:

$$\begin{array}{l}(\#\ M)^\circ\ =\ \mathcal{C}^{-}(\lambda tp.\ tp\ M^\circ)\\ (\mathcal{A}\ M)^\circ\ =\ \mathcal{C}^{-}(\lambda\_.\ tp\ M^\circ)\end{array}$$

The embedding uses a single continuation variable tp to denote the current top-level continuation: a # expression rebinds this variable, and an $\mathcal{A}$-expression invokes it. For simple examples, this has the correct semantics but as soon as things get more complicated we run into problems.

EXAMPLE 4. (Static vs. Dynamic #) Using the $\lambda_{\mathcal{C}\#}$ reductions, the term $\#(\#(\lambda\_.\ \mathcal{A}\ (\lambda\_.\ 3))\ (\lambda\_.\ \mathcal{A}\ 4))$ reduces as follows:

$$\begin{array}{ll}& \#(\#(\lambda\_.\ \mathcal{A}\ (\lambda\_.\ 3))\ (\lambda\_.\ \mathcal{A}\ 4))\\ \rightarrow & \#((\lambda\_.\ \mathcal{A}\ (\lambda\_.\ 3))\ (\lambda\_.\ \mathcal{A}\ 4))\\ \rightarrow & \#\mathcal{A}\ (\lambda\_.\ 3) \twoheadrightarrow (\lambda\_.\ 3)\end{array}$$

Using the suggested embedding, the corresponding $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$ term is:

$$\begin{array}{l}\mathcal{C}^{-}(\lambda tp.\ tp\ ((\mathcal{C}^{-}(\lambda tp'.\ tp'\ (\lambda\_.\ \mathcal{C}^{-}(\lambda\_.\ tp'\ (\lambda\_.\ 3)))))\\ \qquad (\lambda\_.\ \mathcal{C}^{-}(\lambda\_.\ tp\ 4))))\end{array}$$

where we have α-renamed one of the occurrences of the statically-scoped variable tp. Then, if we adopt the reduction rules of the original $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus, treating tp as a regular continuation variable, we have:

$$\begin{array}{ll}\twoheadrightarrow_{\mathcal{C}^{-}_{L},\beta} & \mathcal{C}^{-}(\lambda tp.\ tp\ (\mathcal{C}^{-}(\lambda tp'.tp'\ (\mathcal{C}^{-}(\lambda\_.\ tp'\ 3)))))\\ \rightarrow_{\mathcal{C}^{-}_{idem}} & \mathcal{C}^{-}(\lambda tp.\ tp\ (\mathcal{C}^{-}(\lambda tp'.\ tp'\ 3)))\\ \rightarrow_{\mathcal{C}^{-}_{idem}} & \mathcal{C}^{-}(\lambda tp.\ tp\ 3)\\ \rightarrow_{\mathcal{C}^{-}_{elim}} & 3\end{array}$$

which is inconsistent with the result given in $\lambda_{\mathcal{C}\#}$.

The source of the inconsistency is the α-renaming step. Indeed, in the original $\lambda_{\mathcal{C}\#}$ term, the first occurrence of $\mathcal{A}$ is statically associated with the second occurrence of #. However, after one reduction step this occurrence of $\mathcal{A}$ is actually associated with the first occurrence of #. In $\lambda_{\mathcal{C}\#}$, when a control operation is evaluated, the evaluation refers to the *dynamically-closest* occurrence of a #. We formalise this idea in the next section.

INTERMEZZO 5. *The approach of using static scope for delimiters has been explored further by Thielecke (*2002*) and Kameyama (*2001*). Thielecke considers several variations of control operators with different scope rules. Using his notation, we have* $\#M$ = **here** M *and*

$$
\begin{array}{l}
x, a, v, f, c \in \mathit{Vars} \\
k \in \mathit{StaticKVars} \\
\mathit{DynKVars} = \{\ \widehat{tp}\ \} \\
M, N \in \mathit{Terms} \ ::= \ x \mid \lambda x.M \mid MN \mid \mathcal{C}^{-}(\lambda k.J) \mid \mathcal{C}^{-}(\lambda \widehat{tp}.\ J) \\
\quad V \in \mathit{Values} \ ::= \ x \mid \lambda x.M \\
\quad J \in \mathit{Jumps} \ ::= \ k\ M \mid \widehat{tp}\ M
\end{array}
$$

Figure 10. Syntax of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$

$\mathcal{A}\ M = \mathbf{go}\ M$. *He presents a system with static scope* $\vdash_{\mathbf{s}}$ *and shows that it corresponds to classical logic. Note that* β*-reduction is not fully definable in this system as it has only a single pair of operators* **here**/**go** *and hence the renaming of static* **here**/**go** *bindings, that may be necessary to avoid a capture of* **go***, is not expressible. Kameyama also develops a type system and a semantics for a* static *variant of shift and reset and proves type soundness. His type system formalises the fact that one needs to maintain a sequence of top-level continuation variables in the judgements. Indeed, an expression might be evaluated in the scope of several occurrences of the #, each represented by a statically-scoped (*α*-renamed) continuation variable.*

5. The $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-Calculus: Syntax and Semantics

To maintain the proper semantics of functional continuations, a # should be mapped to a use of $\mathcal{C}^{-}$ but with the receiver being a *dynamic* λ-abstraction. We formalise this idea using an extension of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$ called the $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-calculus.

5.1. Syntax

As the syntax in Figure 10 shows, we have two distinct uses of $\mathcal{C}^{-}$: one for regular statically-scoped continuation variables and one for the unique dynamically-scoped continuation variable $\widehat{tp}$.

We introduce a new abbreviation that is just like $\mathcal{A}^{-}M$ except that it refers to the special dynamic variable $\widehat{tp}$:

$$\widehat{\mathcal{A}^{-}}M \ \triangleq\ \mathcal{C}^{-}(\lambda\_.\widehat{tp}\ M) \qquad \text{(Abbrev. 5)}$$

The anonymous variable _ ranges only over regular variables and static continuation variables but not over the dynamic variable. Similarly, the notions of free and bound variables only apply to the regular variables and the static continuation variables but not the dynamic variable.

$$
\begin{array}{rcl}
(\#\ M)^\circ & = & \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ \widehat{tp}\ M^\circ) \\
(\mathcal{C}\ M)^\circ & = & \mathcal{C}^{\text{-}}(\lambda k.\widehat{tp}\ (M^\circ\ (\lambda x.\mathit{throw}\ k\ x))) \\
(\mathcal{A}\ M)^\circ & = & \widehat{\mathcal{A}^{\text{-}}}\ M^\circ
\end{array}
$$

Figure 11. Embedding of $\lambda_{\mathcal{C}\#}$ into $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ (interesting clauses)

$$
\begin{array}{lll}
\beta_v: & (\lambda x.M)V & \rightarrow\ M\ \hat{[}V/x\hat{]} \\
\mathcal{C}^{\text{-}}_L: & (\mathcal{C}^{\text{-}}(\lambda k.J))\ N & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda k'.J\ \hat{[}k'\ (MN)/k\ M\hat{]}) \\
\mathcal{C}^{\text{-}}_R: & V\ (\mathcal{C}^{\text{-}}(\lambda k.J)) & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda k'.J\ \hat{[}k'\ (VM)/k\ M\hat{]}) \\
\mathcal{C}^{\text{-}}_{idem'}: & \mathcal{C}^{\text{-}}(\lambda k.k''\ (\mathcal{C}^{\text{-}}(\lambda k'.J))) & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda k.J\ [k''/k']) \\
\mathcal{C}^{\text{-}}_{idem}: & \mathcal{C}^{\text{-}}(\lambda k.\widehat{tp}\ (\mathcal{C}^{\text{-}}(\lambda k'.J))) & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda k.J\ \hat{[}\widehat{tp}/k'\hat{]}) \\
\mathcal{C}^{\text{-}}_{elim}: & \mathcal{C}^{\text{-}}(\lambda k.k\ M) & \rightarrow\ M \qquad \text{where } k \notin FV(M) \\
 & & \\
\mathcal{C}^{\text{-}}_{idem'_{\widehat{tp}}}: & \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ k\ (\mathcal{C}^{\text{-}}(\lambda k'.J))) & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ J\ [k/k']) \\
\mathcal{C}^{\text{-}}_{idem_{\widehat{tp}}}: & \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ \widehat{tp}\ (\mathcal{C}^{\text{-}}(\lambda k'.J))) & \rightarrow\ \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ J\ \hat{[}\widehat{tp}/k'\hat{]}) \\
\mathcal{C}^{\text{-}}_{elim'}: & \mathcal{C}^{\text{-}}(\lambda\widehat{tp}.\ \widehat{tp}\ V) & \rightarrow\ V \qquad \textit{even if } \widehat{tp} \in V
\end{array}
$$

Figure 12. Reductions of call-by-value $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$

The embedding of $\lambda_{\mathcal{C}\#}$ in $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ we aim for is given in Figure 11. The embedding of a #-expression is similar to what we attempted in the last section but modified to take the dynamic nature of the # into account. The embeddings of $\mathcal{C}$ and $\mathcal{A}$ are just like the ones from Figure 6 except that they refer to $\widehat{tp}$ instead of tp.

5.2. Reductions

In order to define the semantics of $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ we first need an understanding of the semantics of dynamic scope in the absence of control operators (Moreau, 1998). A dynamic abstraction $(\lambda\widehat{x}.M)$ is generally like a regular function in the sense that when it is called with a value V, the formal parameter $\widehat{x}$ is bound to V. But:

DS1 The association between $\widehat{x}$ and V established when a function is called lasts *exactly* as long as the evaluation of the body of the function. In particular, the association is disregarded when the function returns, and this happens *even* if the function returns something like $(\lambda\_.\ldots\widehat{x}\ldots)$ which contains an occurrence of $\widehat{x}$: no closure is built and the occurrence of $\widehat{x}$ in the return value is allowed to escape.

DS2 The association between $\widehat{x}$ and V introduced by one function may capture occurrences of $\widehat{x}$ that escape from other functions. For

example, if we have two dynamic functions f and g with f directly or indirectly calling g, then during the evaluation of the body of f, occurrences of $\widehat{x}$ that are returned in the result of g will be captured. Turning this example around, the occurrences of $\widehat{x}$ that escape from g are bound by the closest association found up the dynamic chain of calls.

In the presence of control operators, the situation is more complicated because the evaluation of the body of a function may abort or throw to a continuation instead of returning, and capturing a continuation inside the body of a function allows one to re-enter (and hence re-evaluate) the body of the function more than once.

Despite the additional complications, the reduction rules of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ in Figure 12 look essentially like the reduction rules of the $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$-calculus. We discuss these rules informally here and defer the correctness argument until the next section. The first group of reductions is a copy of the corresponding reductions in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$ with occasional dynamic annotations. As implied by DS2, the meta-operation of substitution must allow for the capture of $\widehat{tp}$. The substitution operations $M\ \hat{[}V/x\hat{]}$ and $J\ \hat{[}k\ (M\ N)/k\ M\hat{]}$ do not rename $\widehat{tp}$ but are otherwise standard. For example, if $V = (\lambda\_.\mathcal{C}^{-}(\lambda\_.\widehat{tp}\ y))$ then:

$$\begin{array}{rl} & (\lambda y.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y)))\ \hat{[}V/x\hat{]} \\ = & \lambda y'.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ ((\lambda\_.\mathcal{C}^{-}(\lambda\_.\ \widehat{tp}\ y))\ y')) \end{array}$$

which captures $\widehat{tp}$ but not y. The second group of rules arises because $\mathcal{C}^{-}$-abstractions come in two flavours, so we essentially need to consider every rule that is applicable to expressions of the form $\mathcal{C}^{-}(\lambda k.J)$ and modify it, if appropriate, to apply to expressions of the form $\mathcal{C}^{-}(\lambda\widehat{tp}.J)$. We only modify and include three rules: two *idem* rules which look as usual, and the rule $\mathcal{C}^{-}_{elim'}$ which explicitly allows occurrences of $\widehat{tp}$ in V to escape as suggested by DS1 above. The other rules are not needed to simulate the semantics of $\lambda_{\mathcal{C}\#}$.

5.3. Properties of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ Reductions

We have two semantics for $\lambda_{\mathcal{C}\#}$: one given by the reduction rules of Figure 9 and one given by the embedding in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ and the reduction rules in Figure 12. We need to verify that the two semantics are consistent. A simple way to perform this verification would be to check that the reduction rules on one side can be simulated by the reduction rules on the other side, but this very strong correspondence does not hold in our case. What does hold is a correspondence up to *operational equivalence*.

Given a reduction relation X, and two terms M and N, possibly containing free variables, we say $M \simeq_X N$ if for every context P which binds all the free variables of M and N, $P[M] \twoheadrightarrow_X V_1$ iff $P[N] \twoheadrightarrow_X V_2$ for some values V_1 and V_2. For example, any two terms that are convertible using the reduction relation are operationally equivalent. Two non-convertible terms may still be equivalent if no sequence of reductions in any context can invalidate their equivalence. An example of this kind is the equivalence $(\lambda x.x)\ (y\ z) \simeq_{\mathcal{C}\#} (y\ z)$.

To show that the two semantics are consistent up to operational equivalence, we need to consider the following two cases. If we evaluate a $\lambda_{\mathcal{C}\#}$-program using the original semantics and that evaluation produces an answer V, then the embedding of the program in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ should be operationally equivalent to the embedding of V. Similarly, if the evaluation of a $\lambda_{\mathcal{C}\#}$-program is erroneous or diverges in the original semantics, then it remains undefined after the embedding. Turning the implication around, if evaluating a $\lambda_{\mathcal{C}\#}$-program by embedding it in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ and using the reduction rules of Figure 12 produces a value V, then the original program is operationally equivalent to the mapping of V back to $\lambda_{\mathcal{C}\#}$. The interesting clauses of this mapping are defined as follows:

$$\begin{aligned} (\mathcal{C}^{-}(\lambda k.J))^{\bullet} &= \mathcal{C}\ (\lambda k.J^{\bullet}) \\ (\mathcal{C}^{-}(\lambda \widehat{tp}.J))^{\bullet} &= \#\ J^{\bullet} \\ (\widehat{tp}\ M)^{\bullet} &= M^{\bullet} \end{aligned}$$

The only non-trivial clause is the first one where we silently allow continuation variables to occur in $\lambda_{\mathcal{C}\#}$. The problem as we shall see is that continuations variables are special in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ and the translation loses information about their special status. The third clause emphasises the fact that the top-level continuation is implicit in $\lambda_{\mathcal{C}\#}$.

PROPOSITION 6. *Let M be a* closed *$\lambda_{\mathcal{C}\#}$-term:*

- *If $M \twoheadrightarrow_{\lambda_{\mathcal{C}\#}} V$ then $M^{\circ} \simeq_{\mathcal{C}^{-}\widehat{tp}} V^{\circ}$.*
- *If $M^{\circ} \twoheadrightarrow_{\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}} V$ then $M \simeq_{\mathcal{C}\#} V^{\bullet}$.*

The proof of the first clause reduces to checking that embedding both sides of every $\lambda_{\mathcal{C}\#}$-reduction produces semantically-equivalent terms in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$. For some of the cases, like the reduction $\#_v$, the embedded terms are related by a corresponding reduction in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ and hence are obviously semantically-equivalent. For the $\mathcal{C}_{idem}$ case, the embedded left-hand side does not reduce to the embedded right-hand side, but both can reduce to a common term, and hence are again semantically-equivalent. The lifting rules $\mathcal{C}_L$ are $\mathcal{C}_R$ introduce a complication: proving

the equivalence of the embedded terms requires using the following equivalence:

$$(\lambda x.\mathit{throw}\ k\ x)\ M \quad \simeq_{\mathcal{C}^{-}\widehat{tp}} \quad \mathit{throw}\ k\ M$$

even when M is not a value. This happens because in contrast to the regular substitution operation, structural substitutions can replace arbitrary jumps $(k\ M)$ by $(k\ (V\ M))$ even when M is not a value. The mismatch reflects more the design choices of $\lambda_{c\#}$ and $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ rather than an inconsistency. Indeed it would be possible to design a variant of $\lambda_{c\#}$ in which one could throw a term M to a continuation instead of a value, and it would be possible to design a variant of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ where all arguments to jumps are restricted to be values. In general, requiring that all jump arguments be values forces one to evaluate the argument to the jump in some continuation and then erasing this continuation, instead of the equivalent but more efficient choice of first erasing the continuation and then evaluating the argument to the jump (Ganz et al., 1999).

The proof of the second clause reduces to proving:

1. For all $\lambda_{c\#}$-terms M, we have $M \simeq_{c\#} M^{\circ\bullet}$
2. For every $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-reduction $M \rightarrow N$, we have $M^{\bullet} \simeq_{c\#} N^{\bullet}$.

The proof of the first statement is straightforward. Note however that proving that $\mathcal{C}\ M$ is equivalent to $(\mathcal{C}\ M)^{\circ\bullet}$ requires using $\mathcal{C}_{top}$ which is not a reduction rule but otherwise a valid operational equivalence.

When attempting to prove the second statement, we encounter a problem related to free continuation variables. Even though programs are closed terms, reductions can happen anywhere including under binders and hence it is possible for a $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-reduction to manipulate an open term. In particular, consider $\mathcal{C}^{-}_{idem'_{\widehat{tp}}}$ where the continuation variable k is free. The right-hand side maps to the $\lambda_{c\#}$-term $(\#J[k/k']^{\bullet})$ but the left-hand side is equivalent to the $\lambda_{c\#}$-term $(\#J[(\lambda x.\mathcal{A}\ (k\ x))/k']^{\bullet})$. Since the variable k is not special in $\lambda_{c\#}$ it could, as far as the $\lambda_{c\#}$-theory is concerned, be substituted with an arbitrary procedure and hence it is definitely not the case that one can assume that k and $(\lambda x.\mathcal{A}\ (k\ x))$ are operationally equivalent. This assumption would be correct if we could somehow guarantee that k will be substituted by a continuation. In a complete program, this is clearly the case as the left-hand side must occur in a context $\ldots\mathcal{C}^{-}(\lambda k.\ldots\Box\ldots)\ldots$ which binds k to a continuation. We just need to make this information explicit in the statement of the proposition (Sabry and Felleisen, 1993, Lemma 19):

2'. Let $M \to N$ be a $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-reduction, and let $k_1, \ldots, k_n$ be the free continuation variables in M. We then have $\mathcal{C}\ (\lambda k_1 \ldots \mathcal{C}\ (\lambda k_n.M^{\bullet})) \simeq_{\mathcal{C}\#} \mathcal{C}\ (\lambda k_1 \ldots \mathcal{C}\ (\lambda k_n.N^{\bullet}))$.

The proof of the modified clause proceeds by cases. For the reduction $\mathcal{C}^{-}_{elim}$ we use the fact that even though $\mathcal{C}_{elim}$ is not a reduction of $\lambda_{\mathcal{C}\#}$, it is a valid equivalence. The reductions $\mathcal{C}^{-}_{L}$ and $\mathcal{C}^{-}_{R}$ require the following equivalences in $\lambda_{\mathcal{C}\#}$ where k is a continuation variable and M may not be a value:

$$(\lambda x.k\ (V\ x))\ M\ \simeq_{\mathcal{C}\#}\ k\ (V\ M)$$
$$(\lambda x.k\ (x\ N))\ M\ \simeq_{\mathcal{C}\#}\ k\ (M\ N)$$

These again allow one to jump with a non-value. All the required $\lambda_{\mathcal{C}\#}$ equivalences are valid (Kameyama and Hasegawa, 2003; Sabry and Felleisen, 1993).

6. The $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$-Calculus: Types

There are several natural type (and effect) systems for $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ that have appeared in the literature of delimited continuations in one form or another. We investigate three of these systems and reason about their properties. The first system has no effect annotations which we show are essential for strong normalisation.

6.1. System I: Fixed Answer Type

Even though the status of the top-level continuation changes from a constant to a dynamic variable, the type system of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\mathsf{tp}}$ in Figure 7 is essentially a sound type system for $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$.

DEFINITION 7. The type system $\Lambda^{\rightarrow\ fixed}_{\mathcal{C}^{-}\widehat{tp}}$ is identical to the system of Figure 7 with tp changed into the dynamic continuation variable $\widehat{tp}$ and the rule RAA duplicated for the case the dynamic continuation variable:

$$\frac{\Gamma \vdash J : \bot\!\!\!\bot; T}{\Gamma; T \vdash \mathcal{C}^{-}(\lambda\widehat{tp}.\ J) : T; T}\ RAA^{\widehat{tp}}$$

This type system can be explained on the $\lambda_{\mathcal{C}\#}$ side as an extension of Griffin's type system where the judgements have an additional type T on the right-hand side. This is identical to Murthy's type system (1992) and the type system one gets when defining the control operations on

top of a continuation monad with a fixed answer type T (Wadler, 1994). In all these systems, the rule for typing # is:

$$\frac{\Gamma \vdash M : T; T}{\Gamma \vdash \#M : T; T}\ \#$$

and the type of $\mathcal{A}$ is $T \to A$. Thus, the control operators can only be used in contexts that agree with the top-level type. Despite this restriction, these types may be adequate for some applications (Filinski, 1994) but logically they are unacceptable as they are not strong enough to guarantee strong normalisation.

EXAMPLE 8. (Loss of SN) Let $T = (\mathsf{int} \to \mathsf{int}) \to \mathsf{int}$ be the fixed top-level type, then we can calculate the following types for the given expressions:

$$\begin{array}{lll} g :: \mathsf{int} \to T & = & \lambda\_.\lambda\_.0 \\ f :: T & = & \lambda x.\ (\#\ (g\ (x\ 0)))\ x \\ s :: \mathsf{int} \to \mathsf{int} & = & \lambda\_.(\mathcal{A}\ f) \\ e :: \mathsf{int} & = & f\ s \end{array}$$

Despite being well-typed, e goes into an infinite loop:

$$f\ s \twoheadrightarrow (\#\ (g\ (\mathcal{A}\ f)))\ s \twoheadrightarrow (\#\ f)\ s \twoheadrightarrow f\ s$$

One can confirm this behaviour using Filinski's ML library (1994) which provides *shift* and *reset* on top of a continuation monad with a fixed answer type by running the transliteration of our term:

```
let val g = fn _ => fn _ => 0
    val f = fn x => reset (fn () =>  g (x 0)) x
in f (fn _ => shift (fn _ => f ))
end
```

6.2. System II: Dynamic Scope as Effects

The requirement that all occurrences of $\widehat{tp}$ (or equivalently, all occurrences of the #) are typed with the same fixed top-level type T is overly restrictive. Each introduction of $\widehat{tp}$ can be given a different type as shown in rule $RAA^{\widehat{tp}}$ in Figure 13: if $\widehat{tp}$ is introduced in a context expecting a value of type A, then it can be called with arguments of type A. In other words, a new top-level type is introduced for the typing of J. Thus, the judgement:

$$\Gamma \vdash M : A; T$$

$$
\begin{array}{rcl}
b \in \mathit{BaseType} & = & \{\ \bot, \ldots\ \} \\
A, B, T \in \mathit{TypeExp} & ::= & b \mid A \to_T B \\
\Gamma \in \mathit{Contexts} & ::= & \cdot \mid \Gamma, x : A \mid \Gamma, k : A \to_T \mathbb{\bot}
\end{array}
$$

$$
\frac{}{\Gamma, x : A \vdash x : A; T}\ Ax \qquad \frac{\Gamma, x : A \vdash M : B; T}{\Gamma \vdash \lambda x.M : A \to_T B; T'}\ \to_i
$$

$$
\frac{\Gamma \vdash M : A \to_T B; T \quad \Gamma \vdash M' : A; T}{\Gamma \vdash M M' : B; T}\ \to_e
$$

$$
\frac{\Gamma, k : A \to_T \mathbb{\bot} \vdash J : \mathbb{\bot}; T}{\Gamma \vdash \mathcal{C}^{-}(\lambda k.J) : A; T}\ RAA
$$

$$
\frac{\Gamma \vdash J : \mathbb{\bot}; A}{\Gamma \vdash \mathcal{C}^{-}(\lambda \widehat{tp}.\ J) : A; T}\ RAA^{\widehat{tp}}
$$

$$
\frac{\Gamma, k : A \to_T \mathbb{\bot} \vdash M : A; T}{\Gamma, k : A \to_T \mathbb{\bot} \vdash k\ M : \mathbb{\bot}; T}\ \to_e^k \qquad \frac{\Gamma \vdash M : T; T}{\Gamma \vdash \widehat{tp}\ M : \mathbb{\bot}; T}\ \to_e^{\widehat{tp}}
$$

Figure 13. $\Lambda^{\to\ \mathit{eff}_{eq}}_{\mathcal{C}^{-}\widehat{tp}}$: a type-and-effect system of $\lambda^{\to}_{\mathcal{C}^{-}\widehat{tp}}$

can be read as "term M returns a value of type A to its immediate context or a value of type T to its enclosing #." On the $\lambda_{\mathcal{C}\#}$ side, this corresponds to the following rule:

$$
\frac{\Gamma \vdash M : A; A}{\Gamma \vdash \#M : A; T}\ \#
$$

This modification is unsound by itself as it changes the type of $\widehat{tp}$ without taking into account that it is dynamically bound. Simply adding $RAA^{\widehat{tp}}$ to the type system of Figure 7 produces a type system for the static variant of #: the term considered in Example 4 which evaluates to $(\lambda\_.3)$ when # is dynamic and to 3 when # is static would be given the type int.

Therefore, in addition to having the rule $RAA^{\widehat{tp}}$, we also need to modify the system to take into account that $\widehat{tp}$ is a dynamic variable. A possible modification is to repeat what must be done for other dynamically-bound entities like exceptions (Guzmán and Suárez, 1994), and to add an effect annotation on *every* arrow type to pass around the type of $\widehat{tp}$.

Using the system with dynamic effect annotations, the term in Example 4 has type $A \to_B \mathsf{int}$ which is consistent with the value $\lambda\_.3$. Also the term $\lambda x.\ x\ +\ \mathcal{A}$ `"Hello"` discussed in Section 3.3 has the

type $\mathsf{int} \rightarrow_{\mathsf{string}} \mathsf{int}$ with the string constraint correctly moved to when the function is *called*, not when it is defined.

PROPOSITION 9. *Subject reduction: given* $\lambda^{\rightarrow}_{\mathcal{C}\text{-}\widehat{tp}}$ *terms* M *and* N *if* $\Gamma \vdash M : A; T$ *and* $M \rightarrow N$ *then* $\Gamma \vdash N : A; T$.

On the $\lambda_{\mathcal{C}\#}$ side, the corresponding type-and-effect system (which we do not present) addresses the loss of strong normalisation discussed in Example 8. The effect annotations impose the following recursive constraint $T = (\mathsf{int} \rightarrow_T \mathsf{int}) \rightarrow_T \mathsf{int}$. In other words, for the terms to typecheck, we must allow recursive type definitions. This is a situation similar to the one described by Lillibridge (1999) where unchecked exceptions can be used to violate strong normalisation.

More generally, we can prove that under the type-and-effect system $\Lambda^{\rightarrow\ eff_{eq}}_{\mathcal{C}\text{-}\widehat{tp}}$ of Figure 13, typed $\lambda^{\rightarrow}_{\mathcal{C}\text{-}\widehat{tp}}$-terms are strongly normalising. (See Proposition 22.)

INTERMEZZO 10. *In the approach discussed above, different occurrences of the symbol # in a program may have different types. Gunter* et al. *(*1995*) take the (quite natural) position that occurrences of # with different types should have different names. Each name still has a fixed type but that type is not constrained to be the same as the top-level, nor is it constrained to be related to the types of the other names. This proposal shares with* $\Lambda^{\rightarrow\ eff_{eq}}_{\mathcal{C}\text{-}\widehat{tp}}$ *that different occurrences of control delimiters can have different types and that the type of the delimiter must be propagated to the control operator. However, it is closer in design to the system with a fixed answer type as all calls to a delimiter of a given name must have the same type. Moreover, since the type system has no effect annotations, Example 8 still typechecks and loops, and well-typed control operations may refer to non-existent control delimiters.*

6.3. Understanding the Dynamic Annotations

The dynamic annotations we used happen to produce a sound type system but on closer inspection they are not the "right" annotations. In the following analysis, we write $\neg T$ for the type of a continuation variable expecting an argument of type T, deferring for Section 7.1 the question of how to concretely represent this type in our language of types.

It is standard to embed type systems with effects into regular type systems using a monadic transformation. Since our effect annotations have to do with $\widehat{tp}$ which is a dynamic variable, a first guess would be to

use the environment-passing transformation used to explain dynamic scope (Moreau, 1998). At the level of types, the environment-passing transformation (written $(\cdot)^*$) maps $A \to_T B$ to $A^* \wedge \neg T^* \to B^*$, which means that every function is passed the (unique) environment binding as an additional argument. Judgements $\Gamma \vdash A; T$ are mapped to $\Gamma^*, \neg T^* \vdash A^*$ which means that every expression must be typed in the context of its environment. Writing the translation for the pure fragment is easy, but when it comes to $\mathcal{C}^-$, the translation of type rule RAA would produce a continuation whose *input type* is $A^* \wedge \neg T^*$, because continuations also need the environment. But as the rule RAA shows, the input type of the continuation corresponds to the *return type* of the expression that captures it. In other words, expressions that might capture continuations must return both their value *and* the environment variable. Thus, our environment-passing embedding becomes a *store-passing transformation*.

A second interpretation of the annotations would be using exceptions: each delimiter installs an exception handler, and calls to $\widehat{tp}$ throw exceptions to the dynamically-closest handler. Indeed, exceptions can be simulated with ordinary dynamic variables (Moreau, 1998) and with dynamic continuation variables (Gunter et al., 1995). According to the standard monadic interpretation of exceptions (Moggi, 1989), this leads to a transformation mapping $A \to_T B$ to $A^* \to B^* \vee T^*$, which means that every function may return a value or throw an exception to its delimiter. Judgements $\Gamma \vdash A; T$ are mapped to $\Gamma^* \vdash A^* \vee T^*$ and have a similar interpretation. When writing such a translation in a general setting (Thielecke, 2001; Thielecke, 2000; Riecke and Thielecke, 1999), one is faced with a choice: should a use of a control operator capture the current exception handler or not? In our setting, the question is: if a continuation is captured under some #, and later invoked under another #, should calls to $\widehat{tp}$ refer to the first # or the second? It is clear that our semantics requires the second choice. Therefore, our semantics is consistent with the SML/NJ control operators `capture` and `escape`. In combination with exceptions, these control operators can simulate state (Thielecke, 2001, Fig. 12) which means that our embedding also becomes a *store-passing transformation*.

The above discussion suggests the following interpretation: functions $A \to_T B$ are mapped to $A^* \wedge \neg T^* \to B^* \wedge \neg T^*$ and judgements $\Gamma \vdash A; T$ are mapped to $\Gamma^*, \neg T^* \vdash A^* \wedge \neg T^*$. This interpretation is a standard store-passing one, which is consistent with Filinski's observation that *shift* and *reset* can be implemented using continuations and state (1994, 1999). The entire analysis is also consistent with the fact

$$
\begin{array}{rcl}
b \in \mathit{BaseType} & = & \{\ \bot, \ldots\ \} \\
A, B, T, U \in \mathit{TypeExp} & ::= & b \mid A\ _U \!\!\rightarrow_T B \\
\Gamma \in \mathit{Contexts} & ::= & \cdot \mid \Gamma, x : A \mid \Gamma, k : A \rightarrow_T \perp\!\!\!\perp
\end{array}
$$

$$
\frac{}{\Gamma, x : A; T \vdash x : A; T}\ Ax \qquad \frac{\Gamma, x : A; U \vdash M : B; T}{\Gamma; T' \vdash \lambda x.M : A\ _U \!\!\rightarrow_T B; T'}\ \rightarrow_i
$$

$$
\frac{\Gamma; U_1 \vdash M : A\ _{U_2} \!\!\rightarrow_{T_1} B; T_2 \quad \Gamma; T_1 \vdash N : A; U_1}{\Gamma; U_2 \vdash MN : B; T_2}\ \rightarrow_e
$$

$$
\frac{\Gamma, k : A \rightarrow_U \perp\!\!\!\perp \vdash J : \perp\!\!\!\perp; T}{\Gamma; U \vdash \mathcal{C}^{\text{-}}(\lambda k.J) : A; T}\ RAA
$$

$$
\frac{\Gamma \vdash J : \perp\!\!\!\perp; A}{\Gamma; T \vdash \mathcal{C}^{\text{-}}(\lambda \widehat{tp}.\ J) : A; T}\ RAA^{\widehat{tp}}
$$

$$
\frac{\Gamma, k : A \rightarrow_U \perp\!\!\!\perp; U \vdash M : A; T}{\Gamma, k : A \rightarrow_U \perp\!\!\!\perp \vdash k\ M : \perp\!\!\!\perp; T}\ \rightarrow_e^k \qquad \frac{\Gamma; U \vdash M : U; T}{\Gamma \vdash \widehat{tp}\ M : \perp\!\!\!\perp; T}\ \rightarrow_e^{\widehat{tp}}
$$

Figure 14. $\Lambda^{\rightarrow\ \mathit{eff}}_{\mathcal{C}^{\text{-}}\widehat{tp}}$: another type-and-effect system of $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$

that in *classical logic*, the following formulae are all equivalent:

$$
\begin{array}{rll}
 & A \wedge \neg T \rightarrow B & (\widehat{tp}\ \textit{as an environment}) \\
= & A \rightarrow B \vee T & (\widehat{tp}\ \textit{as an exception}) \\
= & A \wedge \neg T \rightarrow B \wedge \neg T & (\widehat{tp}\ \textit{as a state})
\end{array}
$$

6.4. System III: State as Effects

Our type-and-effect system $\Lambda^{\rightarrow\ \mathit{eff}_{eq}}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ for $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ in Figure 13, which was motivated by the understanding of $\widehat{tp}$ as a dynamic variable, is sound but too restrictive. As the analysis in the previous section shows, $\widehat{tp}$ can also be understood as a state parameter and this understanding leads to a different, more expressive, type-and-effect system, which maintains the type of $\widehat{tp}$ *before* and *after* each computation. This generalisation gives the type-and-effect system $\Lambda^{\rightarrow\ \mathit{eff}}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ of Figure 14, which is essentially identical to the one developed by Danvy and Filinski as early as 1989.

In the cases of jumps and continuations, the judgements and types of the new type system are the same as before. When typing terms, the judgements have the form $\Gamma; U \vdash M : A; T$ with T and U describing the top-level continuation before and after the evaluation of the term M, respectively. For terms without $\mathcal{C}^{\text{-}}$, we can show by induction that the two formulae T and U are the same. For terms of the form $\mathcal{C}^{\text{-}}(\lambda k.J)$, the

formula U is the type of the top-level continuation when k is invoked. Implication has two effects $A\ _U \rightarrow_T B$ with T describing the top-level continuation *before* the call and U describing the top-level continuation *after* the call. These changes make the typing of applications sensitive to the order of evaluation of the function and argument: the rule $\rightarrow_e$ assumes the function is evaluated before the argument. The new system is sound.

PROPOSITION 11. *Subject reduction: given* $\lambda^{\rightarrow}_{\mathcal{C}^-\widehat{tp}}$ *terms* M *and* N*, if* $\Gamma; U \vdash M : A; T$ *and* $M \rightarrow N$ *then* $\Gamma; U \vdash N : A; T$.

Let C_{df} be the operation of changing each arrow $A \rightarrow_T B$ into the arrow $A\ _T \rightarrow_T B$ in the single-effect formula C. Let Γ_{df} be the extension of this operation to Γ. The new type-and-effect system generalises the previous system.

PROPOSITION 12. *If* $\Gamma \vdash M : A; T$ *(resp* $\Gamma \vdash J : \perp; T$*) in* $\Lambda^{\rightarrow\ eff_{eq}}_{\mathcal{C}^-\widehat{tp}}$ *then* $\Gamma_{df}; T_{df} \vdash M : A_{df}; T_{df}$ *(resp* $\Gamma_{df} \vdash J : \perp; T_{df}$*) in* $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^-\widehat{tp}}$.

The added expressiveness of the type system is illustrated with the embedding of $\#\ (1 + \mathcal{S}(\lambda c.\ 2\ ==\ c\ 3))$ which evaluates as follows:

$$\begin{array}{ll} & \mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (1\ +\ \mathcal{C}^-(\lambda k.\ \widehat{tp}\ (2\ ==\ \mathcal{C}^-(\lambda\widehat{tp}.\ k\ 3))))) \\ \rightarrow & \mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (\mathcal{C}^-(\lambda k.\ \widehat{tp}\ (2\ ==\ \mathcal{C}^-(\lambda\widehat{tp}.\ k\ (1+3)))))) \\ \rightarrow & \mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (2\ ==\ \mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (1+3)))) \\ \twoheadrightarrow & \mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (2\ ==\ 4)) \rightarrow \mathsf{false} \end{array}$$

This term is rejected by $\Lambda^{\rightarrow\ eff_{eq}}_{\mathcal{C}^-\widehat{tp}}$ but accepted in $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^-\widehat{tp}}$. The first occurrence of k is delimited by the outermost occurrence of $\widehat{tp}$ which is of type int, but when k is invoked, the context is delimited by a type bool.

In general, the new type system is expressive enough to give different types to different occurrences of $\widehat{tp}$. For example, in the following example:

$$\mathcal{C}^-(\lambda\widehat{tp}.\ \widehat{tp}\ (\mathcal{A}^-\ 5 == \mathcal{A}^-\ \texttt{"Hello"}))$$

the visible occurrence of $\widehat{tp}$ and the two implicit occurrences in $\mathcal{A}^-$ have the three different types: bool, int and string respectively.

7. The $\lambda^{\rightarrow -}_{\mathcal{C}\text{-}}$-Calculus

We show that the dual connective of implication, namely subtraction (written $A - B$) arises as a natural type for carrying around the type of the top-level continuation, and formally introduce the $\lambda^{\rightarrow -}_{\mathcal{C}\text{-}}$-calculus.

7.1. Subtraction

Our analysis in Section 6.3 together with our understanding of the system $\Lambda^{\rightarrow\ eff}_{\mathcal{C}\text{-}\widehat{tp}}$ suggest we interpret the more general effect annotations as follows:

$$\begin{array}{rcl}(A\ {}_U{\rightarrow}_T\ B)^* & = & A^* \wedge \neg T^* \rightarrow B^* \wedge \neg U^* \\ (\Gamma; U \vdash A; T)^* & = & \Gamma^*, \neg T^* \vdash A^* \wedge \neg U^*\end{array}$$

This is possible, but it can be refined as we discuss next. First, the type of a continuation variable is really of the form $T^* \rightarrow \perp\!\!\!\perp$. While this type can be injected into $T^* \rightarrow \perp$ as shown in Section 3.3, doing so loses important information since special continuations are injected into the domain of regular functions. Second, the formula $A \wedge \neg T$ is classically the same as $\neg(\neg A \vee T) = \neg(A \rightarrow T)$, *i.e.*, it represents the *dual of a function type* which corresponds to a *subtractive type* $A - T$.

The *subtractive type* has been previously studied by Rauszer (1974) and Crolard (2001), and has been integrated by Curien and Herbelin (2000) in their study of the *duality* between the *producers* of values (which are regular terms) and the *consumers* of values (which are contexts or continuations). In many cases, the dual of a function type appears as a technical formality. Here it arises as the natural type for pairing a term and a continuation and we prefer to use it as a more "abstract" representation of this information. Indeed, in an intuitionistic setting where continuations are not first-class values, it would still make sense to use the type $A - T$ and in that case the type would *not* be equivalent to $A \wedge \neg T$ (Crolard, 2004). Dually, the types $A \rightarrow B$ and $\neg A \vee B$ are classically (but not intuitionistically) equivalent, and we generally prefer to describe functions using the type $A \rightarrow B$ which does not rely on the presence of first-class continuations.

The actual interpretation of effects we use in Figure 15 incorporates both concerns. (The correctness of the interpretation is discussed in Section 8 after we explain the target of the translation in detail.) The interpretation shows that a continuation k has type $A - T \rightarrow \perp\!\!\!\perp$. Understanding $A - T$ as the type $A \wedge \neg T$ means that, as Danvy and Filinski explain, every functional continuation must be given a value and another continuation (called the metacontinuation in their original article (Danvy and Filinski, 1990) and referring to the top-level continuation as we explain). The type $A - T$ is also equivalent to $\neg(\neg T \rightarrow \neg A)$

$$\begin{array}{lcl} b^* & = & b \\ (A \;{}_U{\to}_T\; B)^* & = & (A^* - T^*) \to (B^* - U^*) \\ \\ (\cdot)^* & = & \cdot \\ (\Gamma, x : A)^* & = & \Gamma^*, x : A^* \\ (\Gamma, k : A \to_U \bot\!\!\!\bot)^* & = & \Gamma^*, k : A^* - U^* \to \bot\!\!\!\bot \end{array}$$

Figure 15. Interpreting the effect annotations

$$\frac{\Gamma \vdash \Delta, C^\gamma; A \quad \Gamma, B \vdash \Delta, C^\gamma; C}{\Gamma \vdash \Delta, C^\gamma; A - B}\; -i$$

$$\frac{\Gamma \vdash \Delta; A - B \quad \Gamma, A \vdash \Delta; B}{\Gamma \vdash \Delta; C}\; -_e$$

Figure 16. Subtractive logic in Parigot's style (classical natural deduction)

and with that view, a functional continuation is more like a continuation transformer, which is an idea closely related to Queinnec and Moreau's formalisation of functional continuations as the difference between two continuations (1994).

The embedding of a function type is of the form $(A-T) \to (B-U)$ which captures the idea that the top-level type T needed to execute the function's body should be available when the function is called, and that the new top-level type U is returned as part of the result.

INTERMEZZO 13.

*Figure 16 presents the subtraction rules introduced by Crolard (*2001*) in the style of Parigot's classical natural deduction. In that setting one works with two sets of assumptions:* Γ *and* Δ*, where* Δ *maintains the continuation variables. This suggests that* $\Delta = \{A_1, \cdots, A_n\}$ *could be represented as* $\neg_\bot\Delta$ *defined as* $\{\neg_\bot A_1, \cdots, \neg_\bot A_n\}$*. The subtraction rules would then become:*

$$\frac{\Gamma, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash A \quad \Gamma, B, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash C}{\Gamma, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash A - B}\; -_i$$

$$\frac{\Gamma, \neg_\bot\Delta \vdash A - B \quad \Gamma, A, \neg_\bot\Delta \vdash B}{\Gamma, \neg_\bot\Delta \vdash C}\; -_e$$

*Figure 17 presents the subtraction rules in Prawitz's style natural deduction (*1965*) where all the assumptions are kept on the left-hand side of the sequent. This is closer to our notation and consistent with the rules above which are derivable. Indeed, we have:*

$$\frac{\Gamma, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash A \quad \dfrac{\Gamma, B, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash C}{\Gamma, B, \neg_\bot\Delta, \gamma : \neg_\bot C \vdash \bot\!\!\!\bot}\; \to_e^k}{\Gamma, \neg_\bot\Delta, \gamma : C \to \bot\!\!\!\bot \vdash A - B}\; -_i$$

$$
\begin{array}{rcl}
X & ::= & \{\ \bot, \ldots\ \} \\
A, B & ::= & X \mid A \to B \mid A - B \\
\Gamma & ::= & \cdot \mid \Gamma, A \mid \Gamma, A \to \!\!\!\bot
\end{array}
$$

$$\frac{}{\Gamma, A \vdash A}\ Ax$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_i \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to_e$$

$$\frac{\Gamma, A \to \!\!\!\bot \vdash \bot\!\!\!\bot}{\Gamma \vdash A}\ RAA \qquad \frac{\Gamma, A \to \!\!\!\bot \vdash A}{\Gamma, A \to \!\!\!\bot \vdash \bot\!\!\!\bot}\ \to_e^k$$

$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash \bot\!\!\!\bot}{\Gamma \vdash A - B}\ -i \qquad \frac{\Gamma \vdash A - B \quad \Gamma, A, B \to \!\!\!\bot \vdash C}{\Gamma \vdash C}\ -_e$$

Figure 17. Subtractive logic in Prawitz's style

$$
\begin{array}{rcl}
x, a, v, f \in \mathit{Vars} & & \\
k, tp \in \mathit{KVars} & & \\
M, N \in \mathit{Terms} & ::= & x \mid \lambda x.M \mid MN \mid \mathcal{C}^{-}(\lambda k.J) \mid \\
 & & (M, k\ E) \mid \mathbf{let}\ (x, k) = M\ \mathbf{in}\ N \\
V \in \mathit{Values} & ::= & x \mid \lambda x.M \mid (V, k\ E) \\
J \in \mathit{Jumps} & ::= & k\ M \\
 & & \\
F \in \mathit{ElemCtxt} & ::= & \Box\ M \mid V\ \Box \mid \\
 & & (\Box, k\ E) \mid \mathbf{let}\ (x, k) = \Box\ \mathbf{in}\ M \\
E \in \mathit{EvCtxt} & ::= & \Box \mid E[F]
\end{array}
$$

Figure 18. Syntax of $\lambda^{\to -}_{\mathcal{C}^-}$

$$\dfrac{\Gamma, \neg_{\bot\!\!\!\bot}\Delta \vdash A - B \qquad \dfrac{\dfrac{\Gamma, \neg_{\bot\!\!\!\bot}\Delta, B \to \!\!\!\bot, A \vdash B}{\Gamma, \neg_{\bot\!\!\!\bot}\Delta, B \to \!\!\!\bot, A \vdash \bot\!\!\!\bot}\ \to_e^k \quad \Gamma, \neg_{\bot\!\!\!\bot}\Delta, B \to \!\!\!\bot, A \vdash C}{\Gamma, \neg_{\bot\!\!\!\bot}\Delta, B \to \!\!\!\bot, A \vdash C}\ RAA}{\Gamma, \neg_{\bot\!\!\!\bot}\Delta \vdash C}\ -_e$$

7.2. Syntax, Semantics, and Type System

We formalise the $\lambda^{\to -}_{\mathcal{C}^-}$-calculus by extending the original $\lambda^{\to}_{\mathcal{C}^-\mathsf{tp}}$ with subtraction and removing the special constant `tp` since we are maintaining the top-level continuation using the subtractive type. The resulting calculus is a call-by-value variant of Crolard's extension of the $\lambda\mu$-calculus with subtraction (2001).

The syntax and reduction semantics are given in Figures 18 and 19, respectively. Terms include two new forms that introduce and eliminate

β_v :	$(\lambda x.M)V$	$\rightarrow$	$M[V/x]$
$\mathcal{C}^-_{lift}$:	$F[\mathcal{C}^-(\lambda k.J)]$	$\rightarrow$	$\mathcal{C}^-(\lambda k.J\ [kF/k])$
$\mathcal{C}^-_{elim}$:	$\mathcal{C}^-(\lambda k.k\ M)$	$\rightarrow$	M where $k \notin FV(M)$
$\mathcal{C}^-_{idem}$:	$\mathcal{C}^-(\lambda k.k''\ (\mathcal{C}^-(\lambda k'.J)))$	$\rightarrow$	$\mathcal{C}^-(\lambda k.J\ [k''/k'])$
${Sub_v}^{base}$	**let** $(x,k) = (V, k'\Box)$ **in** M	$\rightarrow$	$M[k'/k; V/x]$
${Sub_v}^{step}$	**let** $(x,k) = (V, k'(E[F]))$ **in** M	$\rightarrow$	**let** $(x,k) = (V, k'E)$ **in** $M\ [kF/k]$

Figure 19. Reductions of call-by-value $\lambda^{\rightarrow -}_{\mathcal{C}^-}$

the subtractive type. Subtractions are introduced by terms of the form $(M, k\ E)$ and eliminated by the form **let** $(x,k) = M$ **in** N. The introduction form uses evaluation contexts (which is motivated and explained below). To simplify the presentation of the evaluation rules, these evaluation contexts are defined using nestings of elementary contexts F ending with a hole. For example, instead of having individual rules which lift $\mathcal{C}^-$ across each elementary context as in the cases of $\mathcal{C}^-_L$ and $\mathcal{C}^-_R$ before, we express all the lifting rules using the single rule $\mathcal{C}^-_{lift}$. The notation $M[kE/k]$ refers to the simultaneous substitution $M[k\ (E[M])/k\ M; k\ (E[E'])/k\ E']$.

We now motivate the presence of the evaluation contexts in values of the subtractive type. From the equivalence $A - T = A \wedge \neg T$, subtractive values could be represented as pairs of a value and a continuation. But it turns out that we need something more than a simple continuation variable k: we need a *jump context* $k\ E$. To develop some intuition about jump contexts, we consider a few examples.

Consider the $\lambda^{\rightarrow}_{\mathcal{C}^-\widehat{tp}}$ term $\mathcal{C}^-(\lambda k.\widehat{tp}\ (\mathit{throw}\ k\ 2))$. After the invocation of the continuation k, control goes back to the current top-level continuation bound to $\widehat{tp}$. If this top-level continuation has value tp, then we can express the example using subtraction by writing $\mathcal{C}^-(\lambda k.\ k\ (2, tp\ \Box))$. In $\lambda^{\rightarrow -}_{\mathcal{C}^-}$, the dynamically-scoped variable $\widehat{tp}$ disappears. Instead, the invocation of k is given the current top-level context which must be invoked next. In this particular case, the jump context is essentially just a continuation variable but things can be more complicated.

As another example, consider the term:

$$\mathcal{S}(\lambda k.\ (k\ 2)\ ==\ 3) \quad \text{or} \quad \mathcal{C}^-(\lambda k.\widehat{tp}\ (\mathcal{C}^-(\lambda \widehat{tp}.k\ 2)\ ==\ 3))$$

After the invocation of continuation k, control returns to the context $\Box\ ==\ 3$ and then to the the top-level continuation bound to $\widehat{tp}$. If this top-level continuation has value tp, then we can express the example

using subtraction by writing:

$$\mathcal{C}^{-}(\lambda k.\ k\ (2, tp\ (\Box\ ==\ 3)))$$

which by $\mathcal{C}^{-}_{elim}$ is equivalent to $(2, tp\ (\Box\ ==\ 3))$. In this case, the jump context is not simply a continuation variable, indicating that the continuation composes with another context before returning to the top-level.

To see this behaviour simulated by the reduction rules, consider a context which eliminates the subtraction and invokes the continuation:

$$\textbf{let}\ (x,k) = (2, tp\ (\Box\ ==\ 3))\ \textbf{in}\ \mathit{throw}\ k\ x$$

Using ${Sub_v}^{step}$, the context $\Box\ ==\ 3$ is lifted:

$$\begin{array}{ll}\textbf{let}\ (x,k) = (2, tp\ (\Box\ ==\ 3)) \rightarrow \textbf{let}\ (x,k) = (2, tp\ \Box)\\ \textbf{in}\ \mathit{throw}\ k\ x & \textbf{in}\ \mathit{throw}\ k\ (x\ ==\ 3)\end{array}$$

and then using ${Sub_v}^{base}$ control is transferred to the top-level:

$$\begin{array}{l}\textbf{let}\ (x,k) = (2, tp\ \Box) \rightarrow \mathit{throw}\ tp\ 2\\ \textbf{in}\ \mathit{throw}\ k\ x\end{array}$$

In general the jump context may include an arbitrary nesting of elementary contexts. For example, had we started the previous example with:

$$\mathcal{S}(\lambda k.(((k\ z)\ N_1)\ N_2)\ N_3)$$

we would have reached:

$$\begin{array}{l}\textbf{let}\ (x,k) = (z, tp\ (((\Box\ N_1)\ N_2)\ N_3))\\ \textbf{in}\ \mathit{throw}\ k\ x\end{array}$$

Instead of lifting out the context $((\Box\ N_1)\ N_2)\ N_3$ at once, our reductions lift the contexts $\Box\ N_1$, $\Box\ N_2$ and $\Box\ N_3$ one by one as shown below:

$$\begin{array}{ll}& \textbf{let}\ (x,k) = (z, tp\ (((\Box\ N_1)\ N_2)\ N_3))\ \textbf{in}\ \mathit{throw}\ k\ x\\ \rightarrow & \textbf{let}\ (x,k) = (z, tp\ ((\Box\ N_2)\ N_3))\ \textbf{in}\ \mathit{throw}\ k\ (x\ N_1)\\ \rightarrow & \textbf{let}\ (x,k) = (z, tp\ (\Box\ N_3))\ \textbf{in}\ \mathit{throw}\ k\ ((x\ N_1)\ N_2)\\ \rightarrow & \textbf{let}\ (x,k) = (z, tp\ \Box)\ \textbf{in}\ \mathit{throw}\ k\ (((x\ N_1)\ N_2)\ N_3)\\ \rightarrow & \mathit{throw}\ tp\ (((z\ N_1)\ N_2)\ N_3)\end{array}$$

The following abbreviations capture some common patterns:

$$\begin{array}{rcll}\lambda(x,k).M & \triangleq & \lambda v.\textbf{let}\ (x,k) = v\ \textbf{in}\ M & \text{(Abbrev. 6)}\\ \textbf{join}\ M & \triangleq & \textbf{let}\ (x,k) = M\ \textbf{in}\ \mathit{throw}\ k\ x & \text{(Abbrev. 7)}\\ \textbf{bind}_k\ M\ \textbf{in}\ N & \triangleq & \textbf{let}\ (f,k) = M\ \textbf{in}\ f\ N & \text{(Abbrev. 8)}\end{array}$$

$$
\begin{array}{rcl}
b \in \mathit{BaseType} & = & \{\ \bot, \ldots\ \} \\
A, B, C, T, U \in \mathit{TypeExp} & ::= & b \mid A \to B \mid A - B \\
\Gamma \in \mathit{Contexts} & ::= & \cdot \mid \Gamma, x : A \mid \Gamma, k : A \to \bot\!\!\!\bot
\end{array}
$$

$$
\frac{}{\Gamma, x : A \vdash x : A}\ Ax \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \to_i
$$

$$
\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash M' : A}{\Gamma \vdash M M' : B}\ \to_e
$$

$$
\frac{\Gamma, k : A \to \bot\!\!\!\bot \vdash J : \bot\!\!\!\bot}{\Gamma \vdash \mathcal{C}^-(\lambda k.J) : A}\ RAA \qquad \frac{\Gamma, k : A \to \bot\!\!\!\bot \vdash M : A}{\Gamma, k : A \to \bot\!\!\!\bot \vdash k\ M : \bot\!\!\!\bot}\ \to_e^k
$$

$$
\frac{\Gamma \vdash M : A \quad \Gamma, \Box : B \vdash k\ E : \bot\!\!\!\bot}{\Gamma \vdash (M, k\ E) : A - B}\ -_i
$$

$$
\frac{\Gamma \vdash M : A - B \qquad \Gamma, x : A, k : B \to \bot\!\!\!\bot \vdash M' : C}{\Gamma \vdash \mathbf{let}\ (x, k) = M\ \mathbf{in}\ M' : C}\ -_e
$$

Figure 20. $\Lambda_{\mathcal{C}^-}^{\to -}$: type system of $\lambda_{\mathcal{C}^-}^{\to -}$

In the first abbreviation v is a fresh variable. The operator **join** abbreviates the common pattern where the elimination form of the subtractive value immediately throws the value to the jump context. The operator **bind** is similar to the monadic operator of the same name. Both M and N are expected to be terms that evaluate to subtractive values with N containing free occurrences of k and f is a fresh variable: the effects of M are performed to produce a subtractive value which is bound to (f, k) and then f is applied to N.

The rule $\mathcal{C}^-_{\mathit{lift}}$ is applicable to **join** M and $\mathbf{bind}_k\ M$ **in** N. Indeed, we have the derived reduction rules:

$$
\begin{array}{ll}
\mathbf{join}\ \mathcal{C}^-(\lambda k.J) & \to\ \mathcal{C}^-(\lambda k.J\ [k(\mathbf{join}\ \Box)/k]) \\
\mathbf{bind}_{k'}\ \mathcal{C}^-(\lambda k.J)\ \mathbf{in}\ N & \to\ \mathcal{C}^-(\lambda k.J\ [k(\mathbf{bind}_{k'}\ \Box\ \mathbf{in}\ N)/k])
\end{array}
$$

The typing rules for the complete language are in Figure 20. The system named $\Lambda_{\mathcal{C}^-}^{\to -}$ is completely standard with no effect annotations and not even a global parameter T. To type the introduction rule of subtraction, we need to type evaluation contexts E with rules essentially identical to the rules for typing terms. We use the notation $\Gamma, \Box : B \vdash k\ E : \bot\!\!\!\bot$ to denote any judgement of the form $\Gamma, x : B \vdash k\ E[x] : \bot\!\!\!\bot$ up to the name x which is a fresh variable.

PROPOSITION 14.

$$
\begin{array}{rcl}
X & = & \{\ \bot, \ldots\ \} \\
A, B, T, U & ::= & X \mid A\ {}_U{\rightarrow}_T\ B \\
\Gamma & ::= & \cdot \mid \Gamma, A \mid \Gamma, A \rightarrow_T \perp\!\!\!\perp
\end{array}
$$

$$
\frac{}{\Gamma, A; T \vdash A; T}\ Ax \qquad \frac{\Gamma, A; U \vdash B; T}{\Gamma; T' \vdash : A\ {}_U{\rightarrow}_T\ B; T'}\ {\rightarrow_i}
$$

$$
\frac{\Gamma; U_1 \vdash A\ {}_{U_2}{\rightarrow}_{T_1}\ B; T_2 \quad \Gamma; T_1 \vdash A; U_1}{\Gamma; U_2 \vdash B; T_2}\ {\rightarrow_e}
$$

$$
\frac{\Gamma, A \rightarrow_U \perp\!\!\!\perp \vdash \perp\!\!\!\perp; T}{\Gamma; U \vdash A; T}\ RAA
$$

$$
\frac{\Gamma \vdash \perp\!\!\!\perp; A}{\Gamma; T \vdash A; T}\ RAA^{\widehat{tp}}
$$

$$
\frac{\Gamma, A \rightarrow_U \perp\!\!\!\perp; U \vdash A; T}{\Gamma, A \rightarrow_U \perp\!\!\!\perp \vdash \perp\!\!\!\perp; T}\ {\rightarrow_e^k} \qquad \frac{\Gamma; U \vdash U; T}{\Gamma \vdash \perp\!\!\!\perp; T}\ {\rightarrow_e^{\widehat{tp}}}
$$

Figure 21. Natural deduction with effects

1. *Subject reduction: Given* $\lambda_{\mathcal{C}^{\text{-}}}^{\rightarrow -}$ *terms* M *and* N *if* $\Gamma \vdash M : A$ *and* $M \rightarrow N$ *then* $\Gamma \vdash N : A$.

2. *Typed* $\lambda_{\mathcal{C}^{\text{-}}}^{\rightarrow -}$ *terms are strongly normalising.*

Proof. Subject reduction is shown using standard proof techniques. The proof of strong normalisation is expanded in detail in Appendix A.

8. Embeddings in $\lambda_{\mathcal{C}^{\text{-}}}^{\rightarrow -}$

We introduce the full embedding of $\lambda_{\mathcal{C}^{\text{-}}\widehat{tp}}^{\rightarrow}$ into $\lambda_{\mathcal{C}^{\text{-}}}^{\rightarrow -}$ and show its correctness. We begin by embedding an *effect logic*, which is essentially $\lambda_{\mathcal{C}^{\text{-}}\widehat{tp}}^{\rightarrow}$ with just types and no terms. This embedding is then used to derive and explain the embedding of terms. Finally we show that the embeddings respect both the typing and semantics of $\lambda_{\mathcal{C}^{\text{-}}\widehat{tp}}^{\rightarrow}$ and hence of $\lambda_{C\#}$.

8.1. Effect Logic and its Embedding

We introduce the embedding of $\lambda_{\mathcal{C}^{\text{-}}\widehat{tp}}^{\rightarrow}$ into $\lambda_{\mathcal{C}^{\text{-}}}^{\rightarrow -}$ by first looking at "pure" type judgements (with no terms) and then fill in the term assignments. Our starting point is the *effect logic* of Figure 21 which gives the type

judgements of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ with no terms. The aim is to embed such judgements in the judgements of the subtractive logic of Figure 17, which is a term-free version of the type system of $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$:

$$\begin{array}{lcl}(\Gamma; B \vdash A; C)^* & = & \Gamma^*, \neg_{\perp\!\!\!\perp} C^* \vdash A^* - B^* \\ (\Gamma \vdash \perp\!\!\!\perp; C)^* & = & \Gamma^*, \neg_{\perp\!\!\!\perp} C^* \vdash \perp\!\!\!\perp\end{array}$$

The embedding extends the embedding of Figure 15 which maps $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ types and contexts into $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$.

When extended with a term assignment, the embeddings become:

$$\begin{array}{lcl}(\Gamma; B \vdash M : A; C)^{tp} & = & \Gamma^*, tp : \neg_{\perp\!\!\!\perp} C^* \vdash M^{tp} : A^* - B^* \\ (\Gamma \vdash J : \perp\!\!\!\perp; C)^{tp} & = & \Gamma^*, tp : \neg_{\perp\!\!\!\perp} C^* \vdash J^{tp} : \perp\!\!\!\perp\end{array}$$

where the value tp of the dynamic top-level continuation is required as a parameter.

We now construct the embedding of each individual axiom and inference rule of the effect logic by induction on the structure of the judgements. For each case, we first present the embedding in the logical setting, and then add the term assignment. The derived embedding of terms is collected in Figure 22. To avoid clutter we will omit the $(\cdot)^*$ translation and irrelevant assumptions.

Case Ax. The embedding of the axiom $A; T \vdash A; T$ requires a proof of $A, \neg_{\perp\!\!\!\perp} T \vdash A - T$ in the subtractive logic:

$$\dfrac{\dfrac{}{A, \neg_{\perp\!\!\!\perp} T \vdash A}\ Ax \quad \dfrac{\dfrac{}{A, \neg_{\perp\!\!\!\perp} T, T \vdash T}\ Ax}{A, \neg_{\perp\!\!\!\perp} T, T \vdash \perp\!\!\!\perp}\ \rightarrow^k_e}{A, \neg_{\perp\!\!\!\perp} T \vdash A - T}\ -_i$$

With a term assignment, we get:

$$\dfrac{\dfrac{}{x : A, tp : \neg_{\perp\!\!\!\perp} T \vdash x : A}\ Ax \quad \dfrac{\dfrac{}{x : A, tp : \neg_{\perp\!\!\!\perp} T, \Box : T \vdash \Box : T}\ Ax}{x : A, tp : \neg_{\perp\!\!\!\perp} T, \Box : T \vdash tp\ \Box : \perp\!\!\!\perp}\ \rightarrow^k_e}{x : A, tp : \neg_{\perp\!\!\!\perp} T \vdash (x, tp\ \Box) : A - T}\ -_i$$

In other words, a variable x is translated into $(x, tp\ \Box)$; the top-level continuation is returned unchanged in this case.

Case $\rightarrow_i$. By induction we have $\neg_{\perp\!\!\!\perp} T, A \vdash B - U$ and we need to prove $\neg_{\perp\!\!\!\perp} T' \vdash (A - T \rightarrow B - U) - T'$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A - T \vdash A - T}\ Ax \quad \dfrac{}{\neg_{\perp\!\!\!\perp} T', \neg_{\perp\!\!\!\perp} T, A \vdash B - U}}{\neg_{\perp\!\!\!\perp} T', A - T \vdash B - U}\ -e}{\neg_{\perp\!\!\!\perp} T' \vdash (A - T) \rightarrow (B - U)}\ \rightarrow_i \quad \dfrac{\dfrac{}{\neg_{\perp\!\!\!\perp} T', T' \vdash T'}\ Ax}{\neg_{\perp\!\!\!\perp} T', T' \vdash \perp\!\!\!\perp}\ \rightarrow^k_e}{\neg_{\perp\!\!\!\perp} T' \vdash (A - T \rightarrow B - U) - T'}\ -_i$$

The sequent $\neg_{\perp\!\!\!\perp} T', \neg_{\perp\!\!\!\perp} T, A \vdash B - U$ already signals that particular care has to be devoted to naming the top-level. Thus, to produce a term assignment for the entire proof, we start with:

$$tp : \neg_{\perp\!\!\!\perp} T, x : A \vdash M^{tp} : B - U$$

and apply a weakening step making sure that the new assumption has a new name:

$$tp' : \neg_{\perp\!\!\!\perp} T, tp : \neg_{\perp\!\!\!\perp} T, x : A \vdash M^{tp} : B - U$$

and then we proceed as follows:

$$\dfrac{\dfrac{\dfrac{\overline{z : A - T \vdash z : A - T} \quad tp' : \neg_{\perp\!\!\!\perp} T', tp : \neg_{\perp\!\!\!\perp} T, x : A \vdash M^{tp} : B - U}{tp' : \neg_{\perp\!\!\!\perp} T', z : A - T \vdash \mathbf{let}\ (x, tp) = z : B - U\ \mathbf{in}\ M^{tp}}}{tp' : \neg_{\perp\!\!\!\perp} T' \vdash \lambda z.\mathbf{let}\ (x, tp) = z : A - T \to B - U\ \mathbf{in}\ M^{tp}}\ \Pi}{tp' : \neg_{\perp\!\!\!\perp} T' \vdash (\lambda(x, tp).M^{tp}, tp'\ \Box) : (A - T \to B - U) - T'}$$

where Π is the proof:

$$\dfrac{\dfrac{}{tp' : \neg_{\perp\!\!\!\perp} T', \Box : T' \vdash \Box : T'}\ Ax}{tp' : \neg_{\perp\!\!\!\perp} T', \Box : T' \vdash tp'\ \Box : \perp\!\!\!\perp}\ \to_e^k$$

Thus the translation of every λ-expression introduces a new binding for tp which is used in the translation of its body. More precisely, every function takes a subtractive value as an argument which specifies the original argument and the top-level continuation resulting from the evaluation of the function and the argument.

Case $\to_e$.

$$\dfrac{\Pi \quad \dfrac{\dfrac{}{(A - T_1) \to (B - U_2) \vdash (A - T_1) \to (B - U_2)} \quad \neg_{\perp\!\!\!\perp} U_1 \vdash A - T_1}{(A - T_1) \to (B - U_2), \neg_{\perp\!\!\!\perp} U_1 \vdash B - U_2}\ \to_e}{\neg_{\perp\!\!\!\perp} T_2 \vdash B - U_2}\ -_e$$

where Π is the proof of :

$$\neg_{\perp\!\!\!\perp} T_2 \vdash ((A - T_1) \to (B - U_2)) - U_1$$

For the term assignment, let Γ abbreviate the context $\{tp : \neg_{\perp\!\!\!\perp} T_2, tp' : \neg_{\perp\!\!\!\perp} U_1, f : (A - T_1) \to (B - U_2)\}$, then we have:

$$\dfrac{\Pi \quad \dfrac{\dfrac{}{\Gamma \vdash f : (A - T_1) \to (B - U_2)}\ Ax \quad \Gamma \vdash N^{tp'} : A - T_1}{\Gamma \vdash f\ N^{tp'} : B - U_2}\ \to_e}{tp : \neg_{\perp\!\!\!\perp} T_2 \vdash \mathbf{bind}_{tp'}\ M^{tp}\ \mathbf{in}\ N^{tp'} : B - U_2}\ -_e$$

where Π is the judgement:

$$tp : \neg_{\perp\!\!\!\perp} T_2 \vdash M^{tp} : ((A - T_1) \to (B - U_2)) - U_1$$

The references to the top-level in $N^{tp'}$ are bound by the $\mathbf{bind}_{tp'}$, while the occurrences of the top-level in M^{tp} are free. These relationships mimic the facts that the actual binding for a call to $\widehat{tp}$ in N is the one active when M returns its result, whereas the actual binding for a call to $\widehat{tp}$ in M is the one at the time of evaluating MN.

In the case when M is a value V, the optimised embedding corresponds to the elimination of the following detour:

$$\dfrac{\dfrac{\neg_{\perp\!\!\!\perp} T_2 \vdash ((A - T_1) \to (B - U_2)) \quad \neg_{\perp\!\!\!\perp} T_2, U_1 \vdash \perp\!\!\!\perp}{\neg_{\perp\!\!\!\perp} T_2 \vdash ((A - T_1) \to (B - U_2)) - U_1}\ -_i \quad \Pi}{\neg T_2 \vdash B - U_2}\ -_e$$

where Π is

$$((A - T_1) \to (B - U_2)) - U_1, \neg_{\perp\!\!\!\perp} U_1 \vdash B - U_2$$

Case RAA.

$$\dfrac{k : \neg_{\perp\!\!\!\perp}(A - U), tp : \neg_{\perp\!\!\!\perp} T \vdash J^{tp} : \perp\!\!\!\perp}{tp : \neg_{\perp\!\!\!\perp} T \vdash \mathcal{C}^{-}(\lambda k.\ J^{tp}) : A - U}\ RAA$$

Because all the right-hand sides of the embedding return terms of the subtractive type, a continuation k in $\lambda^{\to}_{\mathcal{C}^{-}\widehat{tp}}$ is translated to a continuation which expects a subtractive value as its argument.

Case $RAA^{\widehat{tp}}$.

$$\dfrac{\dfrac{\neg_{\perp\!\!\!\perp} A, \neg_{\perp\!\!\!\perp} T \vdash \perp\!\!\!\perp}{\neg_{\perp\!\!\!\perp} T \vdash A}\ RAA \quad \dfrac{\dfrac{}{\neg_{\perp\!\!\!\perp} T, T \vdash T}\ Ax}{\neg_{\perp\!\!\!\perp} T, T \vdash \perp\!\!\!\perp}\ \to^{k}_{e}}{\neg_{\perp\!\!\!\perp} T \vdash A - T}\ -_i$$

For the term assignment, we get:

$$\dfrac{\dfrac{tp : \neg_{\perp\!\!\!\perp} A, tp' : \neg_{\perp\!\!\!\perp} T \vdash J^{tp} : \perp\!\!\!\perp}{tp' : \neg_{\perp\!\!\!\perp} T \vdash \mathcal{C}^{-}(\lambda tp.J^{tp}) : A}\ RAA \quad \dfrac{\dfrac{}{tp' : \neg_{\perp\!\!\!\perp} T, \Box : T \vdash \Box : T}\ Ax}{tp' : \neg_{\perp\!\!\!\perp} T, \Box : T \vdash tp'\ \Box : \perp\!\!\!\perp}\ \to^{k}_{e}}{tp' : \neg_{\perp\!\!\!\perp} T \vdash (\mathcal{C}^{-}(\lambda tp.J^{tp}), tp'\ \Box) : A - T}\ -_i$$

The embedding returns a subtractive term with the current top-level continuation tp' as the jump context and a computation which introduces a fresh tp thus simulating the rebinding of the top-level

$$
\begin{array}{rcl}
x^+ & = & x \\
(\lambda x.M)^+ & = & \lambda(x, tp).M^{tp} \\
 & & \\
V^{tp} & = & (V^+, tp\ \Box) \\
(VN)^{tp} & = & V^+\ N^{tp} \\
(MN)^{tp} & = & \mathbf{bind}_{tp}\ M^{tp}\ \mathbf{in}\ N^{tp} \quad (M \text{ not a value}) \\
(\mathcal{C}^{\text{-}}(\lambda k.J))^{tp} & = & \mathcal{C}^{\text{-}}(\lambda k.J^{tp}) \\
(\mathcal{C}^{\text{-}}(\lambda \widehat{tp}.J))^{tp} & = & (\mathcal{C}^{\text{-}}(\lambda tp.J^{tp}), tp\ \Box) \\
 & & \\
(k\ M)^{tp} & = & k\ M^{tp} \\
(\widehat{tp}\ M)^{tp} & = & tp\ (\mathbf{join}\ M^{tp})
\end{array}
$$

Figure 22. Embedding of $\lambda^{\rightarrow}_{\mathcal{C}^{\text{-}}\widehat{tp}}$ terms in $\lambda^{\rightarrow -}_{\mathcal{C}^{\text{-}}}$ terms

continuation in J. Note that tp does *not* expect a subtractive value as its argument. Thus, the argument passed to a top-level jump is evaluated to produce a value and a jump context, which are consumed before returning to tp.

Case $\rightarrow^k_e$.

$$\frac{k : \neg_{\perp\!\!\!\perp}(A - U), tp : \neg_{\perp\!\!\!\perp} T \vdash M^{tp} : A - U}{k : \neg_{\perp\!\!\!\perp}(A - U), tp : \neg_{\perp\!\!\!\perp} T \vdash k\ M^{tp} : \perp\!\!\!\perp}\ \rightarrow^k_e$$

Case $\rightarrow^{\widehat{tp}}_e$. The logical embedding is quite interesting:

$$\dfrac{\dfrac{\neg_{\perp\!\!\!\perp} T \vdash U - U \qquad \dfrac{\dfrac{\dfrac{}{\neg_{\perp\!\!\!\perp} T, U, \neg_{\perp\!\!\!\perp} U \vdash U}\ Ax}{\neg_{\perp\!\!\!\perp} T, U, \neg_{\perp\!\!\!\perp} U \vdash \perp\!\!\!\perp}\ \rightarrow^k_e}{\neg_{\perp\!\!\!\perp} T, U, \neg_{\perp\!\!\!\perp} U \vdash T}\ RAA}{\neg_{\perp\!\!\!\perp} T \vdash T}\ -_e}{\neg_{\perp\!\!\!\perp} T \vdash \perp\!\!\!\perp}\ \rightarrow^k_e$$

The above proof leads to the following term assignment:

$$\dfrac{\dfrac{tp : \neg_{\perp\!\!\!\perp} T \vdash M^{tp} : U - U \qquad \dfrac{\dfrac{\dfrac{}{tp : \neg_{\perp\!\!\!\perp} T, x : U, k : \neg_{\perp\!\!\!\perp} U \vdash x : U}\ Ax}{tp : \neg_{\perp\!\!\!\perp} T, x : U, k : \neg_{\perp\!\!\!\perp} U \vdash k\ x : \perp\!\!\!\perp}\ \rightarrow^k_e}{tp : \neg_{\perp\!\!\!\perp} T, x : U, k : \neg_{\perp\!\!\!\perp} U \vdash \mathit{throw}\ k\ x : T}\ RAA}{tp : \neg_{\perp\!\!\!\perp} T \vdash \mathbf{join}\ M^{tp} : T}\ -_e}{tp : \neg_{\perp\!\!\!\perp} T \vdash tp\ (\mathbf{join}\ M^{tp}) : \perp\!\!\!\perp}\ \rightarrow^k_e$$

Figure 22 summarises the embedding of terms and jumps derived from the term assignments. The translation in the figure includes a special translation $(\cdot)^+$ on values with the following property. The

$$
\begin{array}{lcl}
x^{+} & = & x \\
(\lambda x.M)^{+} & = & \lambda(x, tp).M^{tp} \\
 & & \\
V^{tp} & = & (V^{+}, tp\ \Box) \\
(VN)^{tp} & = & V^{+}\ N^{tp} \\
(MN)^{tp} & = & \mathbf{bind}_{tp}\ M^{tp}\ \mathbf{in}\ N^{tp} \qquad (M \text{ not a value}) \\
(\mathcal{C}\ M)^{tp} & = & \mathcal{C}^{-}(\lambda k.\ tp\ (\mathbf{join} \\
 & & \qquad (\mathbf{bind}_{tp'}\ M^{tp} \\
 & & \qquad\ \mathbf{in}\ (\lambda(x, tp).throw\ k\ (x, tp\ \Box) \\
 & & \qquad\quad , tp'\ \Box)))) \\
(\#\ M)^{tp} & = & (\mathcal{C}^{-}(\lambda tp.\ tp\ (\mathbf{join}\ M^{tp})), tp\ \Box)
\end{array}
$$

Figure 23. Embedding of $\lambda_{\mathcal{C}\#}$ terms in $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ terms

translation of a value has no free occurrences of the continuation variable used in the embedding. This allows us to infer for example, that in the special case where we jump to the top-level with a value, we have $(\widehat{tp}\ V)^{tp} \twoheadrightarrow tp\ V^{+}$.

8.2. Induced Embedding of $\lambda_{\mathcal{C}\#}$ in $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$

Combining the embedding from $\lambda_{\mathcal{C}\#}$ to $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ (Figure 11) with the embedding we just defined, produces a direct embedding from $\lambda_{\mathcal{C}\#}$ into $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ shown in Figure 23. The translation of $\mathcal{C}\ M$ is the most complicated: the continuation k is first captured and then we jump to the top-level with a computation that first evaluates M. Once M produces a value f and perhaps a new top-level continuation tp', we apply f to a subtractive value consisting of the reified continuation and tp'. When the reified continuation is invoked on an argument P, it is given the value and top-level continuation resulting from the evaluation of P which it passes to k aborting its context.

Given the equivalence of $\mathcal{C}$ and $\mathcal{S}$, it is easy to calculate that the embedding of $\mathcal{S}$ differs only in the details of the reified continuation which becomes $\lambda(x, tp).\mathcal{C}^{-}(\lambda tp''.k\ (x, tp''\ (\Box, tp\ \Box)))$. Thus, instead of ignoring the top-level continuation tp'' at the call site like in the case of $\mathcal{C}$, this continuation composes the top-level continuations, first returning to tp and then returning to tp''.

EXAMPLE 15. If the top-level continuation is tp, the embedding of the $\lambda_{\mathcal{C}\#}$ term $(y\ (\#\ x))$ (which corresponds to $(y\ (\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ x)))$ in

$\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$) is:

$$\begin{aligned}(y\ (\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ x)))^{tp} &\rightarrow y\ ((\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ x))^{tp}) \\ &\rightarrow y\ ((\mathcal{C}^{-}(\lambda tp'.tp'\ x), tp\ \Box))\end{aligned}$$

The current top-level continuation is "saved" using the subtraction introduction and a new top-level continuation is used for receiving the value of x.

8.3. Soundness of the embedding of $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ into $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$

For tp a fresh continuation variable, the embedding is sound in the sense that it maps judgements of $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^{-}\widehat{tp}}$ (Figure 14) to judgements of $\Lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ (Figure 20).

PROPOSITION 16.

(i) If $\Gamma; U \vdash M : A; T$ *in* $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^{-}\widehat{tp}}$ *then* $(\Gamma; U \vdash M : A; T)^{tp}$ *in* $\Lambda^{\rightarrow -}_{\mathcal{C}^{-}}$,

(ii) If $\Gamma \vdash J :\perp\!\!\!\perp; T$ *in* $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^{-}\widehat{tp}}$ *then* $(\Gamma \vdash J :\perp\!\!\!\perp; T)^{tp}$ *in* $\Lambda^{\rightarrow -}_{\mathcal{C}^{-}}$.

The soundness argument also extends to the type system $\Lambda^{\rightarrow\ fixed}_{\mathcal{C}^{-}\widehat{tp}}$ (Definition 7) for $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$. Let A_T be the operation of adding twice the same *atomic* effect T on the occurrences of $\rightarrow$ in the effect-free formula A. Let Γ_T be the extension of this operation to an effect-free Γ.

PROPOSITION 17. *Let* T *be atomic. If* $\Gamma \vdash M : A; T$ *in* $\Lambda^{\rightarrow\ fixed}_{\mathcal{C}^{-}\widehat{tp}}$ *then* $\Gamma_T; T \vdash M : A_T; T$ *in* $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^{-}\widehat{tp}}$. *Hence* $(\Gamma_T; T \vdash M : A_T; T)^{tp}$ *in* $\Lambda^{\rightarrow -}_{\mathcal{C}^{-}}$.

The embedding is also sound with respect to the semantics. More precisely, each reduction step in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ maps to a reduction sequence of at least one step (written $\twoheadrightarrow^{\geq 1}$) in $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$, *up to* some quotient relation $\approx$.

We define the equivalence relation $M \approx M'$ as the smallest compatible relation which satisfies $k\ (E[\mathbf{join}\ N]) \approx k'\ (E'[\mathbf{join}\ N'])$ as soon as $N \approx N'$, whatever k, k', E and E' are. The relation easily extends to an equivalence relation between contexts.

The reason for introducing $\approx$ is that the translation of $\widehat{tp}\ M$ has some defect: since $\mathbf{join}\ M^{tp}$ is a term, an extra tp has to be inserted to get a jump as expected. Though this jump is never executed (it is eventually thrown away as M^{tp} reduces to a value), it is not compatible with the embedding by reduction, hence the need for reasoning up to

the name of this extra continuation variable (and up to its possible future instances).

We are now ready to state the soundness of the embedding with respect to the semantics.

PROPOSITION 18. *Let tp be a fresh continuation variable not occurring in M. If $M \to N$ in $\lambda^{\to}_{\mathcal{C}^-\widehat{tp}}$ and $M' \approx M^{tp}$ then there exists some $N' \approx N^{tp}$ such that $M' \twoheadrightarrow^{\geq 1} N'$ in $\lambda^{\to -}_{\mathcal{C}^-}$.*

To prove this proposition, we need an auxiliary lemma which provides the required properties of the embedding with relation to substitution and evaluation contexts.

LEMMA 19. *To embed evaluation contexts, we let $\Box^{tp} = \Box$. Then we can verify the following properties:*

(i) $M^{tp}[V^+/x] = (M[V/x])^{tp}$

(ii) $J^{tp}[(k\ E)^{tp}/k] \twoheadrightarrow (J[k\ E/k])^{tp}$

(iii) $J^{tp}[(\widehat{tp}\ \Box)^{tp}/k] \approx (\hat{J}[\widehat{tp}\ \Box/\hat{k}])^{tp}$

Proof. The proof proceeds by induction on M or J. For the cases (i) and (ii), we first remark that if E is not of the form $E'[\Box\ M']$ or M is not a value, then we actually have:

$$(k\ (E[M]))^{tp} \ = \ (k\ E)^{tp}[M^{tp}]$$

while if $E = E'[\Box\ M']$ then:

$$\begin{array}{rcl}(k\ E)^{tp}[V^{tp}] & = & (k\ E')^{tp}[V^{tp}\ M'^{tp}] \\ & \to & (k\ E')^{tp}[(V\ M')^{tp}] \ = \ (k\ (E[V]))^{tp}\end{array}$$

We then observe that the translated jump contexts and values have no free occurrences of tp and that whether a term M is a value or not when translating $(M\ N)$ is preserved by substitution of values.

For the case (iii), $(\widehat{tp}\ M)^{tp}$ has a free occurrence of tp. This is why we have to reason up to $\approx$.

We also need the compatibility of $\approx$ with substitution, which is direct by induction.

LEMMA 20. *If $M \approx M'$, $V \approx V'$ and $E \approx E'$ then $M[V/x] \approx M'[V'/x]$ and $M[kE/k] \approx M'[kE'/k]$.*

Proof of Proposition 18. For the rules β_v, $\mathcal{C}^-_R$, $\mathcal{C}^-_L$ and $\mathcal{C}^-_{elim}$, we have the following properties from which the result follows by lemma 20 and compatibility of $\approx$ with subterms.

$$\begin{array}{ll}
\beta_v: & ((\lambda x.M)V)^{tp} \;=\; (\lambda(x,tp').M^{tp'})(V^+, tp\ \Box) \\
\twoheadrightarrow^{\geq 1} & M^{tp}[V^+/x] = M[V/x]^{tp} \text{ (by Proposition 19(i))}
\end{array}$$

$$\begin{array}{ll}
\mathcal{C}^-_L: & (\mathcal{C}^-(\lambda k'.J)N)^{tp} \;=\; \mathbf{bind}_{tp'}\ \mathcal{C}^-(\lambda k'.J^{tp})\ \mathbf{in}\ N^{tp'} \\
\to & \mathcal{C}^-(\lambda k'.J^{tp}[k'\ (\mathbf{bind}_{tp'}\ \Box\ \mathbf{in}\ N^{tp'})/k']) \\
= & \mathcal{C}^-(\lambda k'.J^{tp}[(k'\ (\Box\ N))^{tp}/k']) \\
\twoheadrightarrow & (\mathcal{C}^-(\lambda k'.J[k'\ (\Box\ N)/k']))^{tp} \text{ (by Proposition 19(ii))}
\end{array}$$

$$\begin{array}{ll}
\mathcal{C}^-_R: & (V\ \mathcal{C}^-(\lambda k'.J))^{tp} \;=\; V^+\ \mathcal{C}^-(\lambda k'.J^{tp}) \\
\to & \mathcal{C}^-(\lambda k'.J^{tp}[k'\ (V^+\ \Box)/k']) \\
= & \mathcal{C}^-(\lambda k'.J^{tp}[(k'\ (V\ \Box))^{tp}/k']) \\
\twoheadrightarrow & (\mathcal{C}^-(\lambda k'.J[k'\ (V\ \Box)/k']))^{tp} \text{ (by Proposition 19(ii))}
\end{array}$$

$$\begin{array}{ll}
\mathcal{C}^-_{elim}: & (\mathcal{C}^-(\lambda k.k\ M))^{tp} = \mathcal{C}^-(\lambda k.k\ M^{tp}) \\
\to & M^{tp} \quad \text{as } k \notin FV(M)
\end{array}$$

For $\mathcal{C}^-_{elim'}$, let $M' \approx (\mathcal{C}^-(\lambda\widehat{tp}.\widehat{tp}\ V))^{tp}$. Since the right-hand side is $(\mathcal{C}^-(\lambda tp.tp\ (\mathbf{join}\ (V^+, tp\ \Box))), tp\ \Box)$, by definition of $\approx$, there is k, E and $V' \approx V^+$ such that:

$$\begin{array}{ll}
\mathcal{C}^-_{elim'}: & M' = (\mathcal{C}^-(\lambda tp.k\ (E[\mathbf{join}\ (V', tp\ \Box)])), tp\ \Box) \\
\twoheadrightarrow^{\geq 1} & (V', tp\ \Box) \\
\approx & (V^+, tp\ \Box) = V^{tp}
\end{array}$$

The case of $\mathcal{C}^-_{idem'}$ and $\mathcal{C}^-_{idem'_{\widehat{tp}}}$ are similar. Here again, the generalisation up to $\approx$ comes from lemma 20 and compatibility of $\approx$ with the subterm relation:

$$\begin{array}{ll}
\mathcal{C}^-_{idem'}: & \mathcal{C}^-(\lambda k.k''\ \mathcal{C}^-(\lambda k'.J))^{tp} = \mathcal{C}^-(\lambda k.k''\ \mathcal{C}^-(\lambda k'.J^{tp})) \\
\twoheadrightarrow^{\geq 1} & \mathcal{C}^-(\lambda k.J^{tp}[k''/k']) = \mathcal{C}^-(\lambda k.J[k''/k'])^{tp}
\end{array}$$

$$\begin{array}{ll}
\mathcal{C}^-_{idem'_{\widehat{tp}}}: & \mathcal{C}^-(\lambda\widehat{tp}.k\ \mathcal{C}^-(\lambda k'.J))^{tp} = (\mathcal{C}^-(\lambda tp'.k\ \mathcal{C}^-(\lambda k'.J^{tp'})), tp\ \Box) \\
\twoheadrightarrow^{\geq 1} & (\mathcal{C}^-(\lambda tp'.J^{tp'}[k/k']), tp\ \Box) = \mathcal{C}^-(\lambda\widehat{tp}.J[k/k'])^{tp}
\end{array}$$

The case of $\mathcal{C}^{-}_{idem}$ and $\mathcal{C}^{-}_{idem_{\widehat{tp}}}$ are the only cases which really require reasoning up to $\approx$. For $\mathcal{C}^{-}_{idem}$, let $M \approx \mathcal{C}^{-}(\lambda k.\widehat{tp}\ (\mathcal{C}^{-}(\lambda k'.J)))^{tp}$. Since the right-hand side is $\mathcal{C}^{-}(\lambda k.tp\ (\mathbf{join}\ \mathcal{C}^{-}(\lambda k'.J^{tp})))$, by definition of $\approx$, there is k'', E and $J' \approx J^{tp}$ such that:

$$\begin{array}{ll}
\mathcal{C}^{-}_{idem}: & M = \mathcal{C}^{-}(\lambda k.k''\ (E[\mathbf{join}\ \mathcal{C}^{-}(\lambda k'.J')])) \\
\twoheadrightarrow^{\geq 1} & \mathcal{C}^{-}(\lambda k.J'[k''\ (E[\mathbf{join}\ \Box])/k']) \\
\approx & \mathcal{C}^{-}(\lambda k.J^{tp}[tp\ (\mathbf{join}\ \Box)/k']) \\
= & \mathcal{C}^{-}(\lambda k.J^{tp}[(\widehat{tp}\ \Box)^{tp}/k']) \\
\approx & \mathcal{C}^{-}(\lambda k.\hat{J}[\widehat{tp}/k'])^{tp} \text{ (by Proposition 19(iii))}
\end{array}$$

Similarly, for $\mathcal{C}^{-}_{idem_{\widehat{tp}}}$, let $M \approx \mathcal{C}^{-}(\lambda\widehat{tp}.\widehat{tp}\ (\mathcal{C}^{-}(\lambda k'.J)))^{tp}$. Since the right-hand side is $(\mathcal{C}^{-}(\lambda tp'.tp'\ \ (\mathbf{join}\ \mathcal{C}^{-}(\lambda k'.J^{tp}))), tp)$, by definition of $\approx$, there is k'', E and $J' \approx J^{tp'}$ such that:

$$\begin{array}{ll}
\mathcal{C}^{-}_{idem_{\widehat{tp}}}: & M = (\mathcal{C}^{-}(\lambda tp'.k''\ (E[\mathbf{join}\ \mathcal{C}^{-}(\lambda k'.J')])), tp) \\
\twoheadrightarrow^{\geq 1} & (\mathcal{C}^{-}(\lambda tp'.J'[k''\ (E[\mathbf{join}\ \Box])/k']), tp) \\
\approx & (\mathcal{C}^{-}(\lambda tp'.J^{tp'}[tp'\ (\mathbf{join}\ \Box)/k']), tp) \\
= & (\mathcal{C}^{-}(\lambda tp'.J^{tp'}[(\widehat{tp}\ \Box)^{tp'}/k']), tp) \\
\approx & \mathcal{C}^{-}(\lambda\widehat{tp}.\hat{J}[\widehat{tp}/k'])^{tp} \text{ (by Proposition 19(iii))}
\end{array}$$

Having established the soundness of each individual reduction, it remains to show that the property still holds when reducing a subterm. This is done by induction and all cases are trivial except for the case of $(M\ N)$ when M reduces to a value V without being a value itself. We have $(M\ N)^{tp} = \mathbf{bind}_{tp}\ M^{tp}\ \mathbf{in}\ N^{tp}$. By the induction hypothesis, it reduces in at least one step to $\mathbf{bind}_{tp}\ V^{tp}\ \mathbf{in}\ N^{tp}$ so that it remains to check that the latter reduces to $(V\ N)^{tp}$ which is the case.

As an emphasis of how the translation works, the following example shows how to capture dynamic substitution in $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$.

EXAMPLE 21. Consider the example used in Section 5.2 to illustrate the capture of $\widehat{tp}$ during substitution:

$$\begin{array}{ll}
 & (\lambda x.(\lambda y.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y))))\ (\lambda_.\mathcal{C}^{-}(\lambda_.\widehat{tp}\ y)) \\
\rightarrow & \lambda y'.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ ((\lambda_.\mathcal{C}^{-}(\lambda_.\ \widehat{tp}\ y))\ y'))
\end{array}$$

The embedding of the left-hand side is as follows:

$$
\begin{array}{ll}
((\lambda x.(\lambda y.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y))))\ (\lambda\_.\mathcal{C}^{-}(\lambda\_.\widehat{tp}\ y)))^{tp} & = \\
(\lambda x.(\lambda y.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y))))^{+}\ (\lambda\_.\mathcal{C}^{-}(\lambda\_.\widehat{tp}\ y))^{tp} & = \\
(\lambda(x,tp_1).(\lambda y.\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y)))^{tp_1})\ ((\lambda\_.\mathcal{C}^{-}(\lambda\_.\widehat{tp}\ y))^{+},tp\ \Box)) & = \\
((\lambda(x,tp_1).((\lambda(y,tp_3).\mathcal{C}^{-}(\lambda\widehat{tp}.\ \widehat{tp}\ (x\ y)))^{tp_3}),tp_1\ \Box)) & \\
\quad((\lambda\_.\mathcal{C}^{-}(\lambda\_.\widehat{tp}\ y))^{+},tp\ \Box)) & = \\
((\lambda(x,tp_1).(\lambda(y,tp_3).(\mathcal{C}^{-}(\lambda tp_4.tp_4\ (\mathbf{join}\ (x\ (y,tp_4)))),tp_3\ \Box),tp_1\ \Box)) & \\
\quad((\lambda(\_,tp_2).\mathcal{C}^{-}(\lambda\_.tp_2\ (\mathbf{join}\ (y,tp_2\ \Box)))),tp\ \Box)) &
\end{array}
$$

This term reduces to:

$$
(\lambda(y',tp_3).(\mathcal{C}^{-}(\lambda tp_4.tp_4\ (\mathbf{join}\ ((\lambda(\_,tp_2).\mathcal{C}^{-}(\lambda\_.tp_2\ (\mathbf{join}\ (y,tp_2\ \Box))))\ (y',tp_4)))),tp_3\ \Box),tp\ \Box)
$$

which is the embedding of the right-hand side.

The correspondence with $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ does *not* imply however that there is any relationship between $\lambda_{c\#}$ reductions and $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ reductions under the combined embedding of Figure 23 because the intermediate Proposition 6 only provides soundness up to operational equivalence.

The theory $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ is richer than $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$, so that the translation is not complete. Indeed, the dynamically-bound continuation $\widehat{tp}$ is turned into an ordinary statically-bound continuation: the limitations we imposed on lifting $\mathcal{C}^{-}(\lambda\widehat{tp}\ldots)$ in Section 5.2 can no longer be enforced. For instance, the term in Example 15 can be further reduced as follows:

$$
\begin{array}{rl}
y\ ((\mathcal{C}^{-}(tp'.tp'\ x),tp\ \Box)) & \overset{\mathcal{C}^{-}_{lift}}{\rightarrow}\ y\ (\mathcal{C}^{-}(tp'.tp'\ (x,tp\ \Box))) \\
& \overset{\mathcal{C}^{-}_{lift}}{\rightarrow}\ \mathcal{C}^{-}(tp'.tp'\ (y\ (x,tp\ \Box)))
\end{array}
$$

The *lift* reductions move the # while maintaining the proper references to the top-level continuation. This move has no counterpart in either $\lambda_{c\#}$ or $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$. In $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ such a lifting would cause the dynamically-bound top-level continuation $\widehat{tp}$ to be incorrectly captured; in $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$, the top-level continuation is statically-bound so that it is enough to rely on α-conversion to have a safe lifting rule for #. Pulling the lifting rule for # back from $\lambda^{\rightarrow -}_{\mathcal{C}^{-}}$ to $\lambda^{\rightarrow}_{\mathcal{C}^{-}\widehat{tp}}$ is non trivial, if ever possible.

Collecting the previous result with Propositions 12, 14, 16 and 18, we establish strong normalisation for the relevant systems.

PROPOSITION 22.

(i) If $\Gamma; U \vdash M : A; T$ *in* $\Lambda^{\to\ eff}_{\mathcal{C}\text{-}\widehat{tp}}$ *or* $\Gamma \vdash M : A; T$ *in* $\Lambda^{\to\ eff_{eq}}_{\mathcal{C}\text{-}\widehat{tp}}$ *then* M *is strongly normalising.*

(ii) If $\Gamma \vdash M : A; T$ *in* $\Lambda^{\to\ fixed}_{\mathcal{C}\text{-}\widehat{tp}}$ *and* T *is atomic then* M *is strongly normalising.*

Especially, the last item says that we cannot enforce strong normalisation as soon as the top-level type is not atomic. Indeed, if T is non atomic then we have to add the annotation T on the arrows of T itself. This requires a definition by fixpoint and the resulting recursive type can be used to type a fixpoint combinator as shown in Example 8.

9. CPS Semantics

The introduction of the control operators *shift* and *reset* was motivated by their CPS semantics. To complete our investigation of these operators, we therefore relate our results to their standard CPS semantics (Danvy and Filinski, 1989). In particular, we present a CPS translation for $\lambda^{\to -}_{\mathcal{C}\text{-}}$ and show that composing the embeddings of *shift* and *reset* into $\lambda^{\to -}_{\mathcal{C}\text{-}}$ with this CPS transformation produces the standard CPS transformation for *shift* and *reset*.

9.1. Target CPS Calculus

Before presenting the CPS translation from $\lambda^{\to -}_{\mathcal{C}\text{-}}$ we first introduce the target language of the translation. This target language $\lambda^{\wedge}$ is an ordinary *call-by-name* λ-calculus with pairs. In this calculus, we also use the following abbreviation which is similar to Abbrev. 6:

$$\lambda(x, y).M \triangleq \lambda z.\mathbf{let}\ (x, y) = z\ \mathbf{in}\ M \qquad \text{(Abbrev. 9)}$$

The semantics of $\lambda^{\wedge}$ is given using the following three rules:

$$\begin{array}{lllll} \beta: & (\lambda x.M)N & \to & M[N/x] & \\ \eta: & \lambda x.M\ x & \to & M & x \notin FV(M) \\ \wedge: & \mathbf{let}\ (x, y) = (M, N)\ \mathbf{in}\ M' & \to & M'[N/y; M/x] & \end{array}$$

The type system of $\lambda^{\wedge}$, named $\Lambda^{\wedge}$, is an ordinary natural deduction system equipped with *tensorial conjunction*. This is a conjunction whose elimination rule extracts both components of the pair simultaneously, similar to what happens for the tensor product of linear logic. The tensorial conjunction will be used to model subtractions in $\lambda^{\to -}_{\mathcal{C}\text{-}}$; its introduction and elimination rules are as follows:

$$
\begin{array}{lcl}
b^* & = & b \\
(A \to B)^* & = & \neg B^* \to \neg A^* \\
(A - B)^* & = & \neg B^* \wedge A^* \\
 & & \\
(A \to \bot\!\!\!\bot)^* & = & \neg A^* \\
 & & \\
(\cdot)^* & = & \cdot \\
(\Gamma, x : A)^* & = & \Gamma^*, x : A^* \\
 & & \\
(\Gamma \vdash M : A)^k & = & \Gamma^*, k : \neg A^* \vdash [\![M]\!]k : \bot \\
(\Gamma \vdash J :\bot\!\!\!\bot)^* & = & \Gamma^* \vdash [\![J]\!] : \bot
\end{array}
$$

Figure 24. Fischer-style call-by-value $\neg\neg$-translation on $\Lambda^{\to -}_{c^-}$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \wedge B} \wedge_i$$

$$\frac{\Gamma \vdash M : A \wedge B \qquad \Gamma, x : A, y : B \vdash M' : C}{\Gamma \vdash \mathbf{let}\ (x, y) = M\ \mathbf{in}\ M' : C} \wedge_e$$

9.2. CPS Translation of $\lambda^{\to -}_{c^-}$

The CPS translation maps the types, judgements, and terms of $\lambda^{\to -}_{c^-}$ to $\lambda^{\wedge}$. The translation we present extends the call-by-value CPS translation of Fischer (1972, 1993).

The translation on types and judgements is in Figure 24. The following proposition establishes its soundness.

PROPOSITION 23. *The CPS translation is sound with respect to typing:*

- *If* $\Gamma \vdash M : A$ *in* $\Lambda^{\to -}_{c^-}$ *then* $(\Gamma \vdash M : A)^k$ *in* $\Lambda^{\wedge}$
- *If* $\Gamma \vdash J :\bot\!\!\!\bot$ *in* $\Lambda^{\to -}_{c^-}$ *then* $(\Gamma \vdash J :\bot\!\!\!\bot)^*$ *in* $\Lambda^{\wedge}$.

The translation on terms is in Figure 25. Before proving its soundness with respect to reductions, we first state a substitution lemma which allows us to commute the CPS translation with relevant substitutions.

LEMMA 24.

(i) $[\![M[V/x]]\!]k = ([\![M]\!]k)[V^+/x]$ *and* $[\![J[V/x]]\!] = [\![J]\!][V^+/x]$

$$
\begin{array}{lcl}
x^+ & = & x \\
(\lambda x.M)^+ & = & \lambda k'.\lambda x.[\![M]\!]k' \\
(V, k\ E)^+ & = & (\lambda y.[\![k(E[y])]\!], V^+) \\
 & & \\
[\![V]\!]k & = & k\ (V^+) \\
[\![MN]\!]k & = & [\![M]\!](\lambda f.[\![N]\!](f\ k)) \\
[\![(\mathcal{C}^-(\lambda k.J))]\!]k' & = & [\![J]\!][k'/k] \\
[\![(M, k\ E)]\!]k' & = & [\![M]\!](\lambda v.(k'\ (\lambda y.[\![k(E[y])]\!], v))) \\
[\![(\mathbf{let}\ (x, k) = M\ \mathbf{in}\ N)]\!]k' & = & [\![M]\!](\lambda(k, x).[\![N]\!]k') \\
 & & \\
[\![k\ M]\!] & = & [\![M]\!]k
\end{array}
$$

Figure 25. Fischer-style call-by-value CPS translation of $\lambda^{\rightarrow -}_{\mathcal{C}^-}$

(ii) Let G be the continuation of P with respect to the context $(k'\ F[P])$, that is the continuation G such that $[\![F[P]]\!]k' = [\![P]\!]G$, then:

$$
\begin{array}{lcl}
([\![M]\!]K)\ [G/k] & = & [\![M[k'\ F/k]]\!](K\ [G/k]) \\
[\![J]\!]\ [G/k] & = & [\![J[k'\ F/k]]\!]
\end{array}
$$

PROPOSITION 25. *The CPS translation is sound with respect to semantics. If $M \rightarrow N$ in $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ then $[\![M]\!]k =_{\beta,\eta,\wedge} [\![N]\!]k$ in $\lambda^{\wedge}$.*

Proof.

- β_v: For the left-hand side we have:

$$
\begin{array}{ll}
[\![(\lambda x.M)\ V]\!]k & = \\
[\![\lambda x.M]\!](\lambda f.[\![V]\!](f\ k)) & = \\
(\lambda f.[\![V]\!](f\ k))\ (\lambda x.M)^+ & = \\
(\lambda f.[\![V]\!](f\ k))\ (\lambda q.\lambda x.[\![M]\!]q) & =_{\beta} \\
[\![V]\!](\lambda x.[\![M]\!]k) & = \\
(\lambda x.[\![M]\!]k)\ V^+ & = \\
([\![M]\!]k)[V^+/x] &
\end{array}
$$

The right-hand side is $[\![M[V/x]]\!]k$ and the result follows by Proposition 24(i).

- $\mathcal{C}^-_{elim}$: The left-hand side is:

$$
\begin{array}{ll}
[\![\mathcal{C}^-(\lambda k'.k'\ M)]\!]k & = \\
[\![k'\ M]\!][k/k'] & = \\
([\![M]\!]k')[k/k'] & = (\text{ since } k' \notin FV(M)) \\
[\![M]\!]k &
\end{array}
$$

which is equal to the right-hand side.

- $\mathcal{C}^-_{lift}$: By cases on the elementary context F:

  - F is $(\Box\ M)$: The left-hand side is:

$$\begin{array}{ll} [\![(\mathcal{C}^-(\lambda k'.J))\ M]\!]k & = \\ [\![\mathcal{C}^-(\lambda k'.J)]\!](\lambda f.[\![M]\!](f\ k)) & = \\ [\![J]\!][(\lambda f.[\![M]\!](f\ k))/k'] & \end{array}$$

    The right-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^-(\lambda k'.J[k'\ F/k'])]\!]k & = \\ [\![J[k'\ F/k']]\!][k/k'] & = \\ [\![J[k\ F/k']]\!] & \end{array}$$

    The result follows by Proposition 24(ii) since:

$$[\![P\ M]\!]k = [\![P]\!](\lambda f.[\![M]\!](f\ k))$$

  - F is $(V\ \Box)$: The left-hand side is:

$$\begin{array}{ll} [\![V\ \mathcal{C}^-(\lambda k'.J)]\!]k & = \\ [\![V]\!](\lambda f.[\![\mathcal{C}^-(\lambda k'.J)]\!](f\ k)) & = \\ (\lambda f.[\![\mathcal{C}^-(\lambda k'.J)]\!](f\ k))\ V^+ & =_\beta \\ [\![\mathcal{C}^-(\lambda k'.J)]\!](V^+\ k) & = \\ [\![J]\!][(V^+\ k)/k'] & \end{array}$$

    The right-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^-(\lambda k'.J[k'\ F/k'])]\!]k & = \\ [\![J[k'\ F/k']]\!][k/k'] & = \\ [\![J[k\ F/k']]\!] & \end{array}$$

    The result follows by Proposition 24(ii) since:

$$[\![V\ P]\!]k = [\![V]\!](\lambda f.[\![P]\!](f\ k)) =_\beta [\![P]\!](V^+\ k)$$

  - F is $(\Box, q\ E)$: The left-hand side is:

$$\begin{array}{ll} [\![(\mathcal{C}^-(\lambda k'.J), q\ E)]\!]k & = \\ [\![\mathcal{C}^-(\lambda k'.J)]\!](\lambda v.(k\ (\lambda y.[\![q\ E[y]]\!], v))) & = \\ [\![J]\!][(\lambda v.(k\ (\lambda y.[\![E[y]]\!]q, v)))/k'] & = \end{array}$$

    The right-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^-(\lambda k'.J[k'\ F/k'])]\!]k & = \\ [\![J[k'\ F/k']]\!][k/k'] & = \\ [\![J[k\ F/k']]\!] & \end{array}$$

    The result follows by Proposition 24(ii) since:

$$[\![(P, q\ E)]\!]k = [\![P]\!](\lambda v.k(\lambda y.[\![E[y]]\!], v))$$

- F is $\mathbf{let}\ (x,q) = \Box\ \mathbf{in}\ M$: The left-hand side is:

$$\begin{array}{ll} [\![\mathbf{let}\ (x,q) = \mathcal{C}^{-}(\lambda k'.J)\ \mathbf{in}\ M]\!]k & = \\ [\![\mathcal{C}^{-}(\lambda k'.J)]\!](\lambda(q,x).[\![M]\!]k) & = \\ [\![J]\!][(\lambda(q,x).[\![M]\!]k)/k'] & \end{array}$$

The right-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^{-}(\lambda k'.J[k'\ F/k'])]\!]k & = \\ [\![J[k'\ F/k']]\!][k/k'] & = \\ [\![J[k\ F/k']]\!] & \end{array}$$

The result follows by Proposition 24(ii) since:

$$[\![\mathbf{let}\ (x,q) = P\ \mathbf{in}\ M]\!]k = [\![P]\!](\lambda(q,x).[\![M]\!]k)$$

- $\mathcal{C}^{-}_{idem}$: The left-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^{-}(\lambda q.q''\ (\mathcal{C}^{-}(\lambda q'.J)))]\!]k & = \\ [\![q''\ (\mathcal{C}^{-}(\lambda q'.J))]\!][k/q] & = \\ ([\![\mathcal{C}^{-}(\lambda q'.J)]\!]q'')[k/q] & = \\ [\![J]\!][q''/q'][k/q] & = \end{array}$$

The right-hand side is:

$$\begin{array}{ll} [\![\mathcal{C}^{-}(\lambda q.J[q''/q'])]\!]k & = \\ [\![J[q''/q']]\!][k/q] & \end{array}$$

Since $J[q''/q'] = J[q''\ P/q'\ P]$, the result follows by Proposition 24(ii).

- ${Sub_v}^{base}$: The left-hand side is:

$$\begin{array}{ll} [\![\mathbf{let}\ (x,q) = (V,k'\ \Box)\ \mathbf{in}\ M]\!]k & = \\ [\![V,k'\ \Box]\!](\lambda(q,x).[\![M]\!]k) & = \\ [\![V]\!]\lambda v.((\lambda(q,x).[\![M]\!]k)\ (\lambda y.[\![k'\ y]\!],v)) & =_{\beta} \\ (\lambda(q,x).[\![M]\!]k)\ (\lambda y.[\![k'\ y]\!],V^{+}) & =_{\wedge} \\ [\![M]\!]k\ [\lambda y.k'\ y/q;V^{+}/x] & =_{\eta} \\ [\![M]\!]k\ [k'/q;V^{+}/x] & \end{array}$$

The right-hand side is $[\![M[k'/q;V/x]]\!]k$ and the result follows by Proposition 24(i,ii).

- ${Sub_v}^{step}$: The left-hand side is:

$$\begin{array}{ll} [\![\mathbf{let}\ (x,k') = (V,q\ E[F])\ \mathbf{in}\ M]\!]k & = \\ [\![(V,q\ E[F])]\!](\lambda(k',x).[\![M]\!]k) & = \\ [\![V]\!](\lambda v.((\lambda(k',x).[\![M]\!]k)\ (\lambda y.[\![q\ E[F[y]]]\!],v))) & =_{\beta} \\ ((\lambda(k',x).[\![M]\!]k)\ (\lambda y.[\![q\ E[F[y]]]\!],V^{+})) & =_{\wedge} \\ ([\![M]\!]k)[(\lambda y.[\![q\ E[F[y]]]\!])/k';V^{+}/x] & \end{array}$$

The right-hand side is:

$$\begin{array}{ll}
[\![\textbf{let } (x,k') = (V, q\ E) \textbf{ in } M[k'\ F/k']]\!]k & = \\
[\![(V, q\ E)]\!](\lambda(k',x).[\![M[k'\ F/k']]\!]k) & = \\
[\![V]\!](\lambda v.((\lambda(k',x).[\![M[k'\ F/k']]\!]k)\ (\lambda y.[\![q\ E[y]]\!], v))) & =_\beta \\
(\lambda(k',x).[\![M[k'\ F/k']]\!]k)\ (\lambda y.[\![q\ E[y]]\!], V^+) & =_\wedge \\
([\![M[k'\ F/k']]\!]k)[\lambda y.[\![q\ E[y]]\!]/k'; V^+/x] &
\end{array}$$

For $E = \Box$ we have to prove:

$$\begin{array}{l}
([\![M]\!]k)[(\lambda y.[\![q\ F[y]]\!])/k'; V^+/x] = \\
([\![M[k'\ F/k']]\!]k)[\lambda y.[\![q\ y]\!]/k'; V^+/x]
\end{array}$$

We simplify the right-hand side as follows:

$$\begin{array}{ll}
([\![M[k'\ F/k']]\!]k)[\lambda y.[\![q\ y]\!]/k'; V^+/x] & = \\
([\![M[k'\ F/k']]\!]k)[\lambda y.[\![y]\!]q/k'; V^+/x] & = \\
([\![M[k'\ F/k']]\!]k)[\lambda y.q\ y/k'; V^+/x] & =_\eta \\
([\![M[k'\ F/k']]\!]k)[q/k'; V^+/x] & = \\
([\![M[q\ F/k']]\!]k)[V^+/x] &
\end{array}$$

Since $[\![F[P]]\!]q = [\![P]\!](\lambda y.[\![q\ F[y]]\!])$, the result follows by Proposition 24(i,ii).

For E equal to a nesting of elementary contexts we decompose the substitution into a sequence of elementary substitutions. Without loss of generality we let E be equal to a nesting of two elementary contexts. We have to prove:

$$\begin{array}{l}
([\![M]\!]k)[(\lambda y.[\![q\ F_1[F_2[y]]]\!])/k'; V^+/x] = \\
([\![M[k'\ F_2/k']]\!]k)[\lambda y.[\![q\ F_1[y]]\!]/k'; V^+/x]
\end{array}$$

We proceed as follows:

$$\begin{array}{ll}
([\![M]\!]k)[(\lambda y.[\![q\ F_1[F_2[y]]]\!])/k'; V^+/x] & = \\
([\![M]\!]k)[(\lambda y.[\![k'\ F_2[y]]\!])/k'; V^+/x][\lambda y.[\![q\ F_1[y]]\!]/k'] & = \\
([\![M[k'\ F_2/k']]\!]k)[V^+/x][\lambda y.[\![q\ F_1[y]]\!]/k'] & = \\
([\![M[k'\ F_2/k']]\!]k)[\lambda y.[\![q\ F_1[y]]\!]/k'; V^+/x] &
\end{array}$$

The last step is possible since $k' \notin FV(V^+)$.

If the rewriting occurs in a subterm the result follows easily from the induction hypothesis.

$$
\begin{array}{lcl}
b^* & = & b \\
(A \ {}_U \!\!\rightarrow_T B)^* & = & \neg(\neg U^* \wedge B^*) \rightarrow \neg(\neg T^* \wedge A^*) \\
(A \rightarrow_U \perp\!\!\!\perp)^* & = & \neg(\neg U^* \wedge A^*) \\
 & & \\
(\Gamma; B \vdash M : A; C)_k^{tp} & = & \Gamma^*, tp : \neg C^*, k : \neg(\neg B^* \wedge A^*) \vdash [\![M^{tp}]\!]k : \bot \\
(\Gamma \vdash J : \perp\!\!\!\perp; C)^{tp} & = & \Gamma^*, tp : \neg C^* \vdash [\![J^{tp}]\!] : \bot
\end{array}
$$

Figure 26. Derived call-by-value $\neg\neg$-state-transformation of $\Lambda^{\rightarrow\ eff}_{\mathcal{C}^- \widehat{tp}}$

$$
\begin{array}{lcl}
x^+ & = & x \\
(\lambda x.M)^+ & = & \lambda k.\lambda(tp, x).\ [\![M^{tp}]\!]k \\
 & & \\
[\![V^{tp}]\!]k & = & k\ (tp, V^+) \\
[\![(VN)^{tp}]\!]k & = & [\![N^{tp}]\!](V^+\ k) \\
[\![(MN)^{tp}]\!]k & = & [\![M^{tp}]\!](\lambda(tp', f).\ [\![N^{tp'}]\!](f\ k)) \quad M \text{ not a value} \\
[\![(\mathcal{C}^-(\lambda k.J))^{tp}]\!]k' & = & [\![J^{tp}]\!][k'/k] \\
[\![(\mathcal{C}^-(\lambda \widehat{tp}.J))^{tp}]\!]k' & = & [\![J^{tp'}]\!][(\lambda v.\ k'\ (tp, v))/tp'] \\
 & & \\
[\![(k\ M)^{tp}]\!] & = & [\![M^{tp}]\!]k \\
[\![(\widehat{tp}\ M)^{tp}]\!] & = & [\![M^{tp}]\!](\lambda(k, x).kx)
\end{array}
$$

Figure 27. Derived call-by-value CPS translation of $\lambda^{\rightarrow}_{\mathcal{C}^- \widehat{tp}}$

9.3. Induced CPS Translations

We can compose the embedding from $\lambda^{\rightarrow}_{\mathcal{C}^- \widehat{tp}}$ to $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ with the CPS translation from $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ to $\lambda^{\wedge}$ to produce a CPS translation for $\lambda^{\rightarrow}_{\mathcal{C}^- \widehat{tp}}$. For given fresh variables tp and k, the result is given in Figures 26 and 27.

We can further derive a CPS translation for $\mathcal{C}$, *shift*, and *reset* by composing their embeddings into $\lambda^{\rightarrow}_{\mathcal{C}^- \widehat{tp}}$ (Figure 11) or into $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ (Figure 23) with the above CPS translations. Let $I' = \lambda(k, x).kx$, then we have:

$$
\begin{array}{lcl}
[\![(\#\ M)^{tp}]\!]k & = & [\![M^{tp'}]\!]I'[(\lambda v.\ k\ (tp, v))/tp'] \\
[\![(\mathcal{C}\ M)^{tp}]\!]k & = & [\![M^{tp}]\!](\lambda(tp', f).\ f\ I'\ (tp', \mathit{reifyC}\ k)) \\
[\![(\mathcal{S}\ M)^{tp}]\!]k & = & [\![M^{tp}]\!](\lambda(tp', f).\ f\ I'\ (tp', \mathit{reifyS}\ k))
\end{array}
$$

where

$$
\begin{array}{lcl}
\mathit{reifyC}\ k & = & \lambda\_.\lambda(tp, x).\ k\ (tp, x) \\
\mathit{reifyS}\ k & = & \lambda k'.\lambda(tp, x).\ k\ (\lambda y.k'\ (tp, y), x)
\end{array}
$$

The translation is, up to currying and η-conversion, the same CPS translation defining the semantics of *shift* and *reset* in the seminal paper of Danvy and Filinski (1989).

10. Other Control Operators

We have focused exclusively on functional continuations obtained with *shift* (or $\mathcal{C}$) and *reset*. We briefly review and classify some of the other control operators in the literature and discuss them based on our work.

10.1. A Short History

As we have seen, early proposals for functional continuations (Felleisen et al., 1987; Felleisen, 1988; Danvy and Filinski, 1989) had only a *single* control delimiter. The operation like *shift* for capturing the functional continuation implicitly refers to the most recent occurrence of this delimiter.

The limitations of the single control delimiter became quickly apparent, and later proposals generalised the single delimiter by allowing hierarchies of delimiters and control operators like $reset_n$ and $shift_n$ (Danvy and Filinski, 1990; Sitaram and Felleisen, 1990). At about the same time, a different proposal *spawn* allowed new delimiters to be generated dynamically (Hieb and Dybvig, 1990; Hieb et al., 1994). In this system, the base of each functional continuation is rooted at a different #. The action of creating the # returns a specialised control operator for accessing occurrences of this particular #; this specialised control operator can then be used for capturing (and aborting) the particular functional continuation rooted at the newly generated # (and only that one). This is more expressive and convenient than either single delimiters or hierarchies of delimiters and allows arbitrary nesting and composition of continuation-based abstractions. A later proposal by Gunter *et. al.* (1995) separated the operation for *creating* new delimiters from the control operator *using* that #.

10.2. Control Delimiters and Extent

The issues related to hierarchies of control delimiters or the dynamic generation of new names for control delimiters, appear orthogonal to our analysis. Indeed the presence of multiple delimiters does not change the fundamental point about the dynamic behaviour of each individual #.

However, our analysis fundamentally relies on a subtle issue related to the *extent* of delimiters (Moreau and Queinnec, 1994). More pre-

cisely, there is no question that the # delimits the part of the context that a control operator gets to capture, but given that constraint there are still *four* choices to consider with very different semantics (Dybvig et al., 2004):

$$\begin{array}{l} E_{\downarrow}[\#\ (E_{\uparrow}[\mathcal{F}_1\ M])] \ \mapsto\ E_{\downarrow}[M\ (\lambda x.E_{\uparrow}[x])] \\ E_{\downarrow}[\#\ (E_{\uparrow}[\mathcal{F}_2\ M])] \ \mapsto\ E_{\downarrow}[\#\ (M\ (\lambda x.E_{\uparrow}[x]))] \\ E_{\downarrow}[\#\ (E_{\uparrow}[\mathcal{F}_3\ M])] \ \mapsto\ E_{\downarrow}[M\ (\lambda x.\#\ (E_{\uparrow}[x]))] \\ E_{\downarrow}[\#\ (E_{\uparrow}[\mathcal{F}_4\ M])] \ \mapsto\ E_{\downarrow}[\#\ (M\ (\lambda x.\#\ (E_{\uparrow}[x])))] \end{array}$$

All four variants have been proposed in the literature: $\mathcal{F}_1$ is like the operator *cupto* (Gunter et al., 1995); $\mathcal{F}_2$ is Felleisen's $\mathcal{F}$ operator (1988); $\mathcal{F}_3$ is like a *spawn* controller (Hieb and Dybvig, 1990); and $\mathcal{F}_4$ is *shift* (Danvy and Filinski, 1990).

It turns out that the inclusion of the # in the reified continuation (variants $\mathcal{F}_3$ and $\mathcal{F}_4$) simplifies the semantics considerably. For example, when a continuation captured by $\mathcal{F}_3$ or $\mathcal{F}_4$ is invoked, the included # can be used to provide the required top-level context for that invocation. In the case of $\mathcal{F}_1$ or $\mathcal{F}_2$, there is no included #; so when the continuation is invoked, we must "search" for the # required to denote the top-level. In general, the # can be located arbitrarily deep in the calling context. Since the representation of contexts as functional continuations does not support operations for "searching for delimiters," Felleisen *et al.* (1988) developed a special model based on an *algebra of contexts* which supports the required operations. In a recent investigation of control operators for functional continuations, Dybvig *et al.* (2004) show however that it is possible to use standard continuation semantics to explain all the four variants of operators above: the trick is to augment the model with a state variable containing a *sequence of continuations and delimiters*. Chung-chieh Shan (2004) has also recently shown a similar result using a different encoding for the operators in terms of *shift* and *reset*. We believe that these encodings can be adapted as the basis for an embedding of the operators in variant (or extension) of $\lambda_{\mathcal{C}}^{\rightarrow -}$.

Acknowledgements

We thank Olivier Danvy, Matthias Felleisen, Yukiyoshi Kameyama, and Hayo Thielecke for the discussions and help they provided in understanding their results. We would also like to thank the ICFP reviewers who provided corrections and extensive comments on the presentation of the conference version.

Appendix

A. Proof of strong normalisation of $\lambda^{\rightarrow -}_{\mathcal{C}^-}$

The $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ calculus is defined by the reduction rules in Figure 19 and the type system $\Lambda^{\rightarrow -}_{\mathcal{C}^-}$ in Figure 20. We show that this calculus is *strongly normalisable* (SN).

Actually we prove the result for a mild generalisation of $\lambda^{\rightarrow -}_{\mathcal{C}^-}$ where the reduction $\mathcal{C}^-_{idem}$:

$$\mathcal{C}^-(\lambda k.k''(\mathcal{C}^-(\lambda k'.J))) \rightarrow \mathcal{C}^-(\lambda k.J[k''/k'])$$

is generalised into $k''(\mathcal{C}^-(\lambda k'.J)) \rightarrow J[k''/k']$ so that it applies directly to jumps, instead of terms including jumps.

We begin by establishing a closure property of SN that is needed in the rest of the proof. We then generalise the notion of SN to that of *reducibility* and show that all typed terms are reducible.

A.1. Properties of SN

As the following establishes, strong normalisability is preserved by head expansion.

LEMMA 26.

1. *If V and $k(E[M[V/x]])$ are SN, then $k(E[(\lambda x.M)V])$ is SN*
2. *If $k'E'$, V and $k''(E[M[V/x][k'E'/k]])$ are SN then we also have $k''(E[\mathbf{let}\ (x,k) = (V,k'E')\ \mathbf{in}\ M])$ is SN*
3. *If $k'E'$ and $J[k'E'/k]$ are SN then $k'(E'[\mathcal{C}^-(\lambda k.J)])$ is SN.*

Proof.

1. We reason by induction on the proofs of SN. If a reduction step occurs in E leading to E' then $k(E[M[V/x]]) \rightarrow k(E'[M[V/x]])$ and the induction hypothesis on $k(E'[M[V/x]])$ SN applies. Similarly for a reduction step in M. If a reduction step occurs in V leading to V' then $k(E[M[V/x]]) \twoheadrightarrow k(E[M[V'/x]])$. Hence $k(E[M[V'/x]])$ is SN and the induction hypothesis on V' SN applies. Finally, if $k(E[(\lambda x.M)V]) \rightarrow k(E[M[V/x]])$, the result is SN by hypothesis.
2. Let $J = k''(E[M[V/x][k'E'/k]])$. We reason by induction on the structure of $k'E'$, then by induction on the proofs of SN for $k'E'$, V, and J. If $E' = \Box$, then a reduction step in $k''(E[\mathbf{let}\ (x,k) = (V,k'\Box)\ \mathbf{in}\ M])$ occurs either in E, M, V or it yields J. In the first cases, we apply the induction hypotheses as above. In the latter

case, the contractum is directly SN by hypothesis. Otherwise, if E' is some $E''[F]$, we consider the different reduction steps that can occur in $k''(E[\mathbf{let}\ (x,k) = (V, k'(E''[F]))\ \mathbf{in}\ M])$:

- If the reduction step occurs in V or in $E''[F]$ leading to some jump J', then $J \twoheadrightarrow J'$ so that J' is SN and the induction hypotheses on V SN or $k'(E''[F])$ SN apply.
- If the reduction step occurs in E or in M leading to some jump J', then $J \to J'$ and the induction hypothesis on J' SN applies.
- If the reduction yields $k''(E[\mathbf{let}\ (x,k) = (V, k'E'')\ \mathbf{in}\ M[kF/k]])$ then J can be rewritten to $k''(E[M[kF/k][V/x][k'E''/k]])$ so that the induction hypothesis on E'' applies.

3. As above, we reason by induction on the structure of $k'E'$, then by induction on the proofs of SN for $k'E'$, and $J[k'E'/k]$. If $E' = \Box$, then there are two interesting cases.

   - If $k'(\mathcal{C}^-(\lambda k.J)) \to J[k'/k]$, then SN follows by hypothesis.
   - If J is some kM with k not free in M and $k'(\mathcal{C}^- k.J) \to M$, then M is a subterm of $J = J[k'/k]$ so that SN follows from $J[k'/k]$ SN.

   If E' is some $E''[F]$, then there are also two interesting cases:

   - If $k'(E''[F[\mathcal{C}^-(\lambda k.J)]]) \to k'(E''[\mathcal{C}^-(\lambda k.J[kF/k])])$, then SN follows by induction hypothesis on E'' since $J[k'(E''[F])/k]$ can be rewritten into $J[kF/k][k'E''/k]$.
   - If J is some kM with k not free in M and $k'(\mathcal{C}^- k.J) \to M$, then again, M is a subterm of $J[k'E'/k]$ so that SN follows from $J[k'E'/k]$ SN.

A.2. Reducibility

We define the property "*reducible of a given type*" for terms and jump contexts. The definition is by induction on the type, then by mutual induction on values, jump contexts, and non-value terms, with priority given first to values, then to jump contexts, and finally to non-values. The definition is a straightforward adaptation of the notion of reducibility developed by Herbelin (2001) for proving strong normalisation of the call-by-name $\overline{\lambda}\mu\tilde{\mu}$-calculus.

DEFINITION 27. [Reducible of a given type]

– A value V is reducible of type A iff:

  - V is a variable
  - V is $\lambda x.N$ and A is some $B \to C$ such that for all value V' reducible of type B, $V\ V'$ is reducible of type C
  - V is (V', kE) and A is some $B - C$ and V' is reducible of type B and kE is reducible of type C.

– A jump context kE is reducible of type A iff for all value V reducible of type A, $k(E[V])$ is SN.

– A non-value term M is reducible of type A iff for all jump contexts kE reducible of type A, $k(E[M])$ is SN.

Remark 1. Since reducible jumps contexts are SN against any reducible value, the *if* part of the last clause of the definition of reducibility also holds for values.

LEMMA 28. *Reducible values, non-value terms and contexts are SN:*

– *For all M reducible of type A, M is SN.*

– *For all kE reducible of type A, kE is SN.*

Proof. We reason by induction on the structure of M, following the different cases of the definition of being reducible:

– If M is a variable, it is SN.

– If M is $\lambda x.N$, then applying it to x yields a reducible term identical to N. By induction N is SN and hence M is SN.

– If M is (V, kE) with kE and V reducible then both kE and V are SN by induction hypothesis, so that M is SN.

– Consider now a reducible jump context kE. Take a variable x. It is a reducible value, hence $k(E[x])$ is SN. Especially kE is SN.

– Finally, if M is not a value, then $k\Box$ is a reducible context of type A, hence kM is SN. It follows that M is SN.

A.3. Properties of Reducibility

The following lemma shows that reducibility of jump contexts is preserved by context construction.

LEMMA 29. *We have:*

1. *If V is reducible of type $A \to B$ and E reducible of type B then $E[V\,\Box]$ is reducible of type A.*

2. *If M is reducible of type A and E reducible of type B then $E[\Box\ M]$ is reducible of type $A \to B$.*

3. *If kE' is reducible of type A and E reducible of type $B - A$ then $E[(\Box, kE')]$ is reducible of type B.*

4. *If $N[V/x][k'E'/k]$ is reducible of type A for any reducible V of type B and $k'E'$ of type C and if E reducible of type $B - A$ then $E[\mathbf{let}\ (x,k) = \Box\ \mathbf{in}\ N]$ is a reducible jump context of type $B - C$.*

Proof.

1. We have to show that $E[VV']$ is SN for any V' reducible of type A. If V is some variable x then redexes of $E[xV']$ are either in E or in V' which are SN by Lemma 28. If V is some $\lambda x.M$ then, by definition of its reducibility, VV' is reducible. Combined with the reducibility of E, we get $E[VV']$ SN.

2. We have to show that $E[VM]$ is SN for any V reducible of type $A \to B$. By the previous item, $E[V\Box]$ is a reducible context of type A. Hence, $E[VM]$ is SN, by reducibility of M.

3. We have to show that $E[(V, kE')]$ is SN for any V reducible of type B. The reducibility of kE' and V implies the reducibility of the value (V, kE'). Hence the reducibility of E implies $E[(V, kE')]$ SN.

4. We have to show that $E_0 = E[\mathbf{let}\ (x,k) = V\ \mathbf{in}\ N]$ is SN for any V reducible of type $B - C$. If V is some variable y then the only redexes of $E[\mathbf{let}\ (x,k) = y\ \mathbf{in}\ N]$ are in E or in N which are SN by Lemma 28, so that the whole expression is SN. If V is some value of the form $(V', k''E'')$, then by hypothesis, $N[V/x][k''E''/k]$ is reducible, hence $E[N[V/x][k''E''/k]$ is SN. By Lemma 26(2), we conclude that $E[\mathbf{let}\ (x,k) = (V, k''E'')\ \mathbf{in}\ N]$ is SN too.

Finally, we show the main lemma that all typed terms are reducible.

LEMMA 30. *Let Γ be an ordered context of declarations of the form either $x_i : A_i$ or $k_i : B_i \to \perp\!\!\!\perp$. Let V_i be instances for the variables x_i and k'_iE_i be instances for the variables k_i such that the free variables of V_i and k'_iE_i are among the x_j and k_j for $j \leq i$. The V_i are reducible values of respective types A_i and the k'_iE_i are reducible jump contexts of respective types B_i. We write $[\sigma]$ for the ordered substitution mixing the substitution $[V_i/x_i]$ and $[k'_iE_i/k_i]$. We have:*

– $\Gamma, \Delta \vdash M : A$ *implies* $M[\sigma]$ *reducible of type* A

– $\Gamma, \Delta \vdash J :\!\perp\!\!\!\perp$ *implies* $J[\sigma]$ *SN.*

Proof. We reason by induction on the derivation of $\Gamma, \Delta \vdash M : A$ or $\Gamma, \Delta \vdash J :\!\perp\!\!\!\perp$.

– Rule Ax with $M = x_i$: this is direct by reducibility of V_i.

– Rule $\rightarrow_e$ with $M = M_1 M_2$ with M_1 of type $B \rightarrow A$ and M_2 of type B. Let kE a reducible context of type A. Since $M_2' = M_2[\sigma]$ is reducible of type B by induction hypothesis, $k(E[\Box\ M_2'])$ is reducible of type $B \rightarrow A$ by Lemma 29(2). Since $M_1' = M_1[\sigma]$ is reducible by induction hypothesis, we have that $k(E[M_1' M_2'])$ is SN. Hence $M[\sigma] = M_1' M_2'$ is reducible.

– Rule $\rightarrow_i$ with $M = \lambda x.N : A \rightarrow B$. Let V be a reducible value of type A and kE a reducible jump context of type B. By induction hypothesis, we have $N[\sigma][V/x]$ reducible of type B hence $k(E[N[\sigma][V/x]])$ is SN. By Lemma 28, V is SN. By Lemma 26(1), we get $k(E[(\lambda x.(N[\sigma]))V])$ SN so that $(\lambda x.N)[\sigma] = \lambda x.(N[\sigma])$ is reducible.

– Rule $\rightarrow_e^k$ with $J = k_{i_0} M$ with M of type A and k_{i_0} of type $A \rightarrow \perp\!\!\!\perp$. By induction hypothesis we have $M' = M[\sigma]$ reducible of type A. Since $k_{i_0}' E_{i_0}$ is reducible (of type A), we get $J[\sigma] = k_{i_0}' E_{i_0}[M']$ SN.

– Rule RAA with $M = \mathcal{C}^-(\lambda k.J)$. Let $k'E$ a reducible context of type A. By induction hypothesis we have $J' = J[\sigma][k'E/k]$ SN. By Lemma 28, $k'E$ is SN so that we get $k'(E[\mathcal{C}^-(\lambda k.J)])$ SN by Lemma 26(3).

– Rule $-_e$ with $M = (\mathbf{let}\ (x,k) = N_1\ \mathbf{in}\ N_2)$ with N_1 of type $B - C$ and N_2 of type A. Let $k'E$ a reducible context of type A. We have to show that $k'(E[M[\sigma]])$ is SN. By induction hypothesis we already know that $N_1' = N_1[\sigma]$ is reducible of type $B - C$. Let $N_2' = N_2[\sigma]$. By induction hypothesis, we have that $N_2'[V/x][k''E''/k]$ is reducible of type A for every reducible V of type B and $k''E''$ of type C. Hence, by Lemma 29(4), $k'(E[\mathbf{let}\ (x,k) = \Box\ \mathbf{in}\ N_2'])$ is a reducible context of type $B - C$. By reducibility of N_1', we conclude that $k'(E[M[\sigma]]) = k'(E[\mathbf{let}\ (x,k) = N_1'\ \mathbf{in}\ N_2'])$ is SN.

– Rule $-_i$ with $M = (N, kE)$ and $A = B - C$. The typability of kE states that $\Gamma, \Delta, x : C \vdash k(E[x]) :\!\perp\!\!\!\perp$ for some fresh variable x. By induction hypothesis, $((kE)[\sigma])[V] = k(E[x])[\sigma][V/x]$ is SN so that $(kE)[\sigma]$ is reducible of type C. By induction hypothesis, $N' = N[\sigma]$ is reducible too, of type B. If N is a value, we get that

$M[\sigma] = (N', k'E')$ is reducible. Otherwise, we have to show that $k''(E''[(N', k'E')])$ is SN for every reducible jump context $k''E''$ of type $B - C$. But, by Lemma 29(3), we get that $k''(E''[(\Box, k'E')])$ is reducible of type B, hence, by reducibility of N', we conclude that $k''(E''[(N', k'E')])$ is SN.

Combining the above with Lemma 28, and because variables and empty contexts are reducible, we finally get the strong normalisability of $\lambda_{c^-}^{\rightarrow -}$.

THEOREM 31. *Typed $\lambda_{c^-}^{\rightarrow -}$ is strongly normalisable.*

References

Ariola, Z. M. and H. Herbelin: 2003, 'Minimal Classical Logic and Control Operators'. In: *Thirtieth International Colloquium on Automata, Languages and Programming , ICALP'03, Eindhoven, The Netherlands, June 30 - July 4, 2003*, Vol. 2719. pp. 871–885, Springer-Verlag, LNCS.

Ariola, Z. M., H. Herbelin, and A. Sabry: 2004, 'A Type-Theoretic Foundation of Continuations and Prompts'. In: *ACM SIGPLAN International Conference on Functional Programming*. pp. 40–53, ACM Press, New York.

Crolard, T.: 2001, 'Subtractive logic'. *Theor. Comput. Sci.* **254**(1-2), 151–185.

Crolard, T.: 2004, 'A formulae-as-types interpretation of subtractive logic'. *Journal of Logic and Computation (Special issue on Modalities in Constructive Logics and Type Theories)*. To appear.

Curien, P.-L. and H. Herbelin: 2000, 'The duality of computation'. In: *ACM SIGPLAN International Conference on Functional Programming*. pp. 233–243, ACM Press, New York.

Danvy, O. and A. Filinski: 1989, 'A Functional Abstraction of Typed Contexts'. Technical Report 89/12, DIKU, University of Copenhagen, Copenhagen, Denmark.

Danvy, O. and A. Filinski: 1990, 'Abstracting Control'. In: *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*. pp. 151–160, ACM Press, New York.

Dybvig, R. K., S. Peyton-Jones, and A. Sabry: 2004, 'A Monadic Framework for Subcontinuations'. Submitted for publication.

Felleisen, M.: 1988, 'The Theory and Practice of First-Class Prompts'. In: *Proceedings of the 15th ACM Symposium on Principles of Programming Languages (POPL '88)*. pp. 180–190, ACM Press, New York.

Felleisen, M.: 1990, 'On the Expressive Power of Programming Languages'. In: N. Jones (ed.): *ESOP '90 3rd European Symposium on Programming, Copenhagen, Denmark*, Vol. 432. New York, N.Y.: Springer-Verlag, pp. 134–151.

Felleisen, M., D. Friedman, and E. Kohlbecker: 1987, 'A syntactic theory of sequential control'. *Theoretical Computer Science* **52(3)**, 205–237.

Felleisen, M. and R. Hieb: 1992, 'The Revised Report on the Syntactic Theories of Sequential Control and State'. *Theoretical Computer Science* **103**(2), 235–271.

Felleisen, M., M. Wand, D. P. Friedman, and B. F. Duba: 1988, 'Abstract continuations: A mathematical semantics for handling full functional jumps'. In: *In Conference on LISP and Functional Programming, Snowbird, Utah*. pp. 52–62, ACM Press, New York.

Filinski, A.: 1994, 'Representing Monads'. In: *Conf. Record 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'94, Portland, OR, USA, 17-21 Jan. 1994*. pp. 446–457, ACM Press, New York.

Filinski, A.: 1999, 'Representing layered monads'. In: *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. pp. 175–188, ACM Press, New York.

Fischer, M. J.: 1972, 'Lambda-calculus schemata'. In: *Proc. ACM Conference on Proving Assertions About Programs*, Vol. 7(1) of *SIGPLAN Notices*. pp. 104–109, ACM Press, New York.

Fischer, M. J.: 1993, 'Lambda-Calculus Schemata'. *LISP and Symbolic Computation* **6**(3/4), 259–288. `<http://www.brics.dk/~hosc/vol06/03-fischer.html>` Earlier version available in the proceedings of an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

Ganz, S. E., D. P. Friedman, and M. Wand: 1999, 'Trampolined style'. In: *ACM SIGPLAN International Conference on Functional Programming*. pp. 18–27, ACM Press, New York.

Griffin, T. G.: 1990, 'The Formulae-as-Types Notion of Control'. In: *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, S an Francisco, CA, USA, 17-19 Jan 1990*. pp. 47–57, ACM Press, New York.

Gunter, C. A., D. Rémy, and J. G. Riecke: 1995, 'A Generalization of Exceptions and Control in ML-like Languages'. In: *Functional Programming & Computer Architecture*. New York, ACM Press.

Guzmán, J. and A. Suárez: 1994, 'An Extended Type System for Exceptions'. In: *Record of the fifth ACM SIGPLAN workshop on ML and its Applications*. Also appears as Research Report 2265, INRIA, BP 105 - 78153 Le Chesnay Cedex, France.

Haynes, C. T.: 1986, 'Logic Continuations'. In: *Proceedings of the Third International Conference on Logic Programming*, Vol. 225 of *Lecture Notes in Computer Science*. Berlin, pp. 671–685, Springer-Verlag.

Haynes, C. T., D. Friedman, and M. Wand: 1986, 'Obtaining coroutines from continuations'. *Journal of Computer Languages* **11**, 143–153.

Herbelin, H.: 2001, 'Explicit Substitutions and Reducibility'. *Journal of Logic and Computation* **11**(3), 431–451.

Hieb, R., K. Dybvig, and C. W. Anderson, III: 1994, 'Subcontinuations'. *Lisp and Symbolic Computation* **7**(1), 83–110.

Hieb, R. and R. K. Dybvig: 1990, 'Continuations and Concurrency'. In: *PPoPP '90, Symposium on Principles and Practice of Parallel Programming*, Vol. 25(3) of *SIGPLAN NOTICES*. Seattle, Washington, March 14-16, pp. 128–136, ACM Press, New York.

Howard, W.: 1980, 'The formulae-as-types notion of construction'. In: J. R. Hindley and J. P. Seldin (eds.): *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, pp. 479–490.

Kameyama, Y.: 2000, 'A Type-Theoretic Study on Partial Continuations'. In: *IFIP TCS*. pp. 489–504.

Kameyama, Y.: 2001, 'Towards Logical Understanding of Delimited Continuations'. In: *Proceedings of the Third ACM SIGPLAN Workshop on Continuations (CW'01)*.

Kameyama, Y. and M. Hasegawa: 2003, 'A Sound and Complete Axiomatization of Delimited Continuations'. In: *Proc. of 8th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'03, Uppsala, Sweden, 25-29 Aug. 2003*, Vol. 38(9) of *SIGPLAN Notices*. ACM Press, New York, pp. 177–188.

Lillibridge, M.: 1999, 'Unchecked Exceptions Can be Strictly More Powerful Than Call/CC'. *Higher-Order and Symbolic Computation* **12**(1), 75–104.

Moggi, E.: 1989, 'Computational lambda-calculus and monads'. In: *Proceedings of the Fourth Annual Symposium on Logic in computer science*. pp. 14–23, IEEE Press.

Moreau, L.: 1998, 'A Syntactic Theory of Dynamic Binding'. *Higher Order Symbol. Comput.* **11**(3), 233–279.

Moreau, L. and C. Queinnec: 1994, 'Partial Continuations as the Difference of Continuations. A Duumvirate of Control Operators'. In: *International Conference on Programming Language Implementation and Logic Programming (PLILP'94)*. Madrid, Spain, pp. 182–197, Springer-Verlag.

Murthy, C.: 1992, 'Control Operators, Hierarchies, and Pseudo-Classical Type Systems: A-Translation at Work'. In: *ACM workshop on Continuations*. pp. 49–71.

Parigot, M.: 1992, 'Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction'. In: *Logic Programming and Automated Reasoning: International Conference LPAR '92 Proceedings, St. Petersburg, Russia*. pp. 190–201, Springer-Verlag.

Prawitz, D.: 1965, *Natural Deduction, a Proof-Theoretical Study*. Almquist and Wiksell, Stockholm.

Rauszer, C.: 1974, 'Semi-boolean algebras and their application to intuitionistic logic with dual connectives'. *Fundamenta Mathematicae* **83**, 219–249.

Riecke, J. G. and H. Thielecke: 1999, 'Typed Exceptions and Continuations Cannot Macro-Express Each Other'. In: *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 1644 of *Lecture Notes in Computer Science*. Berlin, pp. 635–644, Springer-Verlag.

Sabry, A. and M. Felleisen: 1993, 'Reasoning about programs in continuation-passing style'. *Lisp Symb. Comput.* **6**(3-4), 289–360.

Shan, C.: 2004, 'Shift to Control'. In: O. Shivers and O. Waddell (eds.): *Proceedings of the 5th workshop on Scheme and Functional Programming*. pp. 99–107. Technical report, Computer Science Department, Indiana University, 2004.

Sitaram, D. and M. Felleisen: 1990, 'Control delimiters and their hierarchies'. *Lisp and Symbolic Computation* **3**(1), 67–99.

Thielecke, H.: 2000, 'On Exceptions versus Continuations in the Presence of State'. In: *Proceedings of the ninth European Symposium On Programming (ESOP)*, Vol. 1782 of *Lecture Notes in Computer Science*. Berlin, pp. 397–411, Springer-Verlag.

Thielecke, H.: 2001, 'Contrasting Exceptions and Continuations'. Available from `http://www.cs.bham.ac.uk/~hxt/research/exncontjournal.pdf`.

Thielecke, H.: 2002, 'Comparing Control Constructs by Double-barrelled CPS'. *Higher-order and Symbolic Computation* **15**(2/3), 119–136.

Wadler, P.: 1994, 'Monads and composable continuations.'. *Lisp and Symbolic Computation* **7**(1), 39–56.

Wand, M.: 1999, 'Continuation-based multiprocessing'. *Higher-Order and Symbolic Computation* **12**(3), 285–299. `<http://www.brics.dk/~hosc/vol12/3-wand.html>` Reprinted from the proceedings of the 1980 Lisp Conference, with a foreword.