Expressive Power of One-Shot Control Operators and Coroutines

Kentaro Kobayashi[1][0009−0004−3505−7151] and Yukiyoshi Kameyama[1][0000−0002−2693−5133]

University of Tsukuba, Ibaraki, Japan
{kentaro.kobayashi,kameyama}@acm.org

Abstract. Control operators, such as exceptions and effect handlers, provide a means of representing computational effects in programs abstractly and modularly.
While most theoretical studies have focused on multi-shot control operators, one-shot control operators – which restrict the use of captured continuations to at most once – are gaining attention for their balance between expressiveness and efficiency. This study aims to fill the gap. We present a mathematically rigorous comparison of the expressive power among one-shot control operators, including effect handlers, delimited continuations, and even asymmetric coroutines. Following previous studies on multi-shot control operators, we adopt Felleisen's macro-expressiveness as our measure of expressiveness. We verify the folklore that one-shot effect handlers and one-shot delimited-control operators can be macro-expressed by asymmetric coroutines, but not vice versa. We explain why a previous informal argument fails, and how to revise it to make a valid macro-translation.

Keywords: One-shot continuation · Effect handler & Delimited continuation · Asymmetric coroutine · Macro-expressibility

1 Introduction

Control operators are powerful tools for representing computational effects. Exceptions and coroutines are classic control operators, implemented in many languages. Delimited-control operators (e.g., shift/reset) have been extensively studied in the literature. The last decade has seen growing interest in effect handlers, which support modular abstraction of computational effects [13,14].

This paper studies the theoretical foundation of *one-shot* variants of control operators, where captured continuations are restricted to at most one use. While most studies[1] on control operators focused on unrestricted (i.e., *multi-shot*) control operators, one-shot variants have been recently gaining attention. There are several reasons to consider one-shot variants: First, they can be implemented more efficiently than multi-shot, as they avoid stack copying [3]. Second, they

[1] A notable exception is Berdine et al., who proposed linearly-used continuations [2].

may alleviate the verification burden by reducing the complexity of reasoning about sensitive resources [15]. Third, one-shotness is key to relating control operators based on continuations to classic ones found in many dynamic languages. For instance, a recent example in the former category is one-shot effect handlers in OCaml Version 5.x,[2] while the latter category includes coroutines and the yield operator, both of which are intrinsically one-shot.[3]

Despite these advantages, few authors have studied the theoretical foundation of one-shot control operators. Indeed, it is folklore that results for multi-shot control operators carry over easily to their one-shot counterparts; however, this is not the case. Since one-shotness is a dynamic property, a formal calculus for one-shot control operators should track the validity of each continuation, complicating both the semantics and precise reasoning about them. Among the few authors, de Moura and Ierusalimschy demonstrated a connection between one-shot delimited-control operators and coroutines [12]. However, this correspondence relies heavily on mutable states that can store higher-order functions, which, in our view, obscures the raw expressive power of these control operators.

We note that comparing the expressiveness of control operators is surprisingly difficult. For the case of one-shot control operators, the expressiveness results in the literature often lacked correctness proofs (e.g.,[9]), or were incorrect. In this paper, we explain why a simple and seemingly correct translation from delimited-control operators to coroutines fails to preserve semantics.

This paper studies the relative expressiveness among three one-shot control operators: one-shot effect handlers, one-shot delimited-control operators, and asymmetric coroutines [12]. We adopt macro-expressibility [5] as the basis for our comparison, since it is well-established in the literature, particularly in the study of control operators, such as Forster, Kammar, Lindley and Pretnar [7]. To our knowledge, this is the first systematic study to rigorously analyze the expressiveness of one-shot control operators.

Our contributions are threefold:

1. We prove that one-shot delimited-control operators can be macro-expressed by asymmetric coroutines.
2. We also prove that one-shot effect handlers can be macro-expressed by asymmetric coroutines.
3. We show that the converse direction does not hold: asymmetric coroutines cannot be macro-expressed by either one-shot delimited-control operators or one-shot effect handlers.

Figure 1 illustrates the macro-expressibility results established in this paper.

The remainder of this paper is organized as follows. Section 2 introduces macro-translations and the core calculus. Sections 3 and 4 present three extensions to the core calculus and establish the macro-expressibility results, while Section 5 disproves the macro-expressibility of the converse direction. Section 6 concludes the paper.

[2] https://ocaml.org

[3] James and Sabry argued that a multi-shot variant of the yield operator is as expressive as delimited-control operators [8].

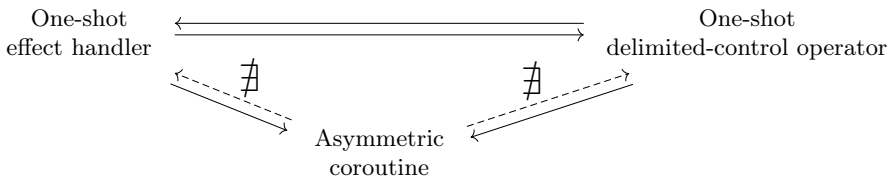

Fig. 1: Macro-expressibility among control operators. Solid lines indicate the existence of a macro-translation; dashed lines indicate its non-existence.

2 Macro-expressibility and Core Calculus

2.1 Macro-expressibility

Felleisen [5] formalized the notion of *macro-expressibility* to compare the expressive power of two programming languages when one is an extension of the other. The notion has later been adjusted to compare two languages when both are extensions of a common language [7]. Below, we show a definition based on operational semantics, in the spirit of Felleisen's formalization.

We assume that a programming language $\mathscr{L}$ is equipped with a set of $\mathscr{L}$-phrases, a set of $\mathscr{L}$-programs, which is a non-empty subset of $\mathscr{L}$-phrases, and an evaluation function $\mathrm{Eval}_{\mathscr{L}}(\cdot)$ that maps $\mathscr{L}$-programs to (suitably defined) values. Phrases and programs may be distinct in some languages. For instance, phrases can contain runtime values such as references. An n-hole syntactic abstraction in $\mathscr{L}$ is an $\mathscr{L}$-phrase that has n holes.

Definition 1. *Let $\mathscr{L}_1$ and $\mathscr{L}_2$ be conservative extensions[4] of $\mathscr{L}$. A partial map ϕ from $\mathscr{L}_1$-phrases to $\mathscr{L}_2$-phrases is a macro-translation if and only if all the following conditions are satisfied:*

1. *If M is an $\mathscr{L}_1$-program, $\phi(M)$ is defined and an $\mathscr{L}_2$-program.*
2. *If F is an n-ary function symbol of $\mathscr{L}$, for any $\mathscr{L}_1$-phrases $M_1, \ldots, M_n$, $\phi(F(M_1, \ldots, M_n)) = F(\phi(M_1), \ldots, \phi(M_n))$.*
3. *For each n-ary function symbol $F \in \mathscr{L}_1 \setminus \mathscr{L}$, there is an n-hole syntactic abstraction A in $\mathscr{L}_2$ such that $\phi(F(M_1, \ldots, M_n)) = A[\phi(M_1), \ldots, \phi(M_n)]$ for any $\mathscr{L}_1$-phrases $M_1, \ldots, M_n$.*
4. *$\mathrm{Eval}_{\mathscr{L}_1}(M)$ terminates if and only if $\mathrm{Eval}_{\mathscr{L}_2}(\phi(M))$ terminates.*

We say $\mathscr{L}_1$ is (strongly) macro-expressible in $\mathscr{L}_2$ if a macro-translation from $\mathscr{L}_1$ to $\mathscr{L}_2$ exists. We can define a weak macro-translation by replacing the "if and only if"-clause in the last condition by "only if".

An example of macro-expressibility is the let-construct in call-by-value lambda calculus. We can define a macro-translation ϕ in such a way that $\phi(\mathbf{let}\ x = M\ \mathbf{in}\ N) = (\lambda x.\ \phi(N))\ \phi(M)$ holds. Macro-expressibility is transitive: the composition of macro-translations is also a macro-translation.

[4] The formal definition of a conservative extension is provided in Appendix A.

$V, W ::=$	value	$\mid (V, W)$		pairing
$\mid x \in \mathscr{V}$	variable	$\mid \mathbf{inj}_{\mathrm{L}}\, V$	$(\mathrm{L} \in \mathscr{C})$	variant
$\mid ()$	unit	$\mid \{M\}$		thunk
$M, N ::=$	computation	$\mid \mathbf{let}\ x = M\ \mathbf{in}\ N$		sequencing
$\mid \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M$	product matching	$\mid \lambda x.\, M$		abstraction
$\mid \mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\, x_i \mapsto M_i)_i\}$	variant matching	$\mid M\ V$		application
$\mid V!$	force	$\mid \langle M, N \rangle$		pairing
$\mid \mathbf{return}\ V$	returner	$\mid \mathbf{prj}_i\, M$		projection

Fig. 2: Syntax of **MAM**

$$
\begin{aligned}
\text{pure frame } \mathcal{P} &::= \mathbf{let}\ x = [\,]\ \mathbf{in}\ N \mid [\,]\ V \mid \mathbf{prj}_i\ [\,] \\
\text{computational frame } \mathcal{F} &::= \mathcal{P} \\
\text{pure context } \mathcal{H} &::= [\,] \mid \mathcal{P}[\mathcal{H}[\,]] \\
\text{evaluation context } \mathcal{C} &::= [\,] \mid \mathcal{F}[\mathcal{C}[\,]]
\end{aligned}
$$

Fig. 3: Frames and Contexts of **MAM**

2.2 Core Calculus MAM

We use the language **MAM** (multi-adjunctive language) by Forster et al. [7], which was designed after Levy's Call-By-Push-Value calculus [10]. It serves as the common core calculus for our extensions.

Figure 2 gives the syntax of **MAM**. **MAM** differs from the untyped lambda calculus in that values and computations are clearly separated, and its own constructs are as follows: The pairing of computations $\langle M, N \rangle$ is *lazy*, a thunk freezes a computation, and a force thaws the thunk out: $\{M\}! \to^{\beta}_{\mathbf{M}} M$.

We consider terms modulo renaming of bound variables as usual. We define **MAM**-Phrases as the union of the set of values and that of computations. **MAM**-Programs are the set of computations.

To present the operational semantics of **MAM**, we define frames and contexts in Figure 3, and the beta reduction rules $\to^{\beta}_{\mathbf{M}}$ on computations in Figure 4. We define the transition relation $\to_{\mathbf{M}}$ on computations by using evaluation contexts [6] as follows:

$$\frac{M \to^{\beta}_{\mathbf{M}} M'}{\mathcal{C}[M] \to_{\mathbf{M}} \mathcal{C}[M']}$$

$(\times)$	$\mathbf{case}\ (V_1, V_2)\ \mathbf{of}\ (x_1, x_2) \mapsto M \to^{\beta}_{\mathbf{M}} M[V_1/x_1, V_2/x_2]$	(F)	$\mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ M \to^{\beta}_{\mathbf{M}} M[V/x]$
		(U)	$\{M\}! \to^{\beta}_{\mathbf{M}} M$
$(+)$	$\mathbf{case}\ \mathbf{inj}_{\mathrm{L}_k}\, V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\, x_i \mapsto M_i)_i\} \to^{\beta}_{\mathbf{M}} M_k[V/x_k]$	$(\to)$	$(\lambda x.\, M)\ V \to^{\beta}_{\mathbf{M}} M[V/x]$
		$(\&)$	$\mathbf{prj}_i\, \langle M_1, M_2 \rangle \to^{\beta}_{\mathbf{M}} M_i$

Fig. 4: Beta Reduction Rules of **MAM**

$V, W ::= \ldots$ value | $l \in \mathscr{L}_{\mathbf{D}}$ continuation label

$M, N ::= \ldots$ computation | $\mathbf{S_0} k.\ M$ shift0 | $\langle M | x.N \rangle$ dollar | **throw** $V\ W$ throw

pure frame $\mathcal{P} ::= \ldots$

computational frame $\mathcal{F} ::= \ldots \mid \langle [\,] | x.N \rangle$

pure context $\mathcal{H} ::= \ldots$

evaluation context $\mathcal{C} ::= \ldots$

Fig. 5: Syntax of $\mathbf{DEL}_{\text{one}}$

Evaluation of a program is defined by: $\text{Eval}_{\mathbf{MAM}}(M) \coloneqq V$ if $M \to_{\mathbf{M}}^{*}$ **return** V. $\text{Eval}_{\mathbf{MAM}}(M)$ is a well-defined partial function, since $\to^{\mathbf{M}}$ is deterministic.

3 One-Shot Delimited Continuations as Asymmetric Coroutines

This section investigates the macro-expressibility of one-shot delimited continuations in terms of asymmetric coroutines, which has been considered folklore, but this proves unexpectedly complicated. We first introduce two calculi: $\mathbf{DEL}_{\text{one}}$ for one-shot delimited continuations and **AC** for asymmetric coroutines, then present a macro-translation from $\mathbf{DEL}_{\text{one}}$ to **AC**.

3.1 The Calculus for One-Shot Delimited Continuations

We present the calculus $\mathbf{DEL}_{\text{one}}$, which incorporates a one-shot version of the control operator *shift*$_0$/*dollar*. The *shift*$_0$/*dollar* operator, proposed by Materzok and Biernacki [11], is a variant of *shift*$_0$/*reset*$_0$ [4] and has the same macro-expressive power. We adopt the calculus **DEL** defined by Forster et al. [7], restricting each captured continuation to one-shot use.

Figure 5 shows the syntax of $\mathbf{DEL}_{\text{one}}$ as an extension of **MAM**, where the ellipses $(\ldots)$ indicate the syntax from **MAM**. $\mathscr{L}_{\mathbf{D}}$ is a countable set of continuation labels, where l is a dynamically generated label for a continuation. The dollar term $\langle M | x.N \rangle$ is similar to reset0: when evaluated, it installs a delimiter for continuations captured in M. Unlike reset0, it has an additional part $x.N$ where x is bound in N. When M evaluates to a value, its result is bound to x, and N is evaluated. When the term $\mathbf{S_0} k.\ M$ is evaluated, a continuation delimited by the nearest dollar term is captured, k is bound to the continuation, and the body M is evaluated. If there is no surrounding dollar term, the evaluation gets stuck. **throw** $l\ V$ invokes the continuation represented by l, passing V as an argument. $\mathbf{DEL}_{\text{one}}$-Phrases is the set of all $\mathbf{DEL}_{\text{one}}$-values and $\mathbf{DEL}_{\text{one}}$-computations, and $\mathbf{DEL}_{\text{one}}$-Programs is the set of computations without continuation labels.

In $\mathbf{DEL}_{\text{one}}$, continuations can be invoked at most once. Consider the example:

$$\left\langle \begin{array}{l} \mathbf{S_0} k.\ \textbf{let}\ a = \textbf{throw}\ k\ 1\ \textbf{in} \\ \qquad \textbf{let}\ b = \textbf{throw}\ k\ 2\ \textbf{in return}\ (a, b) \end{array} \middle| x.\textbf{return}\ x \right\rangle$$

$$(\mathbf{MAM})\ \frac{M \to^{\beta}_{\mathbf{M}} M'}{\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M';\theta\rangle}$$

$$(\mathrm{ret})\quad \langle\langle \mathbf{return}\ V | x.M\rangle;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M[V/x];\theta\rangle$$

$$(\mathrm{shift})\ \frac{l \text{ is a fresh label}}{\langle\langle \mathcal{H}[\mathbf{S_0}k.\ M] | x.N\rangle;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M[l/k];\theta[l := \lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y] | x.N\rangle]\rangle}$$

$$(\mathrm{throw})\ \frac{\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y] | x.N\rangle}{\langle \mathbf{throw}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle\langle \mathcal{H}[\mathbf{return}\ V] | x.N\rangle;\theta[l := \mathbf{nil}]\rangle}$$

$$(\mathrm{fail})\ \frac{\theta(l) = \mathbf{nil}}{\langle \mathbf{throw}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{D}} \bot}$$

Fig. 6: Beta reduction rules of $\mathbf{DEL}_{\mathrm{one}}$

The term attempts to use the continuation k twice, first to compute a and then to compute b. Such usage violates the one-shotness constraint, and to detect it, we use a *store* to record the content and the status of continuations.

A store is a partial function $\theta : \mathscr{L}_{\mathbf{D}} \rightharpoonup \mathsf{computation} \sqcup \{\mathbf{nil}\}$. $\mathrm{Dom}(\theta)$ is the set of continuation labels l such that $\theta(l)$ is defined. Note that, **nil** is not an undefined element, and the complement of $\mathrm{Dom}(\theta)$ is *not* equal to $\theta^{-1}(\mathbf{nil})$. If $\theta(l) = \mathbf{nil}$, which means that the continuation labeled l has already been invoked. The empty set $\emptyset$ represents the store with no bindings.

We introduce *configurations* to describe the runtime states of programs. A configuration C is a pair of a $\mathbf{DEL}_{\mathrm{one}}$-computation M and a store θ, or the error state $\bot$. The beta reduction rules $\to^{\mathbf{D}}_{\beta}$ on configurations are defined in Figure 6.[5]

The reduction rules of $\mathbf{DEL}_{\mathrm{one}}$ are defined as follows:

$$\frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M';\theta'\rangle}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{D}} \langle \mathcal{C}[M'];\theta'\rangle} \qquad \frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \bot}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{D}} \bot}$$

Evaluation of a $\mathbf{DEL}_{\mathrm{one}}$-program is defined by: $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M) \coloneqq V$ if there exists a store θ such that $\langle M;\emptyset\rangle \to^{*}_{\mathbf{D}} \langle \mathbf{return}\ V;\theta\rangle$. $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is a well-defined partial function, since $\to_{\mathbf{D}}$ is deterministic.

3.2 The Calculus for Asymmetric Coroutines

De Moura and Ierusalimschy [12] studied several variations of coroutines and introduced two calculi, symmetric coroutines and asymmetric coroutines. In this paper, we present the calculus **AC** for asymmetric coroutines, which are prevalent in modern programming languages.

Figure 7 presents the syntax of **AC**. Besides the constructors of **MAM**, **AC** has coroutine labels as values, and four constructs to manipulate coroutines as computations: labeled computation, **create**, **resume**, and **yield**. $\mathscr{L}_{\mathbf{AC}}$ is a

[5] In (shift), we assume that y does not freely occur in the context $\mathcal{H}$. Similarly, throughout the rest of this paper, we assume that the same condition holds for variables that are freshly generated, whether in reduction rules or in translations.

$$
\begin{array}{lll}
V, W ::= \ldots & \mathsf{value} & M, N ::= \ldots \quad \mathsf{computation} \mid \mathbf{yield}\ V \quad \text{yield} \\
\quad \mid l \in \mathscr{L}_{\mathbf{AC}} & \text{coroutine label} & \quad \mid \mathbf{create}\ V \ \text{create} \quad \mid \mathbf{resume}\ V\ W \ \text{resume} \\
 & & \quad \mid l : M \quad \text{labeled computation} \\
\mathsf{pure\ frame}\ \mathcal{P} ::= \ldots & & \mathsf{pure\ context}\ \mathcal{H} ::= \ldots \\
\mathsf{computational\ frame}\ \mathcal{F} ::= \ldots \mid l : [\,] & & \mathsf{evaluation\ context}\ \mathcal{C} ::= \ldots
\end{array}
$$

Fig. 7: Syntax of **AC**

$$
\begin{array}{ll}
(\mathbf{MAM}) & \dfrac{M \to^{\beta}_{\mathbf{M}} M'}{\langle M;\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M';\theta\rangle} \\[2ex]
(\text{create}) & \dfrac{l \text{ is a fresh label}}{\langle \mathbf{create}\ V;\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ l;\theta[l := V]\rangle} \\[2ex]
(\text{resume}) & \dfrac{l \in \mathrm{Dom}(\theta) \qquad \theta(l) \neq \mathbf{nil}}{\langle \mathbf{resume}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle l : (\theta(l)!\ V);\theta[l := \mathbf{nil}]\rangle} \\[2ex]
(\text{fail}) & \dfrac{\theta(l) = \mathbf{nil}}{\langle \mathbf{resume}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{AC}} \bot} \\[2ex]
(\text{ret}) & \langle l : \mathbf{return}\ V;\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ V;\theta\rangle \\
(\text{yield}) & \langle l : \mathcal{H}[\mathbf{yield}\ V];\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ V;\theta[l := \{\lambda y.\ \mathcal{H}[\mathbf{return}\ y]\}]\rangle
\end{array}
$$

$$
\frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M';\theta'\rangle}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{AC}} \langle \mathcal{C}[M'];\theta'\rangle} \qquad \frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{AC}} \bot}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{AC}} \bot}
$$

Fig. 8: Semantics of **AC**

countable set of coroutine labels. The labeled computation $l : M$ represents the coroutine l executing the computation M. **create** V produces a new coroutine whose computation is V, and **resume** V W starts (or resumes) a coroutine whose label is V with a parameter W. **yield** V suspends the current coroutine, yielding V to its caller. **AC**-Phrases are the values and computations of **AC**, and **AC**-Programs are the computations without coroutine labels.

A store θ is a partial function that maps labels to *values* or `nil`. Note the difference from the store in $\mathbf{DEL}_{\text{one}}$, which maps labels to *computations* or `nil`. As in $\mathbf{DEL}_{\text{one}}$, a configuration in **AC** is a pair $\langle M;\theta\rangle$ of a computation M and a store θ, or the error state $\bot$.

Figure 8 defines the operational semantics of **AC** which includes the beta reduction rules $\to^{\beta}_{\mathbf{AC}}$ and the reduction rules $\to_{\mathbf{AC}}$. Note that coroutines are inherently one-shot: while a coroutine may be called multiple times by suspending and resuming it, it cannot be duplicated and reused.

Evaluation of an **AC**-program is defined by: $\mathrm{Eval}_{\mathbf{AC}}(M) \coloneqq V$ if there exists a store θ such that $\langle M;\emptyset\rangle \to^{*}_{\mathbf{AC}} \langle \mathbf{return}\ V;\theta\rangle$. Since $\to_{\mathbf{AC}}$ is deterministic, $\mathrm{Eval}_{\mathbf{AC}}$ is a well-defined partial function.

$$\underline{\langle M|x.N\rangle} \coloneqq \begin{pmatrix} \textbf{let } z = \textbf{create } \{\lambda\_.\ \textbf{let } x = \underline{M} \textbf{ in return } \{\lambda\_.\ \underline{N}\}\} \textbf{ in} \\ \textbf{let } \mathit{res} = \textbf{resume } z\ () \textbf{ in} \\ \mathit{res}!\ z \end{pmatrix}$$
$$\underline{\mathbf{S_0}k.\ L} \coloneqq \textbf{yield } \{\lambda k.\ \underline{L}\}$$
$$\underline{\textbf{throw } V\ W} \coloneqq \begin{pmatrix} \textbf{let } \mathit{res} = \textbf{resume } \underline{V}\ \underline{W} \textbf{ in} \\ \mathit{res}!\ \underline{V} \end{pmatrix}$$

Fig. 9: Naive translation from $\mathbf{DEL}_{\text{one}}$ to $\mathbf{AC}$

3.3 Naive Translation and its Failure

Given the similarity between delimited continuations and coroutines, one may think that it is straightforward to macro-translate $\mathbf{DEL}_{\text{one}}$ to $\mathbf{AC}$ with the following correspondence: a dollar term in $\mathbf{DEL}_{\text{one}}$ is translated to a create term in $\mathbf{AC}$, a shift0 term to a yield term, and a throw term to a resume term. Based on this intuition, we can define the naive translation from $\mathbf{DEL}_{\text{one}}$ to $\mathbf{AC}$ in Figure 9. Note that on the right-hand side of $\underline{\langle M|x.N\rangle}$, x corresponds to the binding occurrence of x in the original term $\langle M|x.N\rangle$. Figure 9 only shows non-trivial cases. Other cases are translated homomorphically except for continuation labels, whose translations are not defined. This is not problematic, since a macro-translation only has to translate all $\mathbf{DEL}_{\text{one}}$-programs, which do not contain continuation labels.

The naive translation works for simple expressions. However, we found a counterexaple to it. Consider the following $\mathbf{DEL}_{\text{one}}$-program M.

$$M \coloneqq \left\langle \begin{array}{l} \textbf{let } j = (\mathbf{S_0}k_1.\ \textbf{let } r_1 = \textbf{throw } k_1\ 10 \textbf{ in} \\ \qquad\qquad \textbf{let } r_2 = \textbf{throw } k_1\ 20 \textbf{ in return } r_1) \textbf{ in} \\ \mathbf{S_0}k_2.\ \textbf{return } 30 \end{array} \middle| \ i.\,\textbf{return } i \right\rangle$$

In $\mathbf{DEL}_{\text{one}}$, M is evaluated as follows: First, the shift0 term $\mathbf{S_0}k_1.\ \cdots$ is invoked, then the pure context surrounding it is captured and stored under a fresh label l_1:[6]

$$M \to_{\mathbf{D}} \textbf{let } r_1 = \textbf{throw } l_1\ 10 \textbf{ in let } r_2 = \textbf{throw } l_1\ 20 \textbf{ in return } r_1 \tag{1}$$

Next, $\textbf{throw } l_1\ 10$ invokes the captured continuation labeled l_1, rendering it invalid:

$$\begin{aligned} \ldots \to_{\mathbf{D}}^{+} & \begin{pmatrix} \textbf{let } r_1 = \left\langle \begin{array}{l} \textbf{let } j = \textbf{return } 10 \textbf{ in} \\ \mathbf{S_0}k_2.\ \textbf{return } 30 \end{array} \middle| \ i.\,\textbf{return } i \right\rangle \textbf{ in} \\ \textbf{let } r_2 = \textbf{throw } l_1\ 20 \textbf{ in return } r_1 \end{pmatrix} \\ \to_{\mathbf{D}} & \begin{pmatrix} \textbf{let } r_1 = \langle \mathbf{S_0}k_2.\ \textbf{return } 30 \mid i.\,\textbf{return } i\rangle \textbf{ in} \\ \textbf{let } r_2 = \textbf{throw } l_1\ 20 \textbf{ in return } r_1 \end{pmatrix} \end{aligned} \tag{2}$$

[6] For brevity, the full content of the store is omitted from the evaluation trace. We will only mention the changes relevant to the main computation.

$$\underline{M} \to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let } r_1 = \underline{\textbf{throw}}\ (l, l_c, 0)\ 10 \textbf{ in} \\ \textbf{let } r_2 = \underline{\textbf{throw}}\ (l, l_c, 0)\ 20 \textbf{ in} \\ \textbf{return } r_1 \end{pmatrix} \begin{bmatrix} \text{where} \\ l \mapsto \left\{ \lambda y.\ \mathcal{P}\begin{bmatrix} \textbf{let } j = \textbf{return } y \textbf{ in} \\ \textbf{yield } \{\lambda k_2.\ \textbf{return } 30\} \end{bmatrix} \right\} \\ l_c \mapsto \{\mathit{RefCell}(0)\} \end{bmatrix}$$

(**throw** increments l_c and resumes l.)

$$\to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let } r_1 = \begin{pmatrix} \textbf{let } \mathit{res} = l : \left(\mathcal{P}\begin{bmatrix} \textbf{let } j = \textbf{return } 10 \textbf{ in} \\ \textbf{yield } \{\lambda k_2.\ \textbf{return } 30\} \end{bmatrix}\right) \textbf{ in} \\ \mathit{res}!\ \underbrace{(l, l_c, 1)}_{\text{valid}} \end{pmatrix} \textbf{ in} \\ \textbf{let } r_2 = \underline{\textbf{throw}}\ \underbrace{(l, l_c, 0)}_{\text{invalid}}\ 20 \textbf{ in} \\ \textbf{return } r_1 \end{pmatrix} \tag{3}$$

$$\begin{bmatrix} \text{where} \\ l \mapsto \textbf{nil} \\ l_c \mapsto \{\mathit{RefCell}(1)\} \end{bmatrix}$$

(The coroutine labeled l is suspended by **yield**.)

$$\to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let } r_1 = \{\lambda k_2.\ \textbf{return } 30\}!\ (l, l_c, 1) \textbf{ in} \\ \textbf{let } r_2 = \underline{\textbf{throw}}\ (l, l_c, 0)\ 20 \textbf{ in} \\ \textbf{return } r_1 \end{pmatrix} \begin{bmatrix} \text{where} \\ l \mapsto \{\lambda y.\ \mathcal{P}[\textbf{return } y]\} \\ l_c \mapsto \{\mathit{RefCell}(1)\} \end{bmatrix} \tag{4}$$

(The continuation $(l, l_c, 1)$ is not used and l remains active)

$$\to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let } r_2 = \underline{\textbf{throw}}\ (l, l_c, 0)\ 20 \textbf{ in} \\ \textbf{return } 30 \end{pmatrix} \begin{bmatrix} \text{where} \\ l \mapsto \{\lambda y.\ \mathcal{P}[\textbf{return } y]\} \\ l_c \mapsto \{\mathit{RefCell}(1)\} \end{bmatrix} \tag{5}$$

Fig. 10: Evaluation of $\underline{M}$ where $\mathcal{P} \equiv \textbf{let } i = [\,] \textbf{ in return } \{\lambda\_.\ \textbf{return } i\}$. The black parts show the evaluation using the naive translation; the red parts are additions by our refined translation in Figure 11.

Then, the shift0 term $\mathbf{S_0}k_2.\ \cdots$ is invoked. This captures the context and stores it under another fresh label l_2. The evaluation then continues as follows (note that the continuation labeled l_2 is not used):

$$\begin{aligned} \ldots &\to^{+}_{\mathbf{D}} \textbf{let } r_1 = \textbf{return } 30 \textbf{ in let } r_2 = \textbf{throw } l_1\ 20 \textbf{ in return } r_1 \\ &\to_{\mathbf{D}} \textbf{let } r_2 = \textbf{throw } l_1\ 20 \textbf{ in return } 30 \end{aligned}$$

Finally, **throw** l_1 20 is invoked, but since the continuation labeled l_1 has already been consumed, the evaluation fails.

Therefore, in **AC**, $\underline{M}$ must not successfully terminate since macro-translations preserve semantics; however, this is not the case. To understand the reason, consider the evaluation trace shown in the black parts of Figure 10 (ignore the red[7] parts for the moment). First, after applying the translation, $\underline{M}$ re-

[7] In the printed (monochrome) version, the red parts may appear as gray.

duces to the term on the first line (corresponding to (1)) where the continuation captured by $\mathbf{S_0}k_1.\ \cdots$ in M is represented by a coroutine labeled l. Next, **throw** l 10 is evaluated. This resumes the coroutine labeled l and invalidates l by mapping it to **nil**. The evaluation then reaches (3), which corresponds to (2). Then, **let** j = **return** 10 **in** $\cdots$ is evaluated and the term **yield** $\{\lambda k_2.$ **return** $30\}(\equiv \underline{\mathbf{S_0}k_2.\ \mathbf{return}\ 30})$ is invoked. This suspends the coroutine labeled l, *reactivating* it in the store, as shown in (4).

This reacvation is the source of the failure. In the evaluation of M in $\mathbf{DEL}_{\mathrm{one}}$, the two shift0 invocations result in two distinct continuation labels, l_1 and l_2. Our naive translation, however, maps both of these labels to the same coroutine label l. Because of this, the reactivation of l makes the stale continuation (corresponding to l_1) available again. This would be harmless if the thunk returned by **yield** eventually resumed l, as that would invalidate l again. In our counterexample, however, this thunk does not resume l. Therefore, the coroutine labeled l remains active and the final **throw** in (5) succeeds incorrectly.

This phenomenon had been overlooked for years. In fact, there was folklore that a simple macro-translation from $\mathbf{DEL}_{\mathrm{one}}$ to **AC** should exist. For example, Kawahara and Kameyama proposed such a translation from one-shot effect handlers to asymmetric coroutines, which has been implemented in Lua, Go, and several other languages [9]. Our analysis, however, reveals that these simple translations fail to preserve semantics and therefore cannot be macro-translations. The problem becomes apparent when we examine a dollar term that contains more than one occurrence of shift0.[8]

3.4 Refined Translation from $\mathbf{DEL}_{\mathbf{one}}$ to AC

Our key idea to handle this problem is to introduce a counter mechanism into the translation so that one can distinguish valid continuations from stale ones. The red parts of Figure 10 illustrate how these counters are introduced: l_c is a counter associated with a coroutine l, and holds a counter object in the form $\mathit{RefCell}(n)$ where n is a natural number. Each counter is incremented every time the continuation associated with the corresponding coroutine is used. Each continuation also carries an index – a natural number. If this index does not match the counter's value, it indicates that the continuation has already been used, causing the computation to fail. In the second use of k_1, k_1's index is 0 while l_c's value is 1, correctly invalidating the invocation.

Realizing this idea as a macro-translation requires a bit of programming: First, we encode natural numbers by regarding $\mathbf{inj}_{\mathrm{Zero}}\,()$ as 0 and $\mathbf{inj}_{\mathrm{Succ}}$ as the successor function, then we can write increment and comparison functions in **AC**. Second, we encode counters as mutable cells that are expressible by coroutines (see Section 5). Figure 11 shows the complete translation. A dollar term is translated into a term that creates a new counter by $\mathit{ref}!\ \mathbf{inj}_{\mathrm{Zero}}\,()$, and makes a pair (z, zc) consisting of the coroutine and the corresponding counter.

[8] It can be shown that Kawahara and Kameyama's translation fails to preserve semantics by considering a similar counterexample.

$$
\begin{array}{l}
\underline{\mathbf{S_0}k.\ M} \coloneqq \mathbf{yield}\ \{\lambda k.\ \underline{M}\} \\
\underline{\langle M|x.N\rangle} \coloneqq \\
\left(\begin{array}{l}
\mathbf{let}\ z = \mathbf{create} \\
\quad \left\{\begin{array}{l}\lambda\_.\ \mathbf{let}\ x = M\ \mathbf{in} \\ \quad\ \mathbf{return}\ \{\lambda\_.\ \underline{N}\}\end{array}\right\}\ \mathbf{in} \\
\mathbf{let}\ zc = \mathit{ref}!\ 0\ \mathbf{in} \\
\mathbf{let}\ \mathit{res} = \mathbf{resume}\ z\ ()\ \mathbf{in} \\
\mathit{res}!\ ((z, zc), 0)
\end{array}\right) \\
\underline{\mathbf{throw}\ V\ W} \coloneqq \\
\left(\begin{array}{l}
\mathbf{case}\ \underline{V}\ \mathbf{of}\ \{ \\
\quad ((z, zc), i) \mapsto \\
\quad\ \mathbf{let}\ j = \mathit{get}!\ zc\ \mathbf{in} \\
\quad\ \mathbf{let}\ b = \mathit{compare}!\ i\ j\ \mathbf{in} \\
\quad\ \mathbf{case}\ b\ \mathbf{of}\ \{ \\
\quad\quad (\mathbf{inj}_{\mathrm{True}}\ ()) \mapsto \\
\quad\quad\ \mathbf{let}\ i' = \mathit{incr}!\ i\ \mathbf{in} \\
\quad\quad\ \mathbf{let}\ () = \mathit{set}!\ zc\ i'\ \mathbf{in} \\
\quad\quad\ \mathbf{let}\ \mathit{res} = \mathbf{resume}\ z\ \underline{W}\ \mathbf{in} \\
\quad\quad\ \mathit{res}!\ ((z, zc), i') \\
\quad\quad (\mathbf{inj}_{\mathrm{False}}\ ()) \mapsto \mathit{fail}! \\
\quad\ \} \\
\}
\end{array}\right)
\end{array}
$$

where

$$
\begin{array}{l}
\mathit{fail} \coloneqq \\
\left\{\begin{array}{l}\mathbf{let}\ z = \mathbf{create}\ \{\lambda\_.\ \mathbf{return}\ ()\}\ \mathbf{in} \\ \mathbf{let}\ \_ = \mathbf{resume}\ z\ ()\ \mathbf{in} \\ \mathbf{resume}\ z\ ()\end{array}\right\} \\
0 \coloneqq \mathbf{inj}_{\mathrm{Zero}}\ () \\
\mathit{Succ} \coloneqq \mathbf{inj}_{\mathrm{Succ}} \\
\mathit{incr} \coloneqq \{\lambda n.\ \mathbf{return}\ (\mathit{Succ}\ n)\} \\
\mathit{compare} \coloneqq \\
\{(\lambda x.\ \mathit{cmp}!\ \{x!\ x\})\ \{\lambda x.\ \mathit{cmp}!\ \{x!\ x\}\}\} \\
\mathit{cmp} \coloneqq \\
\left\{\begin{array}{l}\lambda f.\lambda n.\lambda m.\ \mathbf{case}\ (n, m)\ \mathbf{of}\ \{ \\ \quad (0,0) \mapsto \mathbf{inj}_{\mathrm{True}}\ () \\ \quad (0, \mathit{Succ}\ n') \mapsto \mathbf{inj}_{\mathrm{False}}\ () \\ \quad (\mathit{Succ}\ n', 0) \mapsto \mathbf{inj}_{\mathrm{False}}\ () \\ \quad (\mathit{Succ}\ n', \mathit{Succ}\ m') \mapsto f!\ n'\ m'\}\end{array}\right\} \\
\mathit{ref} \coloneqq \{\lambda v.\ \mathbf{create}\ \mathit{RefCell}(v)\} \\
\mathit{RefCell}(v) \coloneqq \\
\left\{\begin{array}{l}\lambda y.\ \mathbf{let}\ q' = \mathbf{return}\ y\ \mathbf{in} \\ \quad \{(\lambda x.\ \mathit{th}!\ \{x!\ x\})\ \{\lambda x.\ \mathit{th}!\ \{x!\ x\}\}\}!\ v\ q'\end{array}\right\} \\
\mathit{th} \coloneqq \\
\left\{\begin{array}{l}\lambda f.\lambda s.\lambda q.\ \mathbf{case}\ q\ \mathbf{of}\ \{ \\ \quad (\mathbf{inj}_{\mathrm{Set}}\ v) \mapsto \mathbf{let}\ q' = \mathbf{yield}\ ()\ \mathbf{in}\ f!\ v\ q' \\ \quad (\mathbf{inj}_{\mathrm{Get}}\ ()) \mapsto \mathbf{let}\ q' = \mathbf{yield}\ s\ \mathbf{in}\ f!\ s\ q'\}\end{array}\right\} \\
\mathit{get} \coloneqq \{\lambda c.\ \mathbf{resume}\ c\ \mathbf{inj}_{\mathrm{Get}}\ ()\} \\
\mathit{set} \coloneqq \{\lambda c.\ \lambda v.\ \mathbf{resume}\ c\ \mathbf{inj}_{\mathrm{Set}}\ v\}
\end{array}
$$

Fig. 11: Refined translation from $\mathbf{DEL}_{\mathrm{one}}$ to $\mathbf{AC}$

The last element in the argument passed to *res*! is $\mathbf{inj}_{\mathrm{Zero}}\ ()$, indicating that the first continuation label to be generated should have index 0. A throw term is translated into a term whose first argument V is the tuple $((z, zc), i)$. The translated term checks whether the value of the counter zc matches i using $\mathit{compare}!\ i\ j$. If it holds, the counter is incremented, and the continuation stored in z is resumed with the argument W. Otherwise, an invalidated continuation is about to be invoked, and the execution fails.

3.5 Simulation

To prove that the translation in Figure 11 is a valid macro-translation, we use *simulation*. For two transition systems $\mathscr{L}$ and $\mathscr{L}'$, a binary relation on $\mathscr{L}$-terms and $\mathscr{L}'$-terms is a simulation relation if the following condition holds:

> If $M \to_{\mathscr{L}} M'$ and $M \sim N$ hold, there exists an N' such that $N \to^*_{\mathscr{L}'} N'$ and $M' \sim N'$ hold.

If such a simulation relation exists, we say $\mathscr{L}'$ *simulates* $\mathscr{L}$. In our case, we shall construct a simulation relation $\sim$ on $\mathbf{DEL}_{\mathrm{one}}$-configurations and $\mathbf{AC}$-

configurations that respects the translation in Figure 11. We then use this to show that the translation preserves the semantics.

To establish the simulation, we must resolve the fundamental mismatch between $\mathbf{DEL}_{\mathrm{one}}$ and **AC**: the former generates a fresh label for each continuation capture, while the latter reuses the same coroutine labels. In our translation, this mismatch is addressed by the counter mechanism. Therefore, the simulation relation must incorporate the logic of the counter mechanism.

We introduce several auxiliary notions that are necessary to define a simulation relation. First, since configurations may contain runtime labels, we must extend the translation $\underline{\cdot}$, which is only defined for label-free $\mathbf{DEL}_{\mathrm{one}}$-programs. For this purpose, we introduce a mapping η as a partial function from $\mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}}$ to $\mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$. The extended translation $\underline{\cdot}_\eta$ is then defined by $\underline{l}_\eta \coloneqq \eta(l)$ for continuation labels l, while acting identically to $\underline{\cdot}$ for all other terms. For convenience, we identify the **AC**-representations of Peano numbers with natural numbers ($\mathbb{N}$).

Second, to track the association between a coroutine and its dedicated counter, we use a partial function κ from $\mathscr{L}_{\mathbf{AC}}$ to $\mathscr{L}_{\mathbf{AC}}$. With the partial functions η and κ, we define a binary relation $\overset{\eta}{\underset{\kappa}{\sim}}$ between configurations. We only present two crucial cases, and the complete definition can be found in Appendix B.2.

Definition 2 (excerpt). *Let C be a $\mathbf{DEL}_{\mathrm{one}}$-configuration and D be an **AC**-configuration. We inductively define a binary relation $C \overset{\eta}{\underset{\kappa}{\sim}} D$, which is parameterized by η and κ, as follows:*

(Case: return term)

$$\langle \mathbf{return}\ V; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\mathbf{return}\ V}_\eta; \tau \rangle$$

(Case: dollar term)

$$\frac{\langle M_1; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau \rangle \qquad \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle \langle M_1 | x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{array}{l} \mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in}\ \mathbf{return}\ \left\{ \lambda\_.\ \underline{M_2}_\eta \right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{array} ; \tau \right\rangle}$$

$C \overset{\eta}{\underset{\kappa}{\sim}} D$ relates each $\mathbf{DEL}_{\mathrm{one}}$-computation to one or more corresponding **AC**-computations. In most of the cases, it does so structurally; that is, $\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{M}_\eta; \tau \rangle$. However, the case for a dollar term is an exception: it involves an active coroutine label m, which lacks a corresponding continuation label. This imposes a few constraints on stores and labels. For example, $\mathrm{get}(\kappa(m), \tau) = i$ specifies that the index of the continuation label to be generated must equal the value of its counter; and $\eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil})$ requires that all continuation labels corresponding to m be invalid.

Finally, we lift the relation $\overset{\eta}{\underset{\kappa}{\sim}}$ to the simulation relation $\sim$. The relation $\overset{\eta}{\underset{\kappa}{\sim}}$ checks if the structures of two configurations correspond to each other. However, it is not sufficient; we also need to ensure that the global properties for the counter mechanism are satisfied.

For example, a label for an invalidated continuation in $\mathbf{DEL}_{\text{one}}$ must correspond to a triple in **AC** whose index is strictly smaller than its counter's value, but this property is not ensured by the relation $\overset{\eta}{\underset{\kappa}{\sim}}$. To capture such properties, we define $C \sim D$ as a binary relation that holds if and only if there exist partial functions η and κ such that $C \overset{\eta}{\underset{\kappa}{\sim}} D$ holds and certain *invariant conditions* are satisfied. A key invariant condition is "$\theta(l) = \mathbf{nil} \implies \mathrm{get}(\mathrm{pr}_2(\eta(l)), \tau) > \mathrm{pr}_3(\eta(l))$", which formalizes exactly the rule mentioned above. For the complete definition, see Appendix B.2. We remark that $C \sim D$ has the following property.

Lemma 1. *1. For any* $\mathbf{DEL}_{\text{one}}$ *computation* M, $\langle M; \emptyset \rangle \overset{\emptyset}{\underset{\emptyset}{\sim}} \langle \underline{M}; \emptyset \rangle$. *Therefore,* $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$.

2. If $\langle \mathbf{return}\ V; \theta \rangle \sim \langle N; \tau \rangle$, *then there exist a partial function* $\eta : \mathscr{L}_{\mathbf{DEL}_{\text{one}}} \rightharpoonup \mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$ *such that* $N \equiv \mathbf{return}\ \underline{V}_\eta$.

We can prove that the relation $\sim$ is a simulation relation from $\mathbf{DEL}_{\text{one}}$ to **AC**.

Theorem 1. *Let* C *be a* $\mathbf{DEL}_{\text{one}}$*-configuration and* D *be an* **AC***-configuration and assume that* $C \sim D$ *and* $C \to_{\mathbf{D}} C'$. *Then, there exists an* **AC***-configuration* D' *such that* $D \to_{\mathbf{AC}}^{+} D'$ *and* $C' \sim D'$ *hold.*

Proof. We can prove this theorm by induction on case analysis on $C \to_{\mathbf{D}} C'$. See Appendix B.3 for the complete proof.

Theorem 2. $\mathbf{DEL}_{\text{one}}$ *is macro-expressible in* **AC**.

Proof. We show that $A \mapsto \underline{A}$ is a valid macro-translation. Here, we only prove semantic preservation, i.e., $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(M)$ terminates if and only if $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ does, since the other conditions are straightforward to check. We first prove the "only if" direction. Suppose that there exist a value V and a store θ such that $\langle M; \emptyset \rangle \to_{\mathbf{D}}^{+} \langle \mathbf{return}\ V; \theta \rangle$. By Lemma 1, we have $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$. By repeatedly applying Theorem 1, we obtain $\langle \underline{M}; \emptyset \rangle \to_{\mathbf{AC}}^{+} \langle N; \tau \rangle$ and $\langle \mathbf{return}\ V; \theta \rangle \sim \langle N; \tau \rangle$ for some N and τ. By Lemma 1, we have $N \equiv \mathbf{return}\ \underline{V}_\eta$, which completes the "only if" direction. The proof of the "if" direction is provided in Appendix B.4.

The complexity of the simulation arises from the fundamental mismatch between $\mathbf{DEL}_{\text{one}}$ and **AC**. The naive translation may suffice for establishing weak macro-expressibility of $\mathbf{DEL}_{\text{one}}$ in **AC**; however, its simulation proof must still address the mismatch. In particular, it must track the correspondence between continuation labels and coroutine labels, via η and certain invariant conditions. This reflects the intrinsic difficulty of translating one-shot delimited continuations to asymmetric coroutines.

$$
\begin{array}{rll}
V, W & ::= \ldots & \mathsf{value} \\
& \mid\ l_H \quad (l \in \mathscr{L}_{\mathbf{E}}) & \text{continuation label} \\
M, N & ::= \ldots & \mathsf{computation} \\
& \mid\ \mathtt{op}\ V \quad (\mathtt{op} \in \mathscr{O}) & \text{operation call} \\
& \mid\ \mathbf{with}\ H\ \mathbf{handle}\ M & \text{handle} \\
& \mid\ \mathbf{throw}\ V\ W & \text{throw} \\
H & ::= \{\mathbf{return}\ x \mapsto M_{\mathrm{ret}},\ (\mathtt{op}_i\ p_i\ k_i \mapsto M_i)_i\} & \mathsf{handler}^{9}
\end{array}
$$

$$
\begin{array}{rlrl}
\text{pure frame} & \mathcal{P} ::= \ldots & \text{pure context} & \mathcal{H} ::= \ldots \\
\text{computational frame} & \mathcal{F} ::= \ldots \mid \mathbf{with}\ H\ \mathbf{handle}\ [\,] & \text{evaluation context} & \mathcal{C} ::= \ldots
\end{array}
$$

Fig. 12: Syntax of $\mathbf{EFF}_{\mathrm{one}}$

4 One-Shot Effect Handlers as Asymmetric Coroutines

Since Plotkin and Pretnar's proposal [14], effect handlers have been actively studied in recent years. This section extends our results to effect handlers, namely, we establish macro-expressibility of one-shot effect handlers in terms of asymmetric coroutines. Since macro-translations are composable, it suffices to show that one-shot effect handlers are macro-expressible by one-shot delimited-control operators.

4.1 The Calculus for One-Shot Effect Handlers

Figure 12 presents the syntax of $\mathbf{EFF}_{\mathrm{one}}$, the calculus for one-shot effect handlers. $\mathscr{L}_{\mathbf{E}}$ denotes a countable set of labels. A continuation label l_H is a pair of its name $l \in \mathscr{L}_{\mathbf{E}}$ and a handler H, and serves as a runtime representation of a one-shot continuation delimited by the handler H. The term $\mathtt{op}\ V$ is an operation invocation with an argument V, where the operation $\mathtt{op}$ is an element of a countable set $\mathscr{O}$. The term $\mathbf{with}\ H\ \mathbf{handle}\ M$ installs a handler H and evaluates M under it. The term $\mathbf{throw}\ V\ W$ invokes the continuation represented by V, which should be a continuation label.

We assume that every handler handles all operations that appear in programs. This is not an essential restriction, as any handler can be rewritten to satisfy this condition.[10] $\mathbf{EFF}_{\mathrm{one}}$-Phrases are the values and computations, and $\mathbf{EFF}_{\mathrm{one}}$-Programs are the computations without continuation labels.

Similarly to $\mathbf{DEL}_{\mathrm{one}}$, captured continuations can be invoked at most once during program execution, hence handling the operation $\mathtt{E}$ under the following handler raises a runtime error:

$$
H \equiv \left\{ \begin{array}{l} \mathbf{return}\ x \mapsto \mathbf{return}\ x, \\ \mathtt{E}\ p\ k \mapsto \mathbf{let}\ a = \mathbf{throw}\ k\ 1\ \mathbf{in}\ \mathbf{let}\ b = \mathbf{throw}\ k\ 2\ \mathbf{in}\ \mathbf{return}\ (a, b) \end{array} \right\}
$$

$\mathbf{EFF}_{\mathrm{one}}$ uses a store to detect one-shotness, which is a partial function: $\theta : \mathscr{L}_{\mathbf{E}} \times \mathsf{handler} \rightharpoonup \mathsf{computation} \sqcup \{\mathbf{nil}\}$.

[9] Note that x is bound in M_{ret}, and p_i and k_i are bound in their respective M_i.

[10] For each unhandled operation $\mathtt{op}$, we add a clause $\mathtt{op}\ \ p\ \ k\ \mapsto\ \mathbf{let}\ \ r\ =\ \mathtt{op}\ p\ \mathbf{in}\ \mathbf{throw}\ k\ r$ to the handler.

$$(\mathbf{MAM})\ \frac{M \to^{\beta}_{\mathbf{M}} M'}{\langle M;\theta\rangle \to^{\beta}_{\mathbf{E}} \langle M';\theta\rangle}$$

$$(\text{ret})\ \frac{H \equiv \{\mathbf{return}\ x \mapsto M_{\text{ret}},\ \ldots\}}{\langle \mathbf{with}\ H\ \mathbf{handle}\ (\mathbf{return}\ V);\theta\rangle \to^{\beta}_{\mathbf{E}} \langle M_{\text{ret}}[V/x];\theta\rangle}$$

$$(\text{op})\ \frac{H \equiv \{\mathbf{return}\ x \mapsto M_{\text{ret}},\ (\mathtt{op}_i\ p_i\ k_i \mapsto M_i)_i\} \qquad l_H \text{ is a fresh label}}{\begin{array}{c}\left\langle \mathbf{with}\ H\ \mathbf{handle}\ \left(\mathcal{H}\left[\mathtt{op}_j\ V\right]\right);\theta\right\rangle \\ \to^{\mathbf{E}}_{\beta} \langle M_j[V/p_j, l_H/k_j];\theta[l_H \coloneqq \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ x]]\rangle\end{array}}$$

$$(\text{throw})\ \frac{\theta(l_H) = \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ x]}{\langle \mathbf{throw}\ l_H\ V;\theta\rangle \to^{\beta}_{\mathbf{E}} \langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ V];\theta[l_H \coloneqq \mathbf{nil}]\rangle}$$

$$(\text{fail})\ \frac{\theta(l_H) = \mathbf{nil}}{\langle \mathbf{throw}\ l_H\ V;\theta\rangle \to^{\beta}_{\mathbf{E}} \bot}$$

Fig. 13: Beta Reduction Rules of $\mathbf{EFF}_{\text{one}}$

$$\begin{aligned}
\underline{\mathtt{op}\ V} &\coloneqq \quad \mathbf{S_0} k.\ \lambda h.\ h!\ \mathbf{inj}_{\mathtt{op}}\ (\underline{V}, \{\lambda y.\ \mathbf{throw}\ k\ y\ h\}) \\
\underline{\mathbf{throw}\ V_1\ V_2} &\coloneqq \quad \underline{V_1}!\ \underline{V_2} \\
\underline{\mathbf{with}\ H\ \mathbf{handle}\ M} &\coloneqq \quad \left\langle \underline{M} \middle| H^{\text{ret}} \right\rangle \{H^{\text{ops}}\} \\
&\quad \text{where } (\{\mathbf{return}\ x \mapsto M_{\text{ret}},\ \ldots\})^{\text{ret}} = x.\lambda_.\ \underline{M_{\text{ret}}} \\
&\quad\ \ (\{\mathbf{return}\ x \mapsto M_{\text{ret}},\ (\mathtt{op}_i\ p_i\ k_i \mapsto M_i)_i\})^{\text{ops}} \\
&\quad\ \ = \lambda c.\ \mathbf{case}\ c\ \mathbf{of}\ \{(\mathbf{inj}_{\mathtt{op_i}}\ (p_i, k_i) \mapsto \underline{M_i})_i\}
\end{aligned}$$

Fig. 14: Translation from $\mathbf{EFF}_{\text{one}}$ to $\mathbf{DEL}_{\text{one}}$

A configuration C is a pair of an $\mathbf{EFF}_{\text{one}}$-computation M and a store θ, or the error state $\bot$. The beta reduction rules $\to^{\beta}_{\mathbf{E}}$ on configurations are defined in Figure 13. The reduction of $\mathbf{EFF}_{\text{one}}$ is defined as follows:

$$\frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{E}} \langle M';\theta'\rangle}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{E}} \langle \mathcal{C}[M'];\theta'\rangle} \qquad \frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{E}} \bot}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{E}} \bot}$$

Evaluation of an $\mathbf{EFF}_{\text{one}}$-program is defined by: $\text{Eval}_{\mathbf{EFF}_{\text{one}}}(M) \coloneqq V$ if there exists a store θ such that $\langle M;\emptyset\rangle \to^{*}_{\mathbf{E}} \langle \mathbf{return}\ V;\theta\rangle$. $\text{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ is a well-defined partial function, since $\to_{\mathbf{E}}$ is deterministic.

4.2 Macro-Translation from $\mathbf{EFF}_{\text{one}}$ to $\mathbf{DEL}_{\text{one}}$

We present a translation from $\mathbf{EFF}_{\text{one}}$ to $\mathbf{DEL}_{\text{one}}$ in Figure 14, which is based on the translation by Forster et al. [7].

Theorem 3. $\mathbf{EFF}_{\text{one}}$ *is macro-expressible in* $\mathbf{DEL}_{\text{one}}$.

Proof. First, this translation maps $\mathbf{EFF}_{\text{one}}$-Programs to $\mathbf{DEL}_{\text{one}}$-Programs since it introduces no continuation labels. The translation does not alter **MAM** constructors, so it homomorphically acts on them. Also, Figure 14 clearly shows

V, W	$::= \ldots$	value	M, N	$::= \ldots$	computation	$\mid \mathbf{set}\ V\ W$	set
	$\mid l \in \mathscr{L}_{\mathbf{R}}$	reference cell		$\mid \mathbf{create}\ V$	create	$\mid \mathbf{get}\ V$	get

Fig. 15: Syntax of **REF**

that the program constructs peculiar to $\mathbf{EFF}_{\mathrm{one}}$ are expressed by syntactic abstractions of $\mathbf{DEL}_{\mathrm{one}}$. We prove the preservation of semantics in Appendix C.1.

From Theorem 3 together with Theorem 2 and the compositionality of macro-translations, we get the following theorem.

Theorem 4. $\mathbf{EFF}_{\mathrm{one}}$ *is macro-expressible in* **AC**.

4.3 Macro-Translation from $\mathbf{DEL}_{\mathbf{one}}$ to $\mathbf{EFF}_{\mathbf{one}}$

We can also show the macro-expressibility of $\mathbf{DEL}_{\mathrm{one}}$ to $\mathbf{EFF}_{\mathrm{one}}$, using Forster et al.'s translation for the multi-shot $\mathit{shift}_0/\mathit{dollar}$. See Appendix C.2 for the proof.

Theorem 5. *Assume that* `shift0` *is contained in the set of operations* $\mathscr{O}$*. Then, the following translation is a macro-translation from* $\mathbf{DEL}_{\mathrm{one}}$ *to* $\mathbf{EFF}_{\mathrm{one}}$*.*

$$\begin{aligned}
\underline{\mathbf{S_0}k.\ M} &\coloneqq \mathtt{shift0}\ \{\lambda k.\ \underline{M}\} \\
\underline{\mathbf{throw}\ V\ W} &\coloneqq \mathbf{throw}\ \underline{V}\ \underline{W} \\
\underline{\langle M|x.N\rangle} &\coloneqq \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{N},\ \mathtt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{M}
\end{aligned}$$

5 One-Shot Effect Handlers Cannot Macro-Express Asymmetric Coroutines

When asymmetric coroutines macro-express one-shot effect handlers, it is natural to ask whether the converse direction also holds. In this section, we prove that the converse direction does not hold.

To show this, we introduce **REF**, a calculus with ML-style reference cells, defined in Figure 15. The definition of **REF**-frames is omitted since it is the same as that of **MAM**-frames. **REF**-Phrases are the values and the computations of **REF**, and **REF**-Programs are the values and the computations which contain no reference cells. As in the previous calculi, we use stores and configurations: a store is a partial function that maps reference cells to *values*, and a configuration is a pair of a **REF**-computation and a store. Figure 16 presents the semantics of **REF**.

First, we prove that $\mathbf{EFF}_{\mathrm{one}}$ cannot macro-express **REF**:

Theorem 6. *There is no valid macro-translation from* **REF** *to* $\mathbf{EFF}_{\mathrm{one}}$*.*

$$(\textbf{MAM})\ \frac{M \to^{\beta}_{\mathbf{M}} M'}{\langle M;\theta\rangle \to^{\beta}_{\mathbf{R}} \langle M';\theta\rangle}$$

$$(\text{create})\ \frac{l \text{ is a fresh reference cell}}{\langle \mathbf{create}\ V;\theta\rangle \to^{\beta}_{\mathbf{R}} \langle \mathbf{return}\ l;\theta[l := V]\rangle}$$

$$(\text{set})\ \frac{l \in \mathrm{Dom}(\theta)}{\langle \mathbf{set}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{R}} \langle \mathbf{return}\ ();\theta[l := V]\rangle}$$

$$(\text{get})\ \frac{\theta(l) = V}{\langle \mathbf{get}\ l;\theta\rangle \to^{\beta}_{\mathbf{R}} \langle \mathbf{return}\ V;\theta\rangle}$$

$$\boxed{\frac{\langle M;\theta\rangle \to^{\beta}_{\mathbf{R}} \langle M';\theta'\rangle}{\langle \mathcal{C}[M];\theta\rangle \to_{\mathbf{R}} \langle \mathcal{C}[M'];\theta'\rangle}}$$

Fig. 16: Semantics of **REF**

Proof sketch. Suppose there is a macro-translation from **REF** to $\mathbf{EFF}_{\text{one}}$, and consider the following term M in **REF**.

$$M := \begin{pmatrix} \mathbf{let}\ r = \mathbf{create}\ \mathbf{inj}_{\mathrm{A}}\,()\ \mathbf{in} \\ \mathbf{let}\ i = \mathbf{get}\ r\ \mathbf{in} \\ \mathbf{let}\ \_ = \mathbf{set}\ r\ \mathbf{inj}_{\mathrm{B}}\,()\ \mathbf{in} \\ \mathbf{let}\ k = \mathbf{get}\ r\ \mathbf{in} \\ \mathbf{return}\ (i,k) \end{pmatrix} \mapsto \underline{M} = \begin{pmatrix} \mathbf{let}\ r = \underline{\mathbf{create}}\ (\mathbf{inj}_{\mathrm{A}}\,())\ \mathbf{in} \\ \mathbf{let}\ i = \underline{\mathbf{get}}\ r\ \mathbf{in} \\ \mathbf{let}\ \_ = \underline{\mathbf{set}}\ r\ (\mathbf{inj}_{\mathrm{B}}\,())\ \mathbf{in} \\ \mathbf{let}\ k = \underline{\mathbf{get}}\ r\ \mathbf{in} \\ \mathbf{return}\ (i,k) \end{pmatrix}$$

In **REF**, M evaluates to $(\mathbf{inj}_{\mathrm{A}}\,(), \mathbf{inj}_{\mathrm{B}}\,())$. In $\mathbf{EFF}_{\text{one}}$, however, $\underline{M}$ ought to evaluate to $(\mathbf{inj}_{\mathrm{A}}\,(), \mathbf{inj}_{\mathrm{A}}\,())$. This is because a macro-translation must be local and compositional, which prevents $\underline{M}$ from being enclosed by any handler in $\mathbf{EFF}_{\text{one}}$, forcing the two evaluations of $\underline{\mathbf{get}}\ r$ in $\underline{M}$ to yield the same value. See Appendix D.1 for the complete proof.

On the other hand, **AC** can macro-express **REF**:

Theorem 7. REF *is macro-expressible in* **AC**.

Proof. The following translation is a valid macro-translation from **REF** to **AC**.[11]

$$\begin{aligned} \underline{\mathbf{create}\ V} &:= \mathbf{create}\ \mathit{RefCell}(\underline{V}) \\ \underline{\mathbf{set}\ V\ W} &:= \mathbf{resume}\ \underline{V}\ (\mathbf{inj}_{\mathrm{Set}}\,\underline{W}) \\ \underline{\mathbf{get}\ V} &:= \mathbf{resume}\ \underline{V}\ (\mathbf{inj}_{\mathrm{Get}}\,()) \end{aligned}$$

We prove the details in Appendix D.2.

Corollary 1. $\mathbf{EFF}_{\text{one}}$ *cannot macro-express* **AC**.

Proof. Suppose that a macro-translation from **AC** to $\mathbf{EFF}_{\text{one}}$ exists. Then we have a macro-translation from **REF** to $\mathbf{EFF}_{\text{one}}$ by Theorem 7, which contradicts Theorem 6.

[11] $\mathit{RefCell}(\cdot)$ is defined in Figure 11.

Corollary 1 is counter-intuitive, since effect handlers are considered a universal tool for expressing various computational effects, such as coroutines. This gap arose due to the strictness of macro-translations, which adheres to Felleisen's original notion. We conjecture that, under a suitably relaxed definition of macro-translations, $\mathbf{EFF}_{\mathrm{one}}$ can macro-express **REF** and **AC**. One possible relaxation is to allow macro-translations to insert a fixed context at the root of each translated term in order to represent global effects. Such relaxed macro-translations would establish that **REF** and **AC** are macro-expressible in $\mathbf{EFF}_{\mathrm{one}}$, while preserving the compositionality of the translations. We leave the formalization and verification of this idea for future work.

6 Conclusion

This study investigated the relative macro-expressiveness of one-shot control operators, resulting in several key findings: (1) One-shot delimited-control operators and one-shot effect handlers can be macro-expressed by asymmetric coroutines, and (2) The converse direction does not hold. Previously, the former was considered trivial; however, we spotted a gap in the literature and fixed the problem by devising the counter mechanism. As our results demonstrate, establishing such a connection requires a precise mathematical analysis.

To our knowledge, this work is the first study to conduct a systematic comparison of the expressiveness of one-shot control operators. James and Sabry claimed that the one-shot yield operator can be seen as a one-shot variant of delimited-control operators [8], but they did not provide a formal justification. Forster et al. studied multi-shot control operators – effect handlers, monadic reflection, and delimited-control operators – from operational and denotational semantics, and analyzed their macro-expressibility with and without types [7]. Following their approach but adopting a purely operational viewpoint, we have revealed certain aspects of one-shot control operators that cannot be directly derived from prior multi-shot results. Meanwhile, earlier work by de Moura and Ierusalimschy examined the expressiveness of symmetric coroutines, asymmetric coroutines, and one-shot subcontinuations in the presence of mutable states [12]. In contrast, we study the calculi without mutable states, which clarifies the expressive power of control operators.

There are a number of directions for future work. Proposing a type system for coroutines and proving that our translation preserves types would be an important extension to our results. Although Anton and Thiemann proposed a type system for coroutines [1], it remains unclear whether their system aligns with our translations. Introducing affine types to statically guarantee that each continuation may be invoked at most once, in the spirit of linearly used continuations [2], is another interesting future topic.

Acknowledgments

We would like to thank anonymous reviewers for their constructive comments. This work was supported by JSPS KAKENHI Grant Number JP23K24819.

References

1. Anton, K., Thiemann, P.: Typing coroutines. In: Page, R.L., Horváth, Z., Zsók, V. (eds.) Trends in Functional Programming - 11th International Symposium, TFP 2010, Norman, OK, USA, May 17-19, 2010. Revised Selected Papers. Lecture Notes in Computer Science, vol. 6546, pp. 16–30. Springer (2010). https://doi.org/10.1007/978-3-642-22941-1_2, https://doi.org/10.1007/978-3-642-22941-1_2
2. Berdine, J., O'Hearn, P.W., Reddy, U.S., Thielecke, H.: Linear continuation-passing. High. Order Symb. Comput. **15**(2-3), 181–208 (2002). https://doi.org/10.1023/A:1020891112409, https://doi.org/10.1023/A:1020891112409
3. Bruggeman, C., Waddell, O., Dybvig, R.K.: Representing control in the presence of one-shot continuations. In: Fischer, C.N. (ed.) Proceedings of the ACM SIGPLAN'96 Conference on Programming Language Design and Implementation (PLDI), Philadephia, Pennsylvania, USA, May 21-24, 1996. pp. 99–107. ACM (1996). https://doi.org/10.1145/231379.231395, https://doi.org/10.1145/231379.231395
4. Danvy, O., Filinski, A.: A functional abstraction of typed contexts. BRICS 89/12 (Aug 1989)
5. Felleisen, M.: On the expressive power of programming languages. Sci. Comput. Program. **17**(1-3), 35–75 (1991). https://doi.org/10.1016/0167-6423(91)90036-W, https://doi.org/10.1016/0167-6423(91)90036-W
6. Felleisen, M., Friedman, D.P.: A reduction semantics for imperative higher-order languages. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE, Parallel Architectures and Languages Europe, Volume II: Parallel Languages, Eindhoven, The Netherlands, June 15-19, 1987, Proceedings. Lecture Notes in Computer Science, vol. 259, pp. 206–223. Springer (1987). https://doi.org/10.1007/3-540-17945-3_12, https://doi.org/10.1007/3-540-17945-3_12
7. Forster, Y., Kammar, O., Lindley, S., Pretnar, M.: On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. J. Funct. Program. **29**, e15 (2019). https://doi.org/10.1017/S0956796819000121, https://doi.org/10.1017/S0956796819000121
8. James, R.P., Sabry, A.: Yield: Mainstream delimited continuations. In: Proceedings on the 1st International Workshop on Theory and Practice of Delimited Continuations (TPDC 2011), 2011. p. 12 pages (2011)
9. Kawahara, S., Kameyama, Y.: One-shot algebraic effects as coroutines. In: Byrski, A., Hughes, J. (eds.) Trends in Functional Programming - 21st International Symposium, TFP 2020, Krakow, Poland, February 13-14, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12222, pp. 159–179. Springer (2020). https://doi.org/10.1007/978-3-030-57761-2_8, https://doi.org/10.1007/978-3-030-57761-2_8
10. Levy, P.B.: Call-By-Push-Value: A Functional/Imperative Synthesis, Semantics Structures in Computation, vol. 2. Springer (2004)
11. Materzok, M., Biernacki, D.: A dynamic interpretation of the CPS hierarchy. In: Jhala, R., Igarashi, A. (eds.) Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7705, pp. 296–311. Springer (2012). https://doi.org/10.1007/978-3-642-35182-2_21, https://doi.org/10.1007/978-3-642-35182-2_21
12. de Moura, A.L., Ierusalimschy, R.: Revisiting coroutines. ACM Trans. Program. Lang. Syst. **31**(2), 6:1–6:31 (2009). https://doi.org/10.1145/1462166.1462167, https://doi.org/10.1145/1462166.1462167

13. Plotkin, G.D., Power, J.: Algebraic operations and generic effects. Appl. Categorical Struct. **11**(1), 69–94 (2003). https://doi.org/10.1023/A:1023064908962, https://doi.org/10.1023/A:1023064908962
14. Plotkin, G.D., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5502, pp. 80–94. Springer (2009). https://doi.org/10.1007/978-3-642-00590-9_7, https://doi.org/10.1007/978-3-642-00590-9_7
15. de Vilhena, P.E., Pottier, F.: A separation logic for effect handlers. Proc. ACM Program. Lang. **5**(POPL), 1–28 (2021). https://doi.org/10.1145/3434314, https://doi.org/10.1145/3434314

A Definition of Conservative Extensions

In this appendix, we describe the formal definition of *conservative extensions* by Felleisen [5].

Definition 3. *A language $\mathscr{L}$ is a conservative extension of $\mathscr{L}'$ if and only if:*

1. *the constructors of $\mathscr{L}'$ is a proper subset of the constructors of $\mathscr{L}$ with the difference being $\{F_1, \ldots, F_n\}$;*
2. *The set of $\mathscr{L}'$-Phrases is a full subset of $\mathscr{L}$-Phrases which do not contain any constructors in $\{F_1, \ldots, F_n\}$;*
3. *The set of $\mathscr{L}'$-Programs is a full subset of $\mathscr{L}$-Programs which do not contain any constructors in $\{F_1, \ldots, F_n\}$; and*
4. *if an $\mathscr{L}'$-Program M evaluates to V in $\mathscr{L}'$, then also in $\mathscr{L}$, M evaluates to V.*

Informally, a conservative extension of $\mathscr{L}$ is a language extension which includes only terms generated from a finite number of additional constructors.

B Supplementary Proofs for Chapter 3

B.1 Preservation of Well-Formedness

Syntactically, a coroutine l can execute a computation that contains itself as an *active label*:

$$l : (l : M).$$

However, we demonstrate that **AC**-programs cannot reach such states by formalizing the notion of *well-formedness* and proving that it is preserved under reduction.

Definition 4. *For any* **AC***-expression E, we define $AL(E)$ as the set of all active labels appearing in E.*

Definition 5. *An* **AC***-configuration $\langle M; \theta \rangle$ is well-formed if*

1. *WellFormed*(M) *holds,*
2. $AL(V) = \emptyset$ *for all* $V \in \mathrm{Im}(\theta)$*, and*
3. *for all* $l \in AL(M)$*,* $\theta(l) = \mathbf{nil}$*,*

where WellFormed(M) *is a predicate on* **AC***-computations inductively defined as follows:*

1. $\dfrac{AL(V) \cup AL(M) = \emptyset}{\mathit{WellFormed}(\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M)}$
2. $\dfrac{AL(V) \cup (\cup_i AL(M_i)) = \emptyset}{\mathit{WellFormed}(\mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L_i}}\ x_i \mapsto M_i)_i\})}$
3. $\dfrac{AL(V) = \emptyset}{\mathit{WellFormed}(V!)}$
4. $\dfrac{AL(V) = \emptyset}{\mathit{WellFormed}(\mathbf{return}\ V)}$
5. $\dfrac{\mathit{WellFormed}(M) \qquad AL(N) = \emptyset}{\mathit{WellFormed}(\mathbf{let}\ x = M\ \mathbf{in}\ N)}$
6. $\dfrac{AL(M) = \emptyset}{\mathit{WellFormed}(\lambda x.\ M)}$
7. $\dfrac{\mathit{WellFormed}(M) \qquad AL(N) = \emptyset}{\mathit{WellFormed}(M\ V)}$
8. $\dfrac{AL(M) \cup AL(N) = \emptyset}{\mathit{WellFormed}(\langle M, N\rangle)}$
9. $\dfrac{\mathit{WellFormed}(M)}{\mathit{WellFormed}(\mathbf{prj}_i\ M)}$
10. $\dfrac{\mathit{WellFormed}(M) \qquad l \notin AL(M)}{\mathit{WellFormed}(l : M)}$
11. $\dfrac{AL(V) = \emptyset}{\mathit{WellFormed}(\mathbf{create}\ V)}$
12. $\dfrac{AL(V) \cup AL(W) = \emptyset}{\mathit{WellFormed}(\mathbf{resume}\ V\ W)}$
13. $\dfrac{AL(V) = \emptyset}{\mathit{WellFormed}(\mathbf{yield}\ V)}$

Proposition 1. *Suppose that* $\langle \mathcal{C}[M]; \theta\rangle$ *is well-formed and* $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$*. Then,* $\langle \mathcal{C}[M']; \theta'\rangle$ *is also well-formed.*

Proof. By case analysis on $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$.
Case (**MAM**):
SubCase (F):
Suppose that $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$ is

$$\langle \mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M[V/x]; \theta'\rangle.$$

Since $\mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ M$ is well-formed, V and M do not contain any active labels, implying $AL(M[V/x]) = \emptyset$. By straightforward induction on $\mathcal{C}$, we conclude that $\langle \mathcal{C}[M[V/x]]; \theta \rangle$ is well-formed.

The remaining subcases are similar.

Case (create):

Suppose that $\langle M; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta' \rangle$ is

$$\frac{l \text{ is a fresh label}}{\langle \mathbf{create}\ V; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ l; \theta[l := V] \rangle}.$$

Since l is fresh, $l \notin AL(\mathcal{C})$. The well-formedness of $\mathbf{create}\ V$ ensures $AL(V) = \emptyset$. By induction on $\mathcal{C}$, we obtain that $\langle \mathcal{C}[\mathbf{return}\ l]; \theta[l := V] \rangle$ is well-formed.

Case (resume):

Suppose that $\langle M; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta' \rangle$ is

$$\frac{l \in \mathrm{Dom}(\theta) \qquad \theta(l) \neq \mathbf{nil}}{\langle \mathbf{resume}\ l\ V; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle l : (\theta(l)!\ V); \theta[l := \mathbf{nil}] \rangle}.$$

If l is active in a well-formed configuration $\langle \mathcal{C}[\mathbf{resume}\ l\ V]; \theta \rangle$, it follows that $\theta(l) = \mathbf{nil}$, but this is a contradiction. Therefore, l is distinct in $\langle \mathcal{C}[l : (\theta(l)!\ V)]; \theta[l := \mathbf{nil}] \rangle$ and mapped to $\mathbf{nil}$ by $\theta[l := \mathbf{nil}]$. From the well-formedness of $\langle \mathcal{C}[\mathbf{resume}\ l\ M]; \theta \rangle$, we have that $AL(\theta(l)) = \emptyset$ and $AL(V) = \emptyset$. Then, it is easy to check that this configuration is well-formed by induction on $\mathcal{C}$.

Case (fail):

(fail) does not occur, as $\langle \mathcal{C}[M]; \theta \rangle$ reduces to $\langle \mathcal{C}[M']; \theta' \rangle$, not $\bot$.

Case (ret):

Immediate from the definition of well-formedness.

Case (yield):

Suppose $\langle M; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta' \rangle$ is

$$\langle l : \mathcal{H}[\mathbf{yield}\ V]; \theta \rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ V; \theta[l := \{\lambda y.\ \mathcal{H}[\mathbf{return}\ y]\}] \rangle$$

l is no longer active in $\langle \mathcal{C}[\mathbf{return}\ V]; \theta[l := \{\lambda y.\ \mathcal{H}[\mathbf{return}\ y]\}] \rangle$, while other active labels remain distinct and are mapped to $\mathbf{nil}$ by $\theta[l := \{\lambda y.\ \mathcal{H}[\mathbf{return}\ y]\}]$. Moreover, from the well-formedness of $\mathcal{H}[\mathbf{yield}\ V]$, we obtain that $\mathcal{H}$ and V does not contain any active labels. Therefore, by induction on $\mathcal{C}$, we see that $\langle \mathcal{C}[\mathbf{return}\ V]; \theta[l := \{\lambda y.\ \mathcal{H}[\mathbf{return}\ y]\}] \rangle$ is well-formed.

This completes the proof.

B.2 Definition of Simulation Relation

Definition 6. *For any $l \in \mathscr{L}_{\mathbf{AC}}$, $\mathbf{AC}$-store τ, and natural number i, we define* $\mathrm{get}(l, \tau) = i$ *when*

$$\tau(l) = \mathit{RefCell}(i).$$

Definition 7. *Let η be a partial function from $\mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}}$ to $\mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$, κ be a partial function from $\mathscr{L}_{\mathbf{AC}}$ to $\mathscr{L}_{\mathbf{AC}}$, and $\mathrm{Conf}_{\mathbf{DEL}_{\mathrm{one}}}$ and $\mathrm{Conf}_{\mathbf{AC}}$ be the sets of configurations of $\mathbf{DEL}_{\mathrm{one}}$ and $\mathbf{AC}$, respectively. We inductively define a binary relation $\overset{\eta}{\underset{\kappa}{\sim}}$ over $\mathrm{Conf}_{\mathbf{DEL}_{\mathrm{one}}} \times \mathrm{Conf}_{\mathbf{AC}}$, which is parameterized by η and κ, as follows:*

1. $\langle \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M}_{\eta}; \tau \right\rangle$,
2. $\left\langle \mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\ x_i \mapsto M_i)_i\}; \theta \right\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\ x_i \mapsto M_i)_i\}}_{\eta}; \tau \right\rangle$,
3. $\langle V!; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{V!}_{\eta}; \tau\rangle$,
4. $\langle \mathbf{return}\ V; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\mathbf{return}\ V}_{\eta}; \tau\rangle$,
5. $\langle \lambda x.\ M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\lambda x.\ M}_{\eta}; \tau\rangle$,
6. $\langle \langle M_1, M_2\rangle; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\langle M_1, M_2\rangle}_{\eta}; \tau \right\rangle$,
7. $\langle \mathbf{S_0} k.\ M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{S_0} k.\ M}_{\eta}; \tau \right\rangle$,
8. $\langle \langle M_1 | x.M_2\rangle; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\langle M_1 | x.M_2\rangle}_{\eta}; \tau \right\rangle$,
9. $\langle \mathbf{throw}\ V\ W; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\mathbf{throw}\ V\ W}_{\eta}; \tau\rangle$,
10. $l \dfrac{\langle M_1; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau\rangle}{\langle \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_{\eta}; \tau \right\rangle}$,
11. $\dfrac{\langle M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau\rangle}{\langle M\ V; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N\ \underline{V}_{\eta}; \tau \right\rangle}$,
12. $\dfrac{\langle M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau\rangle}{\langle \mathbf{prj}_i\ M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathbf{prj}_i\ N; \tau\rangle}$, *and*
13. $\dfrac{\langle M_1; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau\rangle \quad \mathrm{get}(\kappa(m), \tau) = i \quad \tau(m) = \mathbf{nil} \quad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle \langle M_1 | x.M_2\rangle; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{array}{l} \mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M_2}_{\eta}\right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{array}; \tau \right\rangle}$.

We refer to the pairs of evaluation contexts and stores as *contextual configurations* and write $\mathrm{CConf}_{\mathbf{DEL}_{\mathrm{one}}}$ and $\mathrm{CConf}_{\mathbf{AC}}$ for the sets consisting of them.

Definition 8. *Let η be a partial function from $\mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}}$ to $\mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$, κ be a partial function from $\mathscr{L}_{\mathbf{AC}}$ to $\mathscr{L}_{\mathbf{AC}}$, and $\mathrm{CConf}_{\mathbf{DEL}_{\mathrm{one}}}$ and $\mathrm{CConf}_{\mathbf{AC}}$ be the sets of contextual configurations of $\mathbf{DEL}_{\mathrm{one}}$ and $\mathbf{AC}$, respectively. We inductively define a binary relation $\overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}}$ over $\mathrm{CConf}_{\mathbf{DEL}_{\mathrm{one}}} \times \mathrm{CConf}_{\mathbf{AC}}$, which is parameterized by η and κ, as follows:*

1. $\langle [\,];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle [\,];\tau\rangle$,
2. $\dfrac{\langle \mathcal{C};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D};\tau\rangle}{\langle \mathbf{let}\ x = \mathcal{C}[]\ \mathbf{in}\ M_2;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \left\langle \mathbf{let}\ x = \mathcal{D}[]\ \mathbf{in}\ \underline{M_2}_{\eta};\tau\right\rangle}$,
3. $\dfrac{\langle \mathcal{C};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D};\tau\rangle}{\langle (\mathcal{C}[])\ V;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \left\langle (\mathcal{D}[])\ \underline{V}_{\eta};\tau\right\rangle}$,
4. $\dfrac{\langle \mathcal{C};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D};\tau\rangle}{\langle \mathbf{prj}_i\,\mathcal{C}[];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathbf{prj}_i\,\mathcal{D}[];\tau\rangle}$,
5. $$\frac{\langle \mathcal{C};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D};\tau\rangle \qquad \mathrm{get}(\kappa(m),\tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m,\kappa(m),\_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle \langle \mathcal{C}[]|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \left\langle \begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}[]\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M_2}_{\eta}\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i)\end{array} ;\tau\right\rangle}.$$

Definition 9. *For any configurations of* $\mathbf{DEL}_{\mathrm{one}}$ *and* $\mathbf{AC}$*, say* C *and* D*,* $C \sim D$ *means that* $C \equiv D \equiv \bot$*, or there exist* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M*,* $\mathbf{AC}$*-computation* N*,* $\mathbf{DEL}_{\mathrm{one}}$*-store* θ*,* $\mathbf{AC}$*-store* τ*, and partial functions* $\eta : \mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}} \rightharpoonup \mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$ *and* $\kappa : \mathscr{L}_{\mathbf{AC}} \rightharpoonup \mathscr{L}_{\mathbf{AC}}$ *such that the following conditions are satisfied:*

1. $C = \langle M;\theta\rangle$.
2. $D = \langle N;\tau\rangle$.
3. $C \overset{\eta}{\underset{\kappa}{\sim}} D$.
4. D *is well-formed.*
5. κ *is injective.*
6. $\mathrm{Dom}(\kappa) \cap \mathrm{Im}(\kappa) = \emptyset$.
7. $\mathrm{Dom}(\kappa) \sqcup \mathrm{Im}(\kappa) = \mathrm{Dom}(\tau)$.
8. $\mathrm{Dom}(\eta) = \mathrm{Dom}(\theta)$.
9. *For any* $l \in \mathrm{Dom}(\eta)$*, if* $\eta(l) = (z, zc, i)$*, then* $z \in \mathrm{Dom}(\kappa)$ *and* $\kappa(z) = zc$.
10. *For any* $l \in \mathrm{Dom}(\eta)$*, if* $\theta(l) = \mathbf{nil}$*, then* $\mathrm{get}(\mathrm{pr}_2(\eta(l)),\tau) > \mathrm{pr}_3(\eta(l))$.
11. *For any* $l \in \mathrm{Dom}(\eta)$*, if* $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y]|x.M'\rangle$*, then*
    (a) l *is not contained in* M'*,*
    (b) $\mathrm{get}(\mathrm{pr}_2(\eta(l)),\tau) = \mathrm{pr}_3(\eta(l))$*,*
    (c) *for any* $l' \in \mathrm{Dom}(\theta)$*, if* $l' \neq l$ *and* $\mathrm{pr}_1(\eta(l')) = \mathrm{pr}_1(\eta(l))$*, then* $\theta(l') = \mathbf{nil}$*, and*
    (d) $\tau(\mathrm{pr}_1(\eta(l))) = \left\{\lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_{\eta}[\mathbf{return}\ x]\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M'}_{\eta}\right\}\right\}$.

We define that θ, τ, η, and τ satisfy the *invariant conditions* (IC) if the conditions 5 through 11 in Definition 9 are fulfilled. Since well-formedness is preserved under reduction (Proposition 1), we leave it implicit unless necessary.

B.3 Proof of Theorem 1

In this section, we give a proof of Theorem. 1. First, we extend the translation $\underline{\cdot}_\eta$ on frames and contexts. For example, we define $\underline{\textbf{let } x = \mathcal{C}[] \textbf{ in } M}_\eta$ as $\textbf{let } x = (\underline{\mathcal{C}}_\eta[]) \textbf{ in } \underline{M}_\eta$.

We present several auxiliary lemmas for proving Theorem 1.

Lemma 2. *For any $\eta : \mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}} \rightharpoonup \mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$, $\mathbf{DEL}_{\mathrm{one}}$-computation M with free variables $x_1, \ldots, x_n$, and $\mathbf{DEL}_{\mathrm{one}}$-values $V_1, \ldots, V_n$,*

$$\underline{M}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv \underline{M[V_1/x_1, \ldots, V_n/x_n]}_\eta$$

holds [12].

Proof. We prove a more general form of this lemma.

Suppose E is a $\mathbf{DEL}_{\mathrm{one}}$-computation or a $\mathbf{DEL}_{\mathrm{one}}$-value with free variables. Then, we aim to show that

$$\underline{E}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv \underline{E[V_1/x_1, \ldots, V_n/x_n]}_\eta.$$

by induction on E. Suppose first that E is a $\mathbf{DEL}_{\mathrm{one}}$-value. Then E has six possible forms. If E is a variable x and $x \equiv x_i$ for some i, then

$$\underline{x_i}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv \underline{V_i}_\eta \equiv \underline{x_i[V_1/x_1, \ldots, V_n/x_n]}_\eta.$$

If x is not any of $x_1, \ldots, x_n$,

$$\underline{x}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv x \equiv \underline{x[V_1/x_1, \ldots, V_n/x_n]}_\eta.$$

If $E \equiv ()$ or $E \equiv l$, the result is immediate. Suppose that $E \equiv (V, W)$. By the induction hypothesis, we see that

$$\underline{V}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv \underline{V[V_1/x_1, \ldots, V_n/x_n]}_\eta$$

and

$$\underline{W}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \equiv \underline{W[V_1/x_1, \ldots, V_n/x_n]}_\eta.$$

Using these fact, it follows that

$$\begin{aligned}
\underline{(V,W)}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] &\equiv (\underline{V}_\eta, \underline{W}_\eta)\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right] \\
&\equiv (\underline{V}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right], \underline{W}_\eta\left[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n\right]) \\
&\equiv (\underline{V[V_1/x_1, \ldots, V_n/x_n]}_\eta, \underline{W[V_1/x_1, \ldots, V_n/x_n]}_\eta) \\
&\equiv \underline{(V[V_1/x_1, \ldots, V_n/x_n], W[V_1/x_1, \ldots, V_n/x_n])}_\eta \\
&\equiv \underline{(V,W)[V_1/x_1, \ldots, V_n/x_n]}_\eta.
\end{aligned}$$

[12] We use $\equiv$ to denote syntactic equality

The other cases are similar.

When E is a $\mathbf{DEL}_{\mathrm{one}}$-value, we can prove it similarly. If E is a term unique to $\mathbf{DEL}_{\mathrm{one}}$, the calculation is more tangled but follows essentially the same steps as for the other constructors.

Lemma 3. *For any $\mathbf{DEL}_{\mathrm{one}}$-computation M, $\mathbf{DEL}_{\mathrm{one}}$-store θ, $\mathbf{AC}$-store τ, and partial functions $\eta : \mathscr{L}_{\mathbf{DEL}_{\mathrm{one}}} \rightharpoonup \mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}} \times \mathbb{N}$ and $\kappa : \mathscr{L}_{\mathbf{AC}} \rightharpoonup \mathscr{L}_{\mathbf{AC}}$,*

$$\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{M}_{\eta};\tau\rangle$$

holds.

Proof. We prove this lemma by induction on M, but give only one case since the other cases follow the similar steps. Suppose that $M \equiv (M'\ V)$ for some $\mathbf{DEL}_{\mathrm{one}}$-computation M' and $\mathbf{DEL}_{\mathrm{one}}$-value V. Then, using the induction hypothesis, we obtain $\langle M';\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{M'}_{\eta};\tau\rangle$. Thus, it immediately follows that $\langle M'\ V;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{M'}_{\eta}\ \underline{V}_{\eta};\tau\rangle$.

Lemma 4. *Let $\mathcal{H}$ be a pure $\mathbf{DEL}_{\mathrm{one}}$-context and $\mathcal{E}$ an $\mathbf{AC}$-context. Then, for any θ, τ, η, and κ, $\langle \mathcal{H};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{E};\tau\rangle$ if and only if $\mathcal{E} \equiv \underline{\mathcal{H}}_{\eta}$.*

Proof. Both necessity and sufficiency can be proved by straightforward induction on $\langle \mathcal{H};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{G};\tau\rangle$ and $\mathcal{H}$, respectively.

Lemma 5. *For any pure context $\mathcal{H}$, if $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$, then $\langle \mathcal{H}[M];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\mathcal{H}}_{\eta}[N];\tau\rangle$.*

Proof. By straightforward induction on $\mathcal{H}$.

Lemma 6. *Let $\mathcal{H}$ be a pure context of $\mathbf{DEL}_{\mathrm{one}}$. Then, if $\langle \mathcal{H}[M];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$, there exist a $\mathbf{AC}$-computation N' such that*

$$N \equiv \underline{\mathcal{H}}_{\eta}[N'],$$
$$\langle \mathcal{H}[M];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \underline{\mathcal{H}}_{\eta}[N';\tau\rangle,$$
$$\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N';\tau\rangle.$$

Proof. By straightforward induction on $\mathcal{H}$.

Lemma 7. *Suppose that $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$ and $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N];\tau\rangle$. Then, $\mathcal{D} \equiv []$.*

Proof. By induction on the derivation of $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$. First, suppose that $N \equiv \underline{M}_\eta$. Then, we see that $\mathcal{D}[N] \equiv \underline{M}_\eta$. Therefore, we obtain $\mathcal{D} \equiv []$.

For the inductive step, we only show interesting cases for brevity. We assume that the derivation of $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$ is

$$\frac{\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1;\tau\rangle}{\langle M \equiv (\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2);\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \left(\mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_\eta\right);\tau\right\rangle}.$$

Suppose that $\mathcal{D} \neq []$. Then, there is a context $\mathcal{D}'$ such that $\mathcal{D} \equiv \left(\mathbf{let}\ x = \mathcal{D}'[]\ \mathbf{in}\ \underline{M_2}_\eta\right)$ and $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \mathcal{D}'[N]\left(\equiv \mathcal{D}'\left[\mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_\eta\right]\right);\tau\right\rangle$. Applying the inductive hypothesis to this, we obtain

$$\mathcal{D}'\left[\mathbf{let}\ x = [\,]\ \mathbf{in}\ \underline{M_2}_\eta\right] \equiv [],$$

which is a contradiction.

Next, we assume that the derivation of $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$ is

$$\frac{\begin{array}{c}\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1;\tau\rangle \\ \mathrm{get}(\kappa(m),\tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m,\kappa(m),\_) \subseteq \theta^{-1}(\mathbf{nil})\end{array}}{\langle M \equiv \langle M_1|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \left(\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i)\end{array}\right);\tau\right\rangle}.$$

Suppose that $\mathcal{D} \neq []$. Then, there is a context $\mathcal{D}'$ such that

$$\mathcal{D} \equiv \left(\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}'[]\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i)\end{array}\right)$$

and $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N];\tau\rangle$. This is because, if the derivation of $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N];\tau\rangle$ is

$$\langle\langle M_1|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \mathcal{D}[N] \equiv \underline{\langle M_1|x.M_2\rangle}_\eta;\tau\right\rangle,$$

$\underline{\langle M_1|x.M_2\rangle}_\eta$ contains a labeled computation $m : ()$, which is not in the image of the macro-translation. Therefore, from the inductive hypothesis, we obtain

$$\mathcal{D}'\left[\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = []\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i)\end{array}\right] \equiv [],$$

which is a contradiction. The other cases are similar.

Lemma 8. *Suppose that* $\langle \mathcal{C}[M];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N];\tau\rangle$ *and* $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$. *Then,* $\langle \mathcal{C};\theta\rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D};\tau\rangle$.

Proof. By induction on $\mathcal{C}$. The base case follows directly from Lemma 7. Suppose that $C \equiv (\mathbf{let}\ x = \mathcal{C}'[]\ \mathbf{in}\ M')$ for some $\mathcal{C}'$ and M'. Then, the derivation of $\langle \mathcal{C}[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau\rangle$ is

$$\frac{\langle \mathcal{C}'[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau\rangle}{\langle \mathbf{let}\ x = \mathcal{C}'[M]\ \mathbf{in}\ M'; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \mathbf{let}\ x = \mathcal{D}'[N]\ \mathbf{in}\ \underline{M'}_\eta; \tau\right\rangle}.$$

Hence, we have $\langle \mathcal{C}'[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau\rangle$. From the induction hypothesis, we obtain

$$\langle \mathcal{C}'; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'; \tau\rangle,$$

which implies that $\langle \mathcal{C}; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}; \tau\rangle$.

We present another non-trivial case: $C \equiv \langle \mathcal{C}'[]|x.M'\rangle$. In this case, we argue that the derivation of $\langle \mathcal{C}[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau\rangle$ is

$$\frac{\begin{array}{c}\langle \mathcal{C}'[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau\rangle \\ \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil})\end{array}}{\langle\langle \mathcal{C}'[M]|x.M'\rangle; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}'[N]\ \mathbf{in}\ \mathbf{return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i)\end{array}; \tau\right\rangle}.$$

Otherwise, $\mathcal{D}[N] \equiv \underline{\langle \mathcal{C}'[]|x.M'\rangle}_\eta$ implies that

$$\mathcal{D}[N] \equiv \left(\begin{array}{l}\mathbf{let}\ z = \mathbf{create}\ \{\lambda\_.\ \mathbf{let}\ x = \underline{\mathcal{C}'[M]}_\eta\ \mathbf{in}\ \mathbf{return}\ \{\lambda\_.\ \underline{M'}_\eta\}\}\ \mathbf{in} \\ \mathbf{let}\ zc = ref!\ \mathbf{inj}_{\mathrm{Zero}}\ ()\ \mathbf{in} \\ \mathbf{let}\ res = \mathbf{resume}\ z\ ()\ \mathbf{in} \\ res!\ (z, zc, \mathbf{inj}_{\mathrm{Zero}}\ ())\end{array}\right)$$

and, by the definition of contexts,

$$\mathcal{D} \equiv \left(\begin{array}{l}\mathbf{let}\ z = [\ ]\ \mathbf{in} \\ \mathbf{let}\ zc = ref!\ \mathbf{inj}_{\mathrm{Zero}}\ ()\ \mathbf{in} \\ \mathbf{let}\ res = \mathbf{resume}\ z\ ()\ \mathbf{in} \\ res!\ (z, zc, \mathbf{inj}_{\mathrm{Zero}}\ ())\end{array}\right),$$

$$N \equiv \mathbf{create}\ \{\lambda\_.\ \mathbf{let}\ x = \underline{\mathcal{C}'[M]}_\eta\ \mathbf{in}\ \mathbf{return}\ \{\lambda\_.\ \underline{M'}_\eta\}\};$$

however, $\mathbf{create}\ V$ does not appear in the right hand side of $\overset{\eta}{\underset{\kappa}{\sim}}$, which contradicts $\langle M; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau\rangle$. Thus, we obtain $\langle \mathcal{C}'[M]; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau\rangle$ and by the induction hypothesis, $\langle \mathcal{C}'; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'; \tau\rangle$, yielding that $\langle \mathcal{C}; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}; \tau\rangle$. This completes the proof.

Lemma 9. *Suppose that* $\langle \mathcal{C}[M]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau \rangle$ *and* $\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau \rangle$. *Then,* $\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$.

Proof. By induction on the derivation of $\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau \rangle$.

Proposition 2. *Suppose that* $\langle \mathcal{C}[M]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$ *and that* θ, τ, η, *and* κ *satisfy the IC. Then, there exist an evaluation context* $\mathcal{D}$, *a computation* N', *an extension of* τ', *and a partial function* κ' *such that*

$$\langle N; \tau \rangle \to^*_{\mathbf{AC}} \langle \mathcal{D}[N']; \tau' \rangle,$$
$$\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau' \rangle,$$
$$\langle M; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N'; \tau' \rangle,$$
$$\langle \mathcal{C}[M]; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle \mathcal{D}[N']; \tau' \rangle,$$

where τ' *is a finite extension of* τ, κ' *is a finite extension of* κ, *and* θ, τ', η, *and* κ' *satisfy the IC. Moreover, if* M *is a redex, so is* N'.

Proof. We prove this by induction on the number of computational frames that constitute $\mathcal{C}$. The base case follows from Lemma 6.

For the inductive step, we first show that we may suppose without loss of generality that $\mathcal{C} \equiv \langle \mathcal{C}'[] | x.M' \rangle$, where the outermost frame is computational. If, instead, $\mathcal{C} \equiv \mathcal{P}[\mathcal{C}'[]]$ for some pure frame $\mathcal{P}$, then we can decompose $\mathcal{C}$ and $\mathcal{C}'$ as

$$\mathcal{C} \equiv \mathcal{P}[\mathcal{H}[\langle \mathcal{C}''[] | x.M' \rangle]] \quad \text{and} \quad \mathcal{C}' \equiv \mathcal{H}[\langle \mathcal{C}''[] | x.M' \rangle],$$

where $\mathcal{C}''$ is a possibly computational context and $\mathcal{H}$ is a pure context. First, by applying the argument from the base case, we obtain

$$N \equiv \left(\underline{\mathcal{P}[\mathcal{H}]}_{\eta}\right)[N']$$
$$\langle \langle \mathcal{C}''[M] | x.M' \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N'; \tau \rangle,$$
$$\langle \mathcal{P}[\mathcal{H}[\langle \mathcal{C}''[M] | x.M' \rangle]]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathcal{P}[\mathcal{H}]}_{\eta}[N']; \tau \right\rangle.$$

Then, by treating $\langle \mathcal{C}''[] | x.M' \rangle$ as C, we reduce this case to one in which the outermost frame of $\mathcal{C}$ is computational, and we obtain

$$\langle N'; \tau \rangle \to^*_{\mathbf{AC}} \langle \mathcal{D}'[N'']; \tau' \rangle \quad \text{and} \quad \langle \langle \mathcal{C}''[M] | x.M' \rangle; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle \mathcal{D}'[N'']; \tau' \rangle,$$

where θ, τ', η, and κ' satisfy the IC. Note that if M is a redex, then so is N''. Applying Lemma 4 implies $\langle \mathcal{P}[\mathcal{H}[]]; \theta \rangle \overset{\eta}{\underset{\kappa''}{\sim_{\mathsf{c}}}} \left\langle \underline{\mathcal{P}[\mathcal{H}[]]}_{\eta}; \tau' \right\rangle$. Finally, from Lemma 5, we conclude

$$\langle \mathcal{P}[\mathcal{H}[\langle \mathcal{C}''[M] | x.M' \rangle]]; \theta \rangle \overset{\eta}{\underset{\kappa''}{\sim}} \left\langle \underline{\mathcal{P}[\mathcal{H}[]]}_{\eta}[\mathcal{D}'[N'']]; \tau'' \right\rangle,$$

yielding the desired result.

Now we return to the scenario where $C \equiv \langle \mathcal{C}'[]|x.M'\rangle$. According to the definition of $\overset{\eta}{\underset{\kappa}{\sim}}$, the derivation tree of $\langle\langle \mathcal{C}'[M]|x.M'\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$ has two possible forms:

Case 1:

$$\frac{\langle \mathcal{C}'[M];\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N';\tau\rangle \qquad \mathrm{get}(\kappa(m),\tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m,\kappa(m),\_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle\langle \mathcal{C}'[M]|x.M'\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = N'\ \mathbf{in\ return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i) \end{pmatrix};\tau \right\rangle}$$

Then, by the induction hypothesis, we obtain

$$\langle N';\tau\rangle \to^*_{\mathbf{AC}} \langle \mathcal{D}[N'']';\tau'\rangle,$$
$$\langle \mathcal{C}';\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim_{\mathsf{C}}}} \langle \mathcal{D};\tau'\rangle,$$
$$\langle M;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N'';\tau'\rangle,$$
$$\langle \mathcal{C}'[M];\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle \mathcal{D}[N'']';\tau'\rangle,$$

where τ' is a finite extension of τ, κ' is a finite extension of κ, and θ, τ', η, and κ' satisfy the IC. Note that N'' is a redex if M is also a redex. From (B.3), then, we have

$$\langle N;\tau\rangle \to^*_{\mathbf{AC}} \left\langle \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}[N'']\ \mathbf{in\ return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i) \end{pmatrix};\tau' \right\rangle.$$

Since τ' and κ' are finite extensions, it follows that $\mathrm{get}(\kappa'(m),\tau') = i$, $\tau'(m) = \mathbf{nil}$, and $\eta^{-1}(m,\kappa'(m),\_) \subseteq \theta^{-1}(\mathbf{nil})$. Thus, by (B.3), we obtain

$$\langle\langle \mathcal{C}'[]|x.M'\rangle;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim_{\mathsf{C}}}} \left\langle \begin{matrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}[]\ \mathbf{in\ return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m,\kappa'(m),i) \end{matrix} ;\tau' \right\rangle,$$

and by (B.3),

$$\langle\langle \mathcal{C}'[M]|x.M'\rangle;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \left\langle \begin{matrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}[N'']\ \mathbf{in\ return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m,\kappa'(m),i) \end{matrix} ;\tau' \right\rangle,$$

which completes the proof for the current case.

Case 2:

$$\langle\langle M_1|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \underline{\langle M_1|x.M_2\rangle}_\eta;\tau \right\rangle$$

$\langle N;\tau\rangle$ evaluates to

$$\left\langle N \equiv \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = N'\ \mathbf{in\ return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m,mc,i) \end{pmatrix};\tau' \right\rangle,$$

where τ' is $\tau[m := \mathbf{nil}, mc := \mathit{RefCell}(\mathbf{inj}_{\mathrm{Zero}}\,())]$. Let κ' be $\kappa[m := mc]$. Then, if θ, τ', η, and κ' satisfy the IC, this case is reducible to the previous case. Therefore, it remains to verify the IC.

(4) κ' is injective since mc is taken as a fresh label.
(5) $\mathrm{Dom}(\kappa') = \mathrm{Dom}(\kappa) \sqcup \{m\}$ and $\mathrm{Im}(\kappa') = \mathrm{Im}(\kappa) \sqcup \{mc\}$, so

$$\begin{aligned}\mathrm{Dom}(\kappa') \cap \mathrm{Im}(\kappa') &= (\mathrm{Dom}(\kappa) \cap \mathrm{Im}(\kappa)) \cup (\mathrm{Dom}(\kappa) \cap \{mc\}) \cup (\{m\} \cap \mathrm{Im}(\kappa)) \cup (\{m\} \cap \{mc\}) \\ &= \emptyset,\end{aligned}$$

since m and mc are taken as fresh labels.
(6) With regard to $\mathrm{Dom}(\tau')$,

$$\begin{aligned}\mathrm{Dom}(\tau') &= \mathrm{Dom}(\tau) \sqcup \{m, mc\} \\ &= (\mathrm{Dom}(\kappa) \sqcup \mathrm{Im}(\kappa)) \sqcup \{m, mc\} \\ &= (\mathrm{Dom}(\kappa) \sqcup \{m\}) \sqcup (\mathrm{Im}(\kappa) \sqcup \{mc\}) \\ &= \mathrm{Dom}(\kappa') \sqcup \mathrm{Im}(\kappa').\end{aligned}$$

The other conditions remain true since θ, τ, η, κ satisfy the IC and θ and η are not extended.

This completes the proof.

Proposition 3. *Suppose that $C \sim D$ and $C \to^{\beta}_{\mathbf{D}} C'$, then there exists an* **AC***-configuration D' such that $D \to^{+}_{\mathbf{AC}} D'$ and $C' \sim D'$.*

Proof. Suppose that $C \equiv \langle M; \theta \rangle$. By the definition of $\sim$, there exist N, τ, η, and κ such that

$$\begin{aligned} D &\equiv \langle N; \tau \rangle, \\ C &\overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle, \end{aligned}$$

where D is well-formed and θ, τ, η, and κ satisfy the IC. We analyze each case based on the definition of $C \to^{\beta}_{\mathbf{D}} C'$.

Case (**MAM**):

In this case, we present only one case since the other cases are similar. Suppose that

$$M \equiv (\mathbf{case}\ (V_1, V_2)\ \mathbf{of}\ (x_1, x_2) \mapsto M').$$

Then, $C \equiv \langle M; \theta \rangle$ evaluates to $C' \equiv \langle (M'[V_1/x_1, V_2/x_2]); \theta \rangle$. By the definition of $\overset{\eta}{\underset{\kappa}{\sim}}$, we obtain

$$\begin{aligned} N &\equiv \Big(\underline{\mathbf{case}\ (V_1, V_2)\ \mathbf{of}\ (x_1, x_2) \mapsto M'}_{\eta}\Big) \\ &\equiv \Big(\mathbf{case}\ (\underline{V_1}_{\eta}, \underline{V_2}_{\eta})\ \mathbf{of}\ (\underline{x_1}_{\eta}, \underline{x_2}_{\eta}) \mapsto \underline{M'}_{\eta}\Big). \end{aligned}$$

$\langle N; \tau \rangle$ is reduced in one step to

$$\left\langle \underline{M'}_\eta[\underline{V_1}_\eta/x_1, \underline{V_2}_\eta/x_2]; \tau \right\rangle \equiv \left\langle \underline{M'[V_1/x_1, V_2/x_2]}_\eta; \tau \right\rangle,$$

where the equation follows from Lemma 2. Then, Lemma 3 implies that

$$\langle M'[V_1/x_1, V_2/x_2]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{M'[V_1/x_1, V_2/x_2]}_\eta; \tau \right\rangle,$$

and Proposition 1 implies that the right-hand-side configuration is well-formed. Since θ, τ, η, and κ satisfy the IC, it follows that

$$\langle M'[V_1/x_1, V_2/x_2]; \theta \rangle \sim \left\langle \underline{M'[V_1/x_1, V_2/x_2]}_\eta; \tau \right\rangle,$$

which completes the proof for this case.

Case (throw):

Suppose that $M \equiv (\mathbf{throw}\ l\ V)$ and $\theta(l) = (\lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y] | x.M \rangle)$. Then, $\langle M; \theta \rangle$ evaluates to

$$\langle \langle \mathcal{H}[\mathbf{return}\ V] | x.M \rangle; \theta' \rangle,$$

where $\theta' = \theta[l \coloneqq \mathbf{nil}]$. By the definition of $\overset{\eta}{\underset{\kappa}{\sim}}$, we obtain

$$\begin{aligned} N &\equiv \underline{\mathbf{throw}\ l\ V}_\eta \\ &\equiv \left(\begin{array}{l} \mathbf{case}\ \underline{l}_\eta\ \mathbf{of}\ \{ \\ \quad ((z, zc), i) \mapsto \\ \quad\quad \mathbf{let}\ j = \mathit{get}!\ zc\ \mathbf{in} \\ \quad\quad \mathbf{let}\ b = \mathit{compare}!\ i\ j \\ \quad\quad \mathbf{case}\ b\ \mathbf{of}\ \{ \\ \quad\quad\quad (\mathbf{inj}_{\mathrm{True}}\ ()) \mapsto \\ \quad\quad\quad\quad \mathbf{let}\ i' = \mathit{incr}!\ i\ \mathbf{in} \\ \quad\quad\quad\quad \mathbf{let}\ () = \mathit{set}!\ zc\ i'\ \mathbf{in} \\ \quad\quad\quad\quad \mathbf{let}\ \mathit{res} = \mathbf{resume}\ z\ \underline{V}_\eta\ \mathbf{in} \\ \quad\quad\quad\quad \mathit{res}!\ ((z, zc), i') \\ \quad\quad\quad (\mathbf{inj}_{\mathrm{False}}\ ()) \mapsto \mathit{fail}! \\ \quad\quad \} \\ \} \end{array} \right). \end{aligned}$$

Let $m = \mathrm{pr}_1(l)$ and $i = \mathrm{get}(\mathrm{pr}_2(l), \tau)$, then, since θ, τ η, and κ satisfy the IC, we obtain

$$\begin{gathered} \mathrm{pr}_2(l) = \kappa(l), \\ i = \mathrm{pr}_3(l), \\ \tau(m) = \left\{ \lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_\eta[\mathbf{return}\ y]\ \mathbf{in}\ \mathbf{return}\ \left\{ \lambda \_.\ \underline{M}_\eta \right\} \right\}. \end{gathered}$$

Using these facts, the evaluation of $\langle N; \tau \rangle$ proceeds as follows:

$$\begin{aligned}
\langle N; \tau \rangle &\to^*_{\mathbf{AC}} \left\langle \begin{pmatrix} \textbf{let } i' = \mathit{incr}!\ i \textbf{ in} \\ \textbf{let } () = \mathit{set}!\ \kappa(m)\ i' \textbf{ in} \\ \textbf{let } \mathit{res} = \textbf{resume } m\ \underline{V}_\eta \textbf{ in} \\ \mathit{res}!\ ((m, \kappa(m)), i') \end{pmatrix}; \tau \right\rangle \\
&\to^*_{\mathbf{AC}} \left\langle \begin{pmatrix} \textbf{let } \mathit{res} = \textbf{resume } m\ \underline{V}_\eta \textbf{ in} \\ \mathit{res}!\ ((m, \kappa(m)), i+1) \end{pmatrix}; \tau[\kappa(m) \coloneqq \mathit{RefCell}(i+1)] \right\rangle \\
&\to_{\mathbf{AC}} \left\langle \begin{pmatrix} \textbf{let } \mathit{res} = m : \big(\textbf{let } x = \underline{\mathcal{H}}_\eta\big[\textbf{return } \underline{V}_\eta\big] \textbf{ in return } \big\{\lambda_.\ \underline{M}_\eta\big\}\big) \textbf{ in} \\ \mathit{res}!\ ((m, \kappa(m)), i+1) \end{pmatrix}; \tau' \right\rangle
\end{aligned}$$

where $\tau' \coloneqq \tau[m \coloneqq \textbf{nil}, \kappa(m) \coloneqq \mathit{RefCell}(i+1)]$. It is straightforward to check that $\underline{\mathcal{H}}_\eta\big[\textbf{return } \underline{V}_\eta\big] = \underline{\mathcal{H}[\textbf{return } V]}_\eta$. Thus, by Lemma 3, we obtain

$$\big\langle \underline{\mathcal{H}}_\eta\big[\textbf{return } \underline{V}_\eta\big]; \theta' \big\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \big\langle \underline{\mathcal{H}[\textbf{return } V]}_\eta; \tau' \big\rangle.$$

Now, in order to derive

$$\langle \langle \mathcal{H}[\textbf{return } V] | x.M \rangle; \theta' \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{pmatrix} \textbf{let } \mathit{res} = m : \Big(\textbf{let } x = \underline{\mathcal{H}[\textbf{return } V]}_\eta \textbf{ in return } \big\{\lambda\_.\ \underline{M}_\eta\big\}\Big) \textbf{ in} \\ \mathit{res}!\ ((m, \kappa(m)), i+1) \end{pmatrix}; \tau' \right\rangle,$$

we shall check that the other premises hold.

- $\mathrm{get}(\kappa(m), \tau') = i+1$ since $\tau'(\kappa(m)) = \mathit{RefCell}(i+1)$.
- Obviously, $\tau'(m) = \textbf{nil}$.
- Let l' be a continuation label of $\mathbf{DEL}_{\mathrm{one}}$ and suppose that $\mathrm{pr}_1(\eta(l')) = m$ and $\mathrm{pr}_2(\eta(l')) = \kappa(m)$. We shall prove that $\theta'(l') = \textbf{nil}$. If $l' = l$, then $\theta'(l) = \textbf{nil}$. Otherwise, since θ, τ, η, and κ satisfy the IC, $\theta(l) \neq \textbf{nil}$ implies that $\theta(l') = \textbf{nil}$; hence we obtain $\theta'(l') = \textbf{nil}$.

Thus, we obtain

$$\langle \langle \mathcal{H}[\textbf{return } V] | x.M \rangle; \theta' \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{pmatrix} \textbf{let } \mathit{res} = m : \Big(\textbf{let } x = \underline{\mathcal{H}[\textbf{return } V]}_\eta \textbf{ in return } \big\{\lambda\_.\ \underline{M}_\eta\big\}\Big) \textbf{ in} \\ \mathit{res}!\ ((m, \kappa(m)), i+1) \end{pmatrix}; \tau' \right\rangle.$$

Furthermore, Proposition 1 shows that the right-hand-side configuration is well-formed. Finally, we check that the IC is preserved through the evaluation. However, we present only non-trivial conditions for brevity.

6. $\mathrm{Dom}(\tau') = \mathrm{Dom}(\tau) = \mathrm{Dom}(\kappa) \sqcup \mathrm{Im}(\kappa)$.
7. $\mathrm{Dom}(\eta) = \mathrm{Dom}(\theta) = \mathrm{Dom}(\theta')$.
9. Suppose that $\theta'(l') = \textbf{nil}$. If $l' \equiv l$, then $\mathrm{get}(\mathrm{pr}_2(\eta(l)), \tau') = i+1 > i = \mathrm{pr}_3(\eta(l))$. Otherwise, the previous IC implies that $\mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau) > \mathrm{pr}_3(\eta(l'))$. If $\mathrm{pr}_2(\eta(l')) = \kappa(m)$, then $\mathrm{get}(\kappa(m), \tau') = i+1 > i = \mathrm{get}(\kappa(m), \tau') > \mathrm{pr}_3(\eta(l'))$. If $\mathrm{pr}_2(\eta(l')) \neq \kappa(m)$, then the previous IC yields that $\mathrm{pr}_2(\eta(l')) \neq m$. Hence, $\mathrm{pr}_2(\eta(l')) \in \mathrm{Dom}(\tau)$, yielding $\mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau') = \mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau) > \mathrm{pr}_3(\eta(l'))$.

10. Suppose that $\theta'(l') \neq \mathbf{nil}$. Then, we obtain $l' \neq l$ and $\theta'(l') = \theta(l') = \lambda y.\, \langle \mathcal{G}[\mathbf{return}\ y] | x.S \rangle$ for some $\mathcal{G}$ and S.
    (a) The previous IC yields that l' does not appear in S.
    (b) We shall argue that $\mathrm{pr}_2(\eta(l')) \neq \kappa(m)$. Suppose otherwise. Then, the previous IC implies that $\mathrm{pr}_1(\eta(l')) = m$ and $\theta(l') = \mathbf{nil}$, which is a contradiction. Hence, $\mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau') = \mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau) = \mathrm{pr}_3(\eta(l))$.
    (c) Let $l'' \in \mathrm{Dom}(\theta')$ and suppose that $l'' \neq l'$ and $\mathrm{pr}_1(\eta(l'')) = \mathrm{pr}_1(\eta(l'))$. If $l'' \equiv l$, then the previous IC yields that $\theta(l') = \mathbf{nil}$, which contradicts the assumption. Hence, $l'' \neq l$, which immediately implies that $\theta'(l'') = \theta(l'') = \mathbf{nil}$. Note that the last equation follows from the previous IC.
    (d) This directly follows from the previous IC since $\mathrm{pr}_1(\eta(l')) \neq m$ and $\mathrm{pr}_2(\eta(l')) \neq \kappa(m)$.

This completes the proof for the current case.

Case (fail):

Suppose that $M \equiv (\mathbf{throw}\ l\ V)$ and $\theta(l) = \mathbf{nil}$. Then, $\langle M; \theta \rangle$ evaluates to $\bot$. By the definition of $\overset{\eta}{\underset{\kappa}{\sim}}$, we obtain

$$
\begin{aligned}
N &\equiv \underline{\mathbf{throw}\ l\ V}_{\eta} \\
&\equiv \left(\begin{array}{l}
\mathbf{case}\ \underline{l}_{\eta}\ \mathbf{of}\ \{ \\
\quad ((z, zc), i) \mapsto \\
\qquad \mathbf{let}\ j = \mathit{get}!\ zc\ \mathbf{in} \\
\qquad \mathbf{let}\ b = \mathit{compare}!\ i\ j \\
\qquad \mathbf{case}\ b\ \mathbf{of}\ \{ \\
\qquad\quad (\mathbf{inj}_{\mathrm{True}}\ ()) \mapsto \\
\qquad\qquad \mathbf{let}\ i' = \mathit{incr}!\ i\ \mathbf{in} \\
\qquad\qquad \mathbf{let}\ () = \mathit{set}!\ zc\ i'\ \mathbf{in} \\
\qquad\qquad \mathbf{let}\ \mathit{res} = \mathbf{resume}\ z\ \underline{V}_{\eta}\ \mathbf{in} \\
\qquad\qquad \mathit{res}!\ ((z, zc), i') \\
\qquad\quad (\mathbf{inj}_{\mathrm{False}}\ ()) \mapsto \mathit{fail}! \\
\qquad \} \\
\}
\end{array}\right).
\end{aligned}
$$

Let $m = \mathrm{pr}_1(l)$ and $j = \mathrm{get}(\mathrm{pr}_2(l), \tau)$, then, since θ, τ η, and κ satisfy the IC, we obtain

$$\mathrm{pr}_2(l) = \kappa(l) \quad \text{and} \quad j > \mathrm{pr}_3(l).$$

Using this, the evaluation of $\langle N; \tau \rangle$ proceeds as follows:

$$
\begin{aligned}
\langle N; \tau \rangle &\to^*_{\mathbf{AC}} \langle \mathit{fail}!; \tau \rangle \\
&\to^*_{\mathbf{AC}} \bot.
\end{aligned}
$$

Hence, we obtain $\bot \sim \bot$. This complete the proof.

Now, the remaining cases are for (ret) and (yield), where the computation M is enclosed within a dollar term, i.e., $M \equiv \langle M_1 | x.M_2 \rangle$. In such instances, there are two possible derivations for $\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$:

$$\langle \langle M_1 | x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \left(\underline{\langle M_1 | x.M_2 \rangle}_\eta \right); \tau \right\rangle,$$

or

$$\frac{\begin{array}{c} \langle M_1; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau \rangle \\ \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil}) \end{array}}{\langle \langle M_1 | x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in\ return}\ \left\{ \lambda\_.\ \underline{M_2}_\eta \right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{pmatrix}; \tau \right\rangle}.$$

Then, by applying Proposition 2 with $\mathcal{C}[\langle [\ ] | x.M_2 \rangle]$ as the instance of $\mathcal{C}$, the former case reduces to the latter. Thus, for the remaining cases, we restrict our attention to the situation where

$$N \equiv \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in\ return}\ \left\{ \lambda\_.\ \underline{M_2}_\eta \right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{pmatrix}.$$

Case (ret):

Suppose that $M \equiv \langle \mathbf{return}\ V | x.M' \rangle$. Then, $\langle M; \theta \rangle$ is reduced to $\langle M'[V/x]; \theta \rangle$. As noted previously, we shall assume without loss of generality that the derivation of $\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$ is

$$\frac{\begin{array}{c} \langle \mathbf{return}\ V; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathbf{return}\ \underline{V}_\eta; \tau \rangle \\ \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil}) \end{array}}{\langle \langle \mathbf{return}\ V | x.M' \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{array}{l} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathbf{return}\ \underline{V}_\eta\ \mathbf{in\ return}\ \left\{ \lambda\_.\ \underline{M'}_\eta \right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{array} ; \tau \right\rangle}$$

$\langle N; \tau \rangle$ is evaluated as follows.

$$\begin{aligned} \langle N; \tau \rangle &\to_{\mathbf{AC}} \left\langle \left(\left\{ \lambda\_.\ \underline{M'}_\eta[\underline{V}_\eta/x] \right\}! \ (m, \kappa(m), i) \right); \tau \right\rangle \\ &\to_{\mathbf{AC}} \left\langle \underline{M'}_\eta[\underline{V}_\eta/x]; \tau \right\rangle \\ &\equiv \left\langle \underline{M'[V/x]}_\eta; \tau \right\rangle \qquad \text{by Lemma 2.} \end{aligned}$$

From Lemma 3, we obtain

$$\langle M'[V/x]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{M'[V/x]}_\eta; \tau \right\rangle,$$

where Proposition 1 implies that the configuration on the right-hand side is well-formed. Since θ, τ, η, and κ satisfy the IC, we conclude that

$$\langle M'[V/x]; \theta \rangle \sim \left\langle \underline{M'[V/x]}_\eta; \tau \right\rangle.$$

Case (shift):

Suppose that $M \equiv \langle \mathcal{H}[\mathbf{S_0}k.\ M_1] | x.M_2 \rangle$. Then, $\langle M; \theta \rangle$ evaluates to

$$\langle M_1[l/k]; \theta' \rangle,$$

where l is a fresh continuation label and θ' is $\theta[l \coloneqq \lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y] | x.M_2 \rangle]$. Conducting case analysis implies that the derivation tree of $\langle \langle \mathcal{H}[\mathbf{S_0}k.\ M_1] | x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$ is

$$\frac{\begin{array}{c} \langle \mathcal{H}[\mathbf{S_0}k.\ M_1]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau \rangle \\ \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil}) \end{array}}{\langle \langle \mathcal{H}[\mathbf{S_0}k.\ M_1] | x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \left(\begin{array}{l} \mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in}\ \mathbf{return} \left\{ \lambda\_.\ \underline{M_2}_{\eta} \right\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{array} \right); \tau \right\rangle}.$$

Then, applying Lemma 6 to $\langle \mathcal{H}[\mathbf{S_0}k.\ M_1]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1; \tau \rangle$ implies that there is an **AC**-computation N_1' such that

$$N_1 \equiv \underline{\mathcal{H}}_{\eta}[N_1'] \quad \text{and} \quad \langle \mathbf{S_0}k.\ M_1; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1'; \tau \rangle.$$

Moreover, by performing case inversion on $\langle \mathbf{S_0}k.\ M_1; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1'; \tau \rangle$, we obtain

$$N_1' \equiv \underline{\mathbf{S_0}k.\ M_1}_{\eta} \equiv \mathbf{yield} \left\{ \lambda k.\ \underline{M_1}_{\eta} \right\}.$$

Thus, the evaluation of $\langle N; \tau \rangle$ is as follows:

$$\begin{aligned} \langle N; \tau \rangle &\to^*_{\mathbf{AC}} \left\langle \left(\left\{ \lambda k.\ \underline{M_1}_{\eta} \right\}! \ (m, \kappa(m), i) \right); \tau' \right\rangle \\ &\to^*_{\mathbf{AC}} \left\langle \underline{M_1}_{\eta}[(m, \kappa(m), i)/k]; \tau' \right\rangle, \end{aligned}$$

where

$$\tau' \coloneqq \tau \left[m \coloneqq \left\{ \lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_{\eta}[\mathbf{return}\ y]\ \mathbf{in}\ \mathbf{return} \left\{ \lambda\_.\ \underline{M_2}_{\eta} \right\} \right\} \right].$$

Thus, defining η' to be $\eta[l \coloneqq (m, \kappa(m), i)]$ yields

$$\underline{M_1}_{\eta}[(m, \kappa(m), i)/k] \equiv \underline{M_1}_{\eta'}[(m, \kappa(m), i)/k] \equiv \underline{M_1}_{\eta'}[\eta'(l)/k] \equiv \underline{M_1[l/k]}_{\eta'},$$

where the first equation follows since M_1 does not contain l, which is taken as a fresh label, and the last equation follows from Lemma 2. Therefore by Lemma 3, we have

$$\langle M_1[l/k]; \theta' \rangle \overset{\eta'}{\underset{\kappa}{\sim}} \left\langle \underline{M_1[l/k]}_{\eta'}; \tau' \right\rangle.$$

Moreover, Proposition 1 shows that $\to^*_{\mathbf{AC}} \left\langle \underline{M_1}_{\eta}[(m, \kappa(m), i)/k]; \tau' \right\rangle$ is well-formed. Then, it remains to show that θ', τ', η', and κ satisfy the IC. We verify only the non-trivial conditions here; the remaining conditions hold automatically as θ, τ, η, and κ already satisfy the IC.

6. $\mathrm{Dom}(\kappa) \sqcup \mathrm{Im}(\kappa) = \mathrm{Dom}(\tau) = \mathrm{Dom}(\tau')$.
7. $\mathrm{Dom}(\eta') = \mathrm{Dom}(\eta) \cup \{l\} = \mathrm{Dom}(\theta) \cup \{l\} = \mathrm{Dom}(\theta')$.
8. Let $l' \in \mathrm{Dom}(\eta')$ and suppose that $\eta'(l') = (z, zc, i)$. If $l' \in \mathrm{Dom}(\eta)$, then we immediately see that $z \in \mathrm{Dom}(\kappa)$ and $\kappa(z) = zc$. If $l' \equiv l$, then, $l \in \mathrm{Dom}(\theta)$, $(z, zc, i) = (m, \kappa(m), i)$, and $m \in \mathrm{Dom}(\kappa)$ since
$$\mathrm{Dom}(\tau) \subset \mathrm{Dom}(\kappa) \quad \text{and} \quad m \in \mathrm{Dom}(\tau).$$
9. Suppose that $l' \in \mathrm{Dom}(\eta')$ and $\theta(l') = \mathbf{nil}$. Then, since $' \in \mathrm{Dom}(\theta)$,
$$\mathrm{get}(\mathrm{pr}_2(\eta'(l')), \tau') = \mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau) < \mathrm{pr}_3(\eta(l')) < \mathrm{pr}_3(\eta'(l')).$$
10. Suppose that $l' \in \mathrm{Dom}(\eta')$ and $\theta'(l') = \lambda y.\ \langle \mathcal{G}[\mathbf{return}\ y] | x.S \rangle$. First, assume that $l' \in \mathrm{Dom}(\theta)$. Then, from the previous IC, it follows that l' is not contained in S and that
$$\mathrm{get}(\mathrm{pr}_2(\eta'(l')), \tau') = \mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau') = \mathrm{get}(\mathrm{pr}_2(\eta(l')), \tau) = \mathrm{pr}_3(\eta(l')) = \mathrm{pr}_3(\eta'(l')),$$
where the second equation holds since τ' is an extension of τ and $\mathrm{pr}_2(\eta(l')) \in \mathrm{Dom}(\tau)$. Next, suppose that $l'' \in \mathrm{Dom}(\theta')$, $l'' \neq l'$, and $\mathrm{pr}_1(\eta(l'')) = \mathrm{pr}_1(\eta(l'))$. Then, if $l'' \in \mathrm{Dom}(\theta)$, it immediately follows that
$$\theta'(l'') = \theta(l'') = \mathbf{nil}.$$
If, instead, $l'' \in \mathrm{Dom}(\theta') \setminus \mathrm{Dom}(\theta)$, i.e., $l'' = l$, by one of the premises for deriving $\langle S; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N'; \tau \rangle$, we see that
$$\eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil}).$$
Moreover, since $\mathrm{pr}_1(\eta(l')) = \mathrm{pr}_1(\eta(l)) = m$, we also obtain $\mathrm{pr}_2(\eta(l')) = \kappa(m)$. Thus, we see that $\theta(l') = \mathbf{nil}$, but this is contradiction. Finally, we have $\tau'(\eta'(l')) = \tau'(\eta(l')) = \tau(\eta(l')) = \lambda y.\ \left\langle \underline{\mathcal{G}}_{\eta}[\mathbf{return}\ y] \middle| x.\underline{S}_{\eta} \right\rangle$. Here, $\mathcal{G}$ and S do not contain l since it is taken as a fresh label. Hence, $\underline{\mathcal{G}}_{\eta} = \underline{\mathcal{G}}_{\eta'}$ and $\underline{S}_{\eta} = \underline{S}_{\eta'}$, which implies
$$\tau'(\eta'(l')) = \lambda y.\ \left\langle \underline{\mathcal{G}}_{\eta'}[\mathbf{return}\ y] \middle| x.\underline{S}_{\eta'} \right\rangle.$$
Now, assume that $l' = l$. Since, l is taken as a fresh label, it does not appear in M_2. Then, the second condition is ensured by the one of premises of $\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N'; \tau \rangle$:
$$\mathrm{get}(\kappa(m), \tau) = i$$
The second condition can be also verified as in the case in which $l' \in \mathrm{Dom}(\theta)$. As for the last condition, we see that
$$\begin{aligned} \tau'(\mathrm{pr}_1(\eta(l'))) &= \left\{ \lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_{\eta}[\mathbf{return}\ y]\ \mathbf{in}\ \mathbf{return}\ \left\{ \lambda \_.\ \underline{M_2}_{\eta} \right\} \right\} \\ &= \left\{ \lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_{\eta'}[\mathbf{return}\ y]\ \mathbf{in}\ \mathbf{return}\ \left\{ \lambda \_.\ \underline{M_2}_{\eta'} \right\} \right\}. \end{aligned}$$

This completes the proof for the current case.

Proposition 4. *Suppose that*

$$\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau \rangle,$$
$$\langle M; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle,$$
$$\langle \mathcal{C}[M]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau \rangle,$$
$$\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} \langle M'; \theta' \rangle,$$

where $\langle \mathcal{D}[N]; \tau \rangle$ is well-formed and θ, τ, η, and κ satisfy the IC. Then, there exist an **AC***-computation N', an* **AC***-store τ', partial functions η' and κ' such that*

$$\langle N; \tau \rangle \to^{+}_{\mathbf{AC}} \langle N'; \tau' \rangle,$$
$$\langle M'; \theta' \rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle N'; \tau' \rangle,$$
$$\langle \mathcal{C}[M']; \theta' \rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle \mathcal{D}[N']; \tau' \rangle.$$

Moreover, from Proposition 1, $\langle \mathcal{D}[N']; \tau' \rangle$ is well-formed.

Proof. We prove by induction on the derivation of $\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau \rangle$. The base case follows immediately from Lemma 3.

For the inductive step, we suppose, without loss of generality, that the outermost frame of C is computational; i.e., $\mathcal{C} \equiv \langle \mathcal{C}'[]|x.M_2 \rangle$, which is justified by Lemma 5 and Lemma 6. Suppose that $\langle \langle \mathcal{C}'[M_1]|x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau \rangle$, $\langle M_1; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau \rangle$, and $\langle M_1; \theta \rangle \to^{\beta}_{\mathbf{D}} \langle M_1'; \theta' \rangle$. As noted in the proof of Lemma 3, or alternatively by applying Proposition 2, we shall assume that the derivation of $\langle \langle \mathcal{C}'[M_1]|x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}[N]; \tau \rangle$ is

$$\frac{\langle \mathcal{C}'[M_1]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau \rangle \qquad \mathrm{get}(\kappa(m), \tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m, \kappa(m), \_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle \langle \mathcal{C}'[M_1]|x.M_2 \rangle; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{pmatrix} \textbf{let } res = m : (\textbf{let } x = \mathcal{D}'[N] \textbf{ in return } \left\{ \lambda\_.\ \underline{M_2}_{\eta} \right\}) \textbf{ in} \\ res!\ (m, \kappa(m), i) \end{pmatrix}; \tau \right\rangle}$$

Moreover, from Lemma 8, we obtain $\langle \mathcal{C}'; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim_{\mathsf{c}}}} \langle \mathcal{D}'; \tau \rangle$. Applying the induction hypothesis to $\langle \mathcal{C}'[M_1]; \theta \rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle \mathcal{D}'[N]; \tau \rangle$ implies that there exist N', τ', η', and κ' such that

$$\langle N; \tau \rangle \to^{+}_{\mathbf{AC}} \langle N'; \tau' \rangle$$
$$\langle \mathcal{C}'[M_1']; \theta' \rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle \mathcal{D}'[N']; \tau' \rangle,$$

where θ', τ', η', and κ' satisfy the IC. Then, we shall show that

$$\frac{\langle \mathcal{C}'[M_1'];\theta'\rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle \mathcal{D}'[N'];\tau'\rangle \qquad \mathrm{get}(\kappa'(m),\tau') = i \qquad \tau'(m) = \mathbf{nil} \qquad \eta'^{-1}(m,\kappa'(m),\_) \subseteq \theta'^{-1}(\mathbf{nil})}{\langle\langle \mathcal{C}'[M_1']|x.M_2\rangle;\theta'\rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \left\langle \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}'[N']\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{M_2}_{\eta'}\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i) \end{pmatrix}; \tau' \right\rangle}$$

is derivable by checking that that $\underline{M_2}_{\eta'} \equiv \underline{M_2}_{\eta}$ and that the premises other than $\langle \mathcal{C}'[M_1'];\theta'\rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle \mathcal{D}'[N'];\tau'\rangle$ are also satisfied.

Here, we conduct case analysis on $\langle M_1;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M_1';\theta'\rangle$ as follows.

Case (**MAM**):

In this case, none of θ, τ, η, and κ is updated. Hence, it is trivial to check that the premises are preserved.

Case (throw):

Suppose that $\langle M_1;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M_1';\theta'\rangle$ is

$$\langle \mathbf{throw}\ l\ V;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle\langle \mathcal{H}[\mathbf{return}\ V]|x.S\rangle;\theta'\rangle,$$

where

$$\begin{aligned} \theta(l) &= (\lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y]|x.S\rangle), \\ \theta' &\coloneqq \theta[l \coloneqq \mathbf{nil}]. \end{aligned}$$

From the proof of Lemma 3, we obtain

$$\begin{gathered} N \equiv \underline{\mathbf{throw}\ l\ V}_{\eta}, \\ N' \equiv \begin{pmatrix} \mathbf{let}\ res = n : \left(\mathbf{let}\ x = \underline{\mathcal{H}}_{\eta}\left[\mathbf{return}\ \underline{V}_{\eta}\right]\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{S}_{\eta}\right\}\right)\ \mathbf{in} \\ res!\ ((n,\kappa(n)),j+1) \end{pmatrix}, \\ \langle M_1';\theta'\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N';\tau'\rangle, \end{gathered}$$

where

$$\begin{aligned} m \neq n \quad &\text{(by Proposition 1)}, \\ \tau' \coloneqq \tau[n \coloneqq \mathbf{nil}, \kappa(n) &\coloneqq \mathit{RefCell}(j+1)], \end{aligned}$$

and θ', τ', η, and κ satisfy the IC. Then, we shall check that the premises are satisfied as follows:

- $\mathrm{get}(\kappa(m),\tau') = i$ since $m \neq n$ and $\tau'(\kappa(m)) = \mathit{RefCell}(i)$.
- $\tau'(m) = \tau(m) = \mathbf{nil}$.

– Let $l' \in \mathrm{Dom}(\eta)$ and suppose that $\mathrm{pr}_1(\eta(l')) = m$ and $\mathrm{pr}_2(\eta(l')) = \kappa(m)$. Then, since $m \neq n$, we see that $l' \neq l$ and $\theta'(l') = \theta(l') = \mathbf{nil}$, where the last equation follows from the previous IC.

Case (fail):

(fail) is not applicable here, as $\langle M; \theta\rangle$ reduces to a configuration $\langle M'; \theta'\rangle$, not $\bot$.

Now, the remaining cases for (ret) and (yield). Both of them assumes that the computation M_1 is enclosed within a dollar term: $M_1 \equiv \langle S_1 | x.S_2\rangle$. As in the proof of Proposition 3, we shall assume that the the derivation of $\langle M_1; \theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N; \tau\rangle$ is

$$\frac{\begin{array}{c}\langle S_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1;\tau\rangle \\ \mathrm{get}(\kappa(n),\tau) = n \qquad \tau(n) = \mathbf{nil} \qquad \eta^{-1}(n,\kappa(n),\_) \subseteq \theta^{-1}(\mathbf{nil})\end{array}}{\langle\langle S_1|x.S_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle N \equiv \left(\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{S_2}_\eta\right\}\ )\ \mathbf{in} \\ res!\ (n,\kappa(n),j)\end{array}\right);\tau\right\rangle},$$

not

$$\langle\langle S_1|x.S_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\langle S_1|x.S_2\rangle}_\eta;\tau\right\rangle.$$

This is because, the evaluation from $\left\langle \underline{\langle S_1|x.S_2\rangle}_\eta;\tau\right\rangle$ to

$$\left\langle N \equiv \left(\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = N_1\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{S_2}_\eta\right\}\ )\ \mathbf{in} \\ res!\ (n,\kappa(n),j)\end{array}\right);\tau\right\rangle$$

preserves not only the IC, but also the premises of

$$\frac{\begin{array}{c}\langle \mathcal{C}'[M_1'];\theta'\rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \langle \mathcal{D}'[N'];\tau'\rangle \\ \mathrm{get}(\kappa'(m),\tau) = i \qquad \tau'(m) = \mathbf{nil} \qquad \eta'^{-1}(m,\kappa'(m),\_) \subseteq \theta'^{-1}(\mathbf{nil})\end{array}}{\langle\langle \mathcal{C}'[M_1']|x.M_2\rangle;\theta'\rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \left\langle \left(\begin{array}{l}\mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}'[N']\ \mathbf{in\ return}\ \left\{\lambda\_.\ \underline{M_2}_{\eta'}\right\}\ )\ \mathbf{in} \\ res!\ (m,\kappa(m),i)\end{array}\right);\tau'\right\rangle},$$

as we have proven in the **Case 1** in the proof of Proposition 2.

Case (ret):

Suppose that $\langle M_1;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M_1';\theta'\rangle$ is

$$\langle M_1 \equiv \langle \mathbf{return}\ V|x.M'\rangle;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M'[V/x];\theta\rangle.$$

From the proof of Lemma 3, we see that none of θ, τ, η, and κ is updated, which implies that the premises are preserved.

Case (shift):

Suppose that $\langle M_1;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M_1';\theta'\rangle$ is

$$\langle\langle \mathcal{H}[\mathbf{S_0}k.\ S_1]|x.S_2\rangle;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle S_1[l/k];\theta'\rangle,$$

where l is a fresh continuation label, and θ' is $\theta[l \coloneqq \lambda y.\, \langle \mathcal{H}[\mathbf{return}\ y] | x.S_2 \rangle]$. From the proof of Lemma 3, we obtain

$$N \equiv \begin{pmatrix} \mathbf{let}\ res = n : \Big(\mathbf{let}\ x = \underline{\mathcal{H}}_\eta \Big[\mathbf{yield}\ \Big\{\lambda k.\ \underline{S_1}_\eta\Big\}\Big]\ \mathbf{in\ return}\ \Big\{\lambda\_.\ \underline{S_2}_\eta\Big\}\Big)\ \mathbf{in} \\ res!\ (n, \kappa(n), j) \end{pmatrix},$$

$$N' \equiv \underline{S_1}_\eta[(n, \kappa(n), j)/k],$$

$$\langle M_1'; \theta' \rangle \overset{\eta'}{\underset{\kappa}{\sim}} \langle N'; \tau' \rangle,$$

where

$$\tau' \coloneqq \tau\Big[n \coloneqq \Big\{\lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}}_\eta[\mathbf{return}\ y]\ \mathbf{in\ return}\ \Big\{\lambda\_.\ \underline{S_2}_\eta\Big\}\Big\}\Big],$$

$$\eta' \coloneqq \eta[l \coloneqq (n, \kappa(n), j)],$$

and θ', τ', η', and κ satisfy the IC. First, since η is extended on a fresh label l, we have that $\underline{M_2}_{\eta'} \equiv \underline{M_2}_\eta$. Moreover, the well-formedness implies that $m \neq n$. Then, we shall check that the premises are satisfied as follows:

- $\mathrm{get}(\kappa(m), \tau') = \mathrm{get}(\kappa(m), \tau) = i$.
- $\tau'(m) = \tau(m) = \mathbf{nil}$
- Let $l' \in \mathrm{Dom}(\eta')$ and suppose that $\mathrm{pr}_1(\eta'(l')) = m$ and $\mathrm{pr}_2(\eta'(l')) = \kappa(m)$. Since $m \neq n$, we have $l' \neq l$ and $l \in \mathrm{Dom}(\eta)$. Then, from the previous IC, we obtain $\theta(l) = \mathbf{nil}$. Therefore, $\theta'(l) = \theta(l) = \mathbf{nil}$.

Therefore, we obtain

$$\langle \langle \mathcal{C}'[M_1'] | x.M_2 \rangle; \theta' \rangle \overset{\eta'}{\underset{\kappa'}{\sim}} \left\langle \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = \mathcal{D}'[N']\ \mathbf{in\ return}\ \Big\{\lambda\_.\ \underline{M_2}_{\eta'}\Big\}\ )\ \mathbf{in} \\ res!\ (m, \kappa(m), i) \end{pmatrix}; \tau' \right\rangle,$$

Finally, we give the proof of Theorem 1.

Proof of Theorem 1. Suppose that $\langle \mathcal{C}[M]; \theta \rangle \sim D$ and $\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} C'$. Then, by definition, there exist N and τ such that $D \equiv \langle N; \tau \rangle$, which is well-formed. Moreover, by Proposition 2, there exist $\mathcal{D}$, N', τ', and κ' such that

$$\langle N; \tau \rangle \to^*_{\mathbf{AC}} \langle \mathcal{D}[N']; \tau' \rangle,$$

$$\langle \mathcal{C}; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim_{\mathsf{c}}}} \langle \mathcal{D}; \tau' \rangle,$$

$$\langle M; \theta \rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N'; \tau' \rangle,$$

where θ, τ', η, and κ' satisfy the IC, and Proposition 1 implies that $\langle \mathcal{D}[N']; \tau' \rangle$ is well-formed. Then, we consider two cases for the derivation of $\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} C'$.
Case 1:

$$\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} \bot$$

In this case, the reduction rule (fail) is applied to $\langle M;\theta\rangle$ since it is the only rule that introduces $\bot$. Therefore, from Lemma 3, we obtain $\langle N';\tau'\rangle \to^{\beta}_{\mathbf{D}} \bot$, which implies $\langle \mathcal{D}[N'];\tau'\rangle \to^{\beta}_{\mathbf{D}} \bot$, completing this case.

Case 2:

$$\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M';\theta'\rangle$$

This case follows directly by Lemma 4.

This completes the proof.

B.4 Proposition for proving strong macro-expressibility

To establish the strong macro-expressibility of $\mathbf{DEL}_{\mathrm{one}}$ in $\mathbf{AC}$, we need to prove the following proposition:

Proposition 5. *For any* $\mathbf{DEL}_{\mathrm{one}}$*-program* M*, if* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *terminates, then* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *also terminates.*

Before proving this proposition, we show four lemmas.

Lemma 10. *We say an* $\mathbf{DEL}_{\mathrm{one}}$*-configuration* $\langle M;\theta\rangle$ *is well-formed if for any continuation label* l *that appears in* M*,* $\theta(l)$ *is defined, and this property is preserved under reduction.*

Proof. By straightforward case analysis on M.

Lemma 11. *Suppose that* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *diverges, i.e., the reduction sequence of* M *is infinite. Then* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *also diverges.*

Proof. By applying Theorem 1 repeatedly, we also obtain an infinite reduction sequence of $\underline{M}$. Since the reduction of $\mathbf{AC}$ is deterministic, this implies that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges.

Lemma 12. *Suppose that* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *gets stuck: there exists a term* M' *and a store* θ *such that* $\langle M;\emptyset\rangle \to^{*}_{\mathbf{D}} \langle M';\theta\rangle$*,* $M' \neq \mathbf{return}\ V$ *for any value* V*, and there is no rule that can reduce* $\langle M';\theta\rangle$*. Then,* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *also gets stuck.*

Proof. It suffices to show the following statement:

> Suppose that there exist M, N, θ, τ, η, and κ such that $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$, $\langle M;\theta\rangle$ is well-formed, and θ, τ, η, and κ satisfy the IC. If $\langle M;\theta\rangle$ is stuck, then $\langle N;\tau\rangle$ will also be stuck: there exist N', τ', and κ' such that
> - $\langle N;\tau\rangle \to^{*}_{\mathbf{D}} \langle N';\tau'\rangle$,
> - $\langle M;\theta\rangle \overset{\eta}{\underset{\tau'}{\sim}} \langle N';\tau'\rangle$,
> - θ, τ', η, and τ' satisfy the IC, and
> - $\langle N';\tau'\rangle$ is stuck.

We prove by induction on $\langle M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N;\tau\rangle$. There are twelve cases to consider,[13] but we treat only five cases. The other cases follow by similar reasoning.

Case 1:

$$\langle \mathbf{case}\ V\ \mathbf{of}\ (x_1,x_2)\mapsto M;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1,x_2)\mapsto M}_{\eta};\tau\right\rangle$$

Since the left-hand configuration is stuck, we know that $V \neq (V_1, V_2)$ for any values V_1 and V_2. Consequently, $\underline{V}_\eta$ also cannot be of the form (W_1, W_2) for any values W_1 and W_2. Therefore, the configuration $\left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1,x_2)\mapsto M}_{\eta};\tau\right\rangle$ is stuck, and this concludes the current case.

Case 2:

$$\frac{\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1;\tau\rangle}{\langle \mathbf{let}\ x=M_1\ \mathbf{in}\ M_2;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \mathbf{let}\ x=N_1\ \mathbf{in}\ \underline{M_2}_{\eta};\tau\right\rangle}$$

By the induction hypothesis, there exist N_1', τ', and κ' such that $\langle N_1;\tau\rangle \to_{\mathbf{D}}^* \langle N_1';\tau'\rangle$, $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N_1';\tau'\rangle$, and $\langle N_1';\tau'\rangle$ is stuck. Since $\langle \mathbf{let}\ x=M_1\ \mathbf{in}\ M_2;\theta\rangle$ is stuck, M_1 cannot be of the form $\mathbf{return}\ V$ for any value V. Together with $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N_1';\tau'\rangle$, this implies that $N_1 \neq \mathbf{return}\ W$ for any value W. Consequently, the configuration $\left\langle \mathbf{let}\ x=N_1'\ \mathbf{in}\ \underline{M_2}_{\eta};\tau'\right\rangle$ is also stuck, which completes the current case.

Case 3:

$$\langle \mathbf{throw}\ V\ W;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{throw}\ V\ W}_{\eta};\tau\right\rangle$$

It follows that $V \neq l$ for any continuation label l, since if $V = l$ for some l, then by the well-formedness of $\langle \mathbf{throw}\ V\ W;\theta\rangle$, we know that $l \in \mathrm{Dom}(\theta)$. However, this contradicts that $\langle \mathbf{throw}\ V\ W;\theta\rangle$ is stuck. This implies that $\underline{\mathbf{throw}\ V\ W}_\eta$ is also stuck, which concludes the current case.

Case 4:

$$\langle\langle M_1|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\langle M_1|x.M_2\rangle}_{\eta};\tau\right\rangle$$

The right-hand configuration evaluates to

$$\left\langle \begin{pmatrix} \mathbf{let}\ res = m : (\mathbf{let}\ x = N'\ \mathbf{in}\ \mathbf{return}\ \{\lambda\_.\ \underline{M'}_\eta\}\ )\ \mathbf{in} \\ res!\ (m, mc, i) \end{pmatrix};\tau'\right\rangle,$$

which is reducible to the next case.

Case 5:

[13] $\langle \mathbf{return}\ V;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \underline{\mathbf{return}\ V}_{\eta};\tau\right\rangle$ is excluded by the premise.

$$\frac{\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \langle N_1;\tau\rangle \qquad \mathrm{get}(\kappa(m),\tau) = i \qquad \tau(m) = \mathbf{nil} \qquad \eta^{-1}(m,\kappa(m),\_) \subseteq \theta^{-1}(\mathbf{nil})}{\langle\langle M_1|x.M_2\rangle;\theta\rangle \overset{\eta}{\underset{\kappa}{\sim}} \left\langle \begin{array}{l}\mathbf{let}\ \mathit{res} = m : (\mathbf{let}\ x = N_1\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\ )\ \mathbf{in} \\ \mathit{res}!\ (m,\kappa(m),i)\end{array} ;\tau \right\rangle}$$

By the induction hypothesis, there exist N_1', τ', and κ' such that $\langle N_1;\tau\rangle \to_{\mathbf{D}}^{*} \langle N_1';\tau'\rangle$, $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N_1';\tau'\rangle$, and $\langle N_1';\tau'\rangle$ is stuck. Since $\langle \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2;\theta\rangle$ is stuck, M_1 cannot be neither of the form $\mathcal{H}[\mathbf{S_0}k.\ M_1']$ for any pure context $\mathcal{H}$ and computation M_1', nor of the form $\mathbf{return}\ V$ for any value V. Then, it follows (by induction on $\langle M_1;\theta\rangle \overset{\eta}{\underset{\kappa'}{\sim}} \langle N_1';\tau'\rangle$) that N_1' cannot be neither of the form $\mathcal{P}[\mathbf{yield}\ W]$ nor of the form $\mathbf{return}\ W$, for any pure context $\mathcal{P}$ and value W. Therefore, the configuration

$$\left\langle \begin{array}{l}\mathbf{let}\ \mathit{res} = m : (\mathbf{let}\ x = N_1'\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\ )\ \mathbf{in} \\ \mathit{res}!\ (m,\kappa(m),i)\end{array} ;\tau' \right\rangle$$

is also stuck, and this completes the current case.

Lemma 13. *If* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *reaches the error state* $\bot$*, i.e.,* $\langle M;\emptyset\rangle \to_{\mathbf{D}}^{+} \bot$*, then* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *also reaches* $\bot$*.*

Proof. This lemma directly follows from Theorem 1.

Finally, we give the proof of Proposition 5.

Proof of Proposition 5. We prove the contrapositive of the proposition: if $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ does not terminate, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also does not terminate. There are three cases where $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ does not terminate: (1) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ diverges; (2) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ gets stuck; (3) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ reaches $\bot$. In each case, Lemmas 11, 12, and 13 imply that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges, gets stuck, or reaches $\bot$, respectively. Thus, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ does not terminate, which completes the proof.

C Supplementary Proofs for Chapter 4

C.1 From $\mathbf{EFF}_{\mathbf{one}}$ to $\mathbf{DEL}_{\mathbf{one}}$

To derive a simulation relation, we extend the macro-translation with η, a partial function which maps the continuation labels in $\mathbf{EFF}_{\mathrm{one}}$ onto those in $\mathbf{DEL}_{\mathrm{one}}$:

$$\underline{l_H}_\eta \coloneqq \left\{\lambda y.\ \mathbf{throw}\ (\eta(l_H))\ y\ \left\{H_\eta^{\mathrm{ops}}\right\}\right\}$$

H_η^{ops} denote the translation of operational clauses with respect to η. Moreover, we derive a macro-translation on contexts by mapping holes to holes; we let $\underline{\mathcal{K}}_\eta$ denote the translation of a context $\mathcal{K}$.

Definition 10. *For any* $\mathbf{EFF}_{\mathrm{one}}$*-configuration* C *and* $\mathbf{DEL}_{\mathrm{one}}$*-configuration* D*,* $C \sim D$ *is defined to hold if and only if* $C \equiv D \equiv \bot$*, or there exist an* $\mathbf{EFF}_{\mathrm{one}}$*-computation* M*, a* $\mathbf{DEL}_{\mathrm{one}}$*-computation* N*, an* $\mathbf{EFF}_{\mathrm{one}}$*-store* θ*, a* $\mathbf{DEL}_{\mathrm{one}}$*-store* τ*, and a partial function* η *such that the following conditions are satisfied:*

1. $C = \langle M; \theta \rangle$.
2. $D = \langle N; \tau \rangle$.
3. $N = \underline{M}_\eta$.
4. η *is injective.*
5. $\eta(\mathrm{Dom}(\theta)) \subseteq \mathrm{Dom}(\tau)$
6. *For any continuation label* l_H *appearing in* M*,* $\theta(l_H)$ *is defined.*
7. *For any continuation label* $l_H \in \mathrm{Dom}(\theta)$*, if* $\theta(l_H) = \mathbf{nil}$*, then* $\tau(\eta(l_H)) = \mathbf{nil}$*. Otherwise, there exists a pure context* $\mathcal{H}$ *in* $\mathbf{EFF}_{\mathrm{one}}$ *such that*

$$\theta(l_H) = \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ y]$$
$$\tau(\eta(l_H)) = \lambda y.\ \left\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_\eta \middle| H^{\mathrm{ret}}_\eta \right\rangle$$

Intuitively, $\langle M; \theta \rangle \sim \langle N; \tau \rangle$ means the correspondence between the two configurations: M is translated into N and the continuations stored in θ is translated into those in τ. We define that θ, τ, and η satisfy the *invariant conditions* (IC) if the conditions 4 through 6 in the definition are fulfilled.

Lemma 14.

1. *For any* $\mathbf{EFF}_{\mathrm{one}}$*-computation* M*,* $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$ *holds.*
2. *For any value* V*,* $\langle \mathbf{return}\ V; \theta \rangle \sim \langle N; \tau \rangle$ *implies that* $N \equiv \mathbf{return}\ \underline{V}_\eta$ *for some* η*.*

Proof. By case analysis on M.

Lemma 15. *Suppose that an* $\mathbf{EFF}_{\mathrm{one}}$*-computation* M *has the free variables* $x_1, \ldots x_n$*, then we have* $\underline{M[V_1/x_1, \ldots, V_n/x_n]}_\eta = \underline{M}_\eta[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n]$ *for any values* $V_1, \ldots V_n$ *and* η*.*

Proof. By straightforward induction on M. For example, if $M \equiv M_1\ M_2$, then

$$\begin{aligned}
\underline{M[V_1/x_1, \ldots, V_n/x_n]}_\eta &\equiv \underline{(M_1\ M_2)[V_1/x_1, \ldots, V_n/x_n]}_\eta \\
&= \underline{M_1[V_1/x_1, \ldots, V_n/x_n]}_\eta\ \underline{M_2[V_1/x_1, \ldots, V_n/x_n]}_\eta \\
&= (\underline{M_1}_\eta[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n])\ (\underline{M_2}_\eta[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n]) \\
&= (\underline{M_1}_\eta\ \underline{M_2}_\eta)[\underline{V_1}_\eta/x_1, \ldots, \underline{V_n}_\eta/x_n]
\end{aligned}$$

Lemma 16. *Suppose that* $C \sim D$ *and* $C \to^{\beta}_{\mathbf{E}} C'$*, then there exists a* $\mathbf{DEL}_{\mathrm{one}}$ *configuration* D' *such that* $D \to^{+}_{\mathbf{D}} D'$ *and* $C' \sim D'$*.*

Proof. We prove by case analysis on $C \to^{\beta}_{\mathbf{E}} C'$. For the sake of brevity, we consider a few non-trivial cases.
Case (op):

Suppose $C \equiv \langle \mathbf{with}\ H\ \mathbf{handle}\ (\mathcal{H}[\mathtt{op}\ V]); \theta\rangle$ and

$$C' \equiv \big\langle M_{\mathtt{op}}[V/p, l_H/k]; \theta[l_H \coloneqq \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ x]]\big\rangle,$$

where $H \equiv \{\mathbf{return}\ x \mapsto M^{\mathrm{ret}},\ \ldots (\mathtt{op}\ p\ k \mapsto M^{\mathtt{op}}) \ldots\}$ and l_H is a fresh label. By the definition of $C \sim D$, we have

$$D \equiv \big\langle \big\langle \underline{\mathcal{H}}_\eta \big[\mathbf{S_0} k.\ \lambda h.\ h!\ \mathbf{inj}_{\mathtt{op}}\ (\underline{V}_\eta, \{\lambda y.\ \mathbf{throw}\ k\ y\ h\})\big] \big| H^{\mathrm{ret}}_\eta \big\rangle\ \big\{H^{\mathrm{ops}}_\eta\big\}; \tau \big\rangle$$

for some η. The configuration D evaluates to

$$D' \coloneqq \Big\langle \underline{M_{\mathtt{op}}}_\eta \big[\underline{V}_\eta/p, \big\{\lambda y.\ \mathbf{throw}\ l\ y\ \big\{H^{\mathrm{ops}}_\eta\big\}\big\}/k\big]; \tau\big[l \coloneqq \lambda y.\ \big\langle \underline{\mathcal{H}}_\eta[\mathbf{return}\ y] \big| H^{\mathrm{ret}}_\eta \big\rangle\big] \Big\rangle$$

where l is a fresh label. Let η' be $\eta[l_H \coloneqq l]$. By Lemma 15, it follows that

$$\underline{M_{\mathtt{op}}}_\eta [\underline{V}_\eta/p, \big\{\lambda y.\ \mathbf{throw}\ l\ y\ \big\{H^{\mathrm{ops}}_\eta\big\}\big\}/k] \equiv \underline{M_{\mathtt{op}}[V/p, l_H/k]}_{\eta'}.$$

Note that $\underline{M_{\mathtt{op}}}_\eta \equiv \underline{M_{\mathtt{op}}}_{\eta'}$, $\underline{\mathcal{H}}_\eta \equiv \underline{\mathcal{H}}_{\eta'}$, $H^{\mathrm{ret}}_\eta \equiv H^{\mathrm{ret}}_{\eta'}$, and $H^{\mathrm{ops}}_\eta \equiv H^{\mathrm{ops}}_{\eta'}$ hold since l_H does not appear in any of $M_{\mathtt{op}}$, $\mathcal{H}$, H^{ret}, and H^{ops}_η. Finally, we conclude that

$$\big\langle M_{\mathtt{op}}[V/p, l_H/k]; \theta[l_H \coloneqq \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ x]]\big\rangle \sim D'.$$

It is straightforward to check that the IC are satisfied with respect to the updated stores and η'.

Case (throw):

Suppose $C \equiv \langle \mathbf{throw}\ l_H\ V; \theta\rangle$ and $C' \equiv \langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ V]; \theta[l_H \coloneqq \mathbf{nil}]\rangle$, where $\theta(l_H) = \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ x]$. By the definition of $C \sim D$, we obtain

$$D \equiv \big\langle \big\{\lambda y.\ \mathbf{throw}\ (\eta(l_H))\ y\ \big\{H^{\mathrm{ops}}_\eta\big\}\big\}!\ \underline{V}_\eta; \tau \big\rangle,$$

for some η and τ such that

$$\tau(\eta(l_H)) = \lambda y.\ \Big\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_\eta \Big| H^{\mathrm{ret}}_\eta \Big\rangle.$$

Thus, D evaluates to

$$D' \coloneqq \Big\langle \Big(\Big\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_\eta \Big| H^{\mathrm{ret}}_\eta \Big\rangle [\underline{V}_\eta/y] \Big)\ \big\{H^{\mathrm{ops}}_\eta\big\}; \tau[\eta(l_H) \coloneqq \mathbf{nil}] \Big\rangle.$$

Next, from Lemma 15, we obtain

$$\Big\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_\eta \Big| H^{\mathrm{ret}}_\eta \Big\rangle [\underline{V}_\eta/y] \equiv \Big\langle \underline{\mathcal{H}[\mathbf{return}\ V]}_\eta \Big| H^{\mathrm{ret}}_\eta \Big\rangle.$$

Therefore, we conclude that

$$\langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ V]; \theta[l_H \coloneqq \mathbf{nil}]\rangle \sim D'$$

since

$$\underline{\mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\ V]}_\eta \equiv \Big\langle \underline{\mathcal{H}[\mathbf{return}\ V]}_\eta \Big| H^{\mathrm{ret}}_\eta \Big\rangle \big\{H^{\mathrm{ops}}_\eta\big\}.$$

Note that the updated stores and η satisfy the IC.

Case (fail):
Suppose $C \equiv \langle \mathbf{throw}\ l_H\ ; \theta \rangle$, $\theta(l_H) = \mathbf{nil}$, and $\langle \mathbf{throw}\ l_H\ V; \theta \rangle \to_\beta^{\mathbf{E}} \bot$. By the definition of $C \sim D$, we have

$$D \equiv \left\langle \left\{ \lambda y.\ \mathbf{throw}\ (\eta(l_H))\ y\ \left\{ H_\eta^{\mathrm{ops}} \right\} \right\} !\ \underline{V}_\eta; \tau \right\rangle,$$

where τ is a store such that $\tau(\eta(l_H)) = \mathbf{nil}$. Hence, the invocation of $\eta(l_H)$ fails and D evaluates to $\bot$, which concludes the current case.

Lemma 17. $\underline{\mathcal{K}[M]}_\eta \equiv \underline{\mathcal{K}}_\eta[\underline{M}_\eta]$ *for an arbitrary context* K *and computation* M.

Proof. By straightforward induction on K.

Lemma 18. *If* $\langle M; \theta \rangle \sim \langle N; \tau \rangle$ *and* M *can be decomposed into an evaluation context* $\mathcal{C}$ *and a redex* M'*, then there exist some* η *such that* $N \equiv \underline{M}_\eta$ *can be also decomposed into an evaluation context* $\underline{\mathcal{C}}_\eta$ *and a redex* $\underline{M'}_\eta$.

Proof. It is easy to check this by induction on the number of frames of $\mathcal{C}$. Note that the extended macro-translation maps frames to frames and redexes to redexes.

Proposition 6 (Simulation). *Suppose that* $C \sim D$ *and* $C \to_{\mathbf{E}} C'$*, then there exists a* $\mathbf{DEL}_{\mathrm{one}}$ *configuration* D' *such that* $D \to_{\mathbf{D}}^{+} D'$ *and* $C' \sim D'$.

Proof. Since C is not in normal form, there exist M and θ such that $C \equiv \langle M; \theta \rangle$, and according to the semantics of $\mathbf{EFF}_{\mathrm{one}}$, M can be decomposed into an evaluation context $\mathcal{C}$ and a redex M'. By the definition of $C \sim D$, there exist η and τ such that $D \equiv \langle \underline{M}_\eta; \tau \rangle$, and θ, τ, and η satisfy the IC. By Lemma 18, $\underline{M}_\eta$ can also be decomposed into $\underline{\mathcal{C}}_\eta$ and a redex $\underline{M'}_\eta$, namely $\underline{M}_\eta = \underline{\mathcal{C}}_\eta[\underline{M'}_\eta]$.

If $\langle M; \theta \rangle \to_{\mathbf{E}}^{\beta} \bot$, then we have $(C \equiv \langle \mathcal{C}[M]; \theta \rangle) \to_{\mathbf{E}} \bot$. By Lemma 16, $\langle M'; \tau \rangle$ also evaluates to $\bot$. Hence, we conclude that $\langle \mathcal{D}[M']; \tau \rangle \to_{\mathbf{D}}^{+} \bot$.

Next, suppose $\langle M'; \theta \rangle \to_{\mathbf{E}}^{\beta} \langle M''; \theta' \rangle$. By Lemma 16, we obtain $\langle \underline{M'}_\eta; \tau \rangle \to_{\mathbf{D}}^{+} \langle \underline{M''}_{\eta'}; \tau' \rangle$ and $\langle M''; \theta' \rangle \sim \langle \underline{M''}_{\eta'}; \tau' \rangle$, where θ', τ', and η' satisfy the IC. Moreover, $\underline{\mathcal{C}}_\eta \equiv \underline{\mathcal{C}}_{\eta'}$ holds since the labels in $\mathrm{Dom}(\eta') \setminus \mathrm{Dom}(\eta)$ are fresh and do not appear in $\mathcal{C}$. Therefore, we conclude that $\langle \mathcal{C}[M'']; \theta' \rangle \sim \left\langle \underline{\mathcal{C}[M'']}_{\eta'}; \tau' \right\rangle$ by Lemma 17.

To establish the strong macro-expressibility of $\mathbf{EFF}_{\mathrm{one}}$ in $\mathbf{DEL}_{\mathrm{one}}$, we prove four lemmas.

Lemma 19. *We say an* $\mathbf{EFF}_{\mathrm{one}}$*-configuration* $\langle M; \theta \rangle$ *is well-formed if for any continuation label* l_H *that appears in* M*,* $\theta(l)$ *is defined, and this property is preserved under reduction.*

Proof. By straightforward case analysis on M.

Lemma 20. *Suppose that* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ *diverges, i.e., the reduction sequence of* M *is infinite. Then* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ *also diverges.*

Proof. By applying Theorem 6 repeatedly, we also obtain an infinite reduction sequence of $\underline{M}$. Since the reduction of $\mathbf{DEL}_{\mathrm{one}}$ is deterministic, this implies that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ also diverges.

Lemma 21. *Suppose that* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ *gets stuck: there exists a term* M' *and a store* θ *such that* $\langle M;\emptyset\rangle \to_{\mathbf{E}}^{*} \langle M';\theta\rangle$, $M' \neq \mathbf{return}\ V$ *for any value* V*, and there is no rule that can reduce* $\langle M';\theta\rangle$*. Then,* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ *also gets stuck.*

Proof. It suffices to show the following statement:

> Suppose that there exist M, N, θ, τ, and η such that $N = \underline{M}_{\eta}$, $\langle M;\theta\rangle$ is well-formed, and θ, τ, and η satisfy the IC. If $\langle M;\theta\rangle$ is stuck, then $\langle N;\tau\rangle$ is also be stuck.

We prove by induction on the structure of M. However, we treat only four cases. The other cases follow by similar reasoning.

Case 1:

$$M \equiv \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M'$$

Since the $\langle M;\theta\rangle$ is stuck, we know that $V \neq (V_1, V_2)$ for any values V_1 and V_2. Consequently, $\underline{V}_{\eta}$ also cannot be of the form (W_1, W_2) for any values W_1 and W_2. Therefore, the configuration $\left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M}_{\eta}; \tau \right\rangle$ is stuck, and this concludes the current case.

Case 2:

$$M \equiv \mathbf{throw}\ V\ W$$

It follows that $V \neq l_H$ for any continuation label l_H, since if $V = l_H$ for some l, then by the well-formedness of $\langle \mathbf{throw}\ V\ W;\theta\rangle$, we know that $l_H \in \mathrm{Dom}(\theta)$. However, this contradicts that $\langle \mathbf{throw}\ V\ W;\theta\rangle$ is stuck. This implies that $\underline{\mathbf{throw}\ V\ W}_{\eta}$ is also stuck, which concludes the current case.

Case 3:

$$M \equiv \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$$

Since $\langle M;\theta\rangle$ is stuck, M_1 cannot be of the form $\mathbf{return}\ V$ for any value V. By the induction hypothesis, we obtain that N_1 is stuck and that $N_1 \neq \mathbf{return}\ W$ for any value W. Consequently, the configuration $\left\langle \mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_{\eta}; \tau \right\rangle$ is also stuck, which completes the current case.

Case 4:

$$M \equiv \mathbf{with}\ H\ \mathbf{handle}\ M'$$

By the definition of the translation, we obtain

$$N \equiv \left\langle \underline{M'}_{\eta} \middle| H^{\mathrm{ret}} \right\rangle \{H^{\mathrm{ops}}\}.$$

Since $\langle M;\theta\rangle$ is stuck, M' cannot be neither of the form $\mathcal{H}[\mathtt{op}\ V]$ nor of the form $\mathbf{return}\ V$ for any pure context $\mathcal{H}$, operation symbol $\mathtt{op}$, and value V. Then, it follows by induction on M' that $\underline{M'}_\eta$ cannot be neither of the form $\mathcal{P}[\mathbf{S_0}k.\ N']$ nor of the form $\mathbf{return}\ W$, for any pure context $\mathcal{P}$, computation N', and value W. Therefore, the configuration $\langle N;\tau\rangle$ is also stuck, and this completes the current case.

Lemma 22. *If* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ *reaches the error state* $\bot$*, i.e.,* $\langle M;\emptyset\rangle \to_{\mathbf{E}}^{+} \bot$*, then* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ *also reaches* $\bot$*.*

Proof. This lemma directly follows from Theorem 6.

Corollary 2 (Preservation of semantics). $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ *terminates if and only if* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ *terminates.*

Proof. We first show the "only if" direction. Suppose that $\langle M;\emptyset\rangle \to_{\mathbf{E}}^{+} \langle \mathbf{return}\ V;\theta\rangle$ for some θ. Lemma 14 yields that $\langle \underline{M};\emptyset\rangle$. By applying Proposition 6 iteratively, there exists η and τ such that $\langle \underline{M};\emptyset\rangle \to_{\mathbf{D}}^{+} \langle \mathbf{return}\ \underline{V}_\eta;\tau\rangle$. This implies that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M}) = V$. We then show the "if" direction. We prove the contrapositive of the proposition: if $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ does not terminate, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ also does not terminate. There are three cases where $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ does not terminate: (1) $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ diverges; (2) $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ gets stuck; (3) $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ reaches $\bot$. In each case, Lemmas 20, 21, and 22 imply that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ also diverges, gets stuck, or reaches $\bot$, respectively. Thus, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ does not terminate, which completes the proof.

C.2 From $\mathbf{DEL}_{\mathbf{one}}$ to $\mathbf{EFF}_{\mathbf{one}}$

As in Appendix C.1, we extend the macro-translation with η, a partial function from labels in $\mathbf{DEL}_{\mathrm{one}}$ to those in $\mathbf{EFF}_{\mathrm{one}}$, by

$$\underline{l}_\eta := \eta(l),$$

and derive the macro-translation on contexts $\mathcal{K} \mapsto \underline{\mathcal{K}}_\eta$ by mapping holes to holes.

Definition 11. *For any configurations of* $\mathbf{DEL}_{\mathrm{one}}$ *and* $\mathbf{EFF}_{\mathrm{one}}$*, denoted by* C *and* D*,* $C \sim D$ *is defined to hold if and only if* $C \equiv D \equiv \bot$*, or there exist a* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M*, an* $\mathbf{EFF}_{\mathrm{one}}$*-computation* N*, a* $\mathbf{DEL}_{\mathrm{one}}$*-store* θ*, an* $\mathbf{EFF}_{\mathrm{one}}$*-store* τ*, and a partial function* η *such that the following conditions are satisfied:*

1. $C = \langle M;\theta\rangle$.
2. $D = \langle N;\tau\rangle$.
3. $N = \underline{M}_\eta$.
4. $\eta(\mathrm{Dom}(\theta)) \subseteq \mathrm{Dom}(\tau)$.
5. *For any continuation label* l *appearing in* M*,* $\theta(l)$ *is defined.*

6. *For any continuation label* $l \in \mathrm{Dom}(\theta)$, *if* $\theta(l) = \mathbf{nil}$, *then* $\tau(\eta(l)) = \mathbf{nil}$. *Otherwise, there exists a pure context* $\mathcal{H}$ *of* $\mathbf{DEL}_{\mathrm{one}}$ *such that*

$$\theta(l) = \lambda y.\, \langle \mathcal{H}[\mathbf{return}\ y] | x.N \rangle,$$
$$\tau(\eta(l)) = \lambda y.\, \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{N},\ \mathtt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{\mathcal{H}[\mathbf{return}\ y]}_{\eta}.$$

We define that θ, τ, and η satisfy the *invariant conditions* (IC) if the conditions 4 through 6 in the definition are satisfied.

Lemma 23.

1. *For any* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M, $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$ *holds.*
2. *For any* $\mathbf{DEL}_{\mathrm{one}}$*-value* V, $\langle \mathbf{return}\ V; \theta \rangle \sim \langle N; \tau \rangle$ *implies that* $N \equiv \mathbf{return}\ \underline{V}_{\eta}$ *for some* η.

Proof. By case analysis on M.

Lemma 24. *Suppose that a* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M *has the free variables* $x_1, \ldots x_n$, *then we have* $\underline{M[V_1/x_1, \ldots, V_n/x_n]}_{\eta} = \underline{M}_{\eta}[\underline{V_1}_{\eta}/x_1, \ldots, \underline{V_n}_{\eta}/x_n]$ *for any values* $V_1, \ldots V_n$ *and* η.

Proof. By straightforward induction on M.

Lemma 25. *Suppose that* $C \sim D$ *and* $C \to_{\mathbf{D}}^{\beta} C'$, *then there exists an* $\mathbf{EFF}_{\mathrm{one}}$ *configuration* D' *such that* $D \to_{\mathbf{E}}^{+} D'$ *and* $C' \sim D'$.

Proof. We prove by case analysis on $C \to_{\mathbf{D}}^{\beta} C'$. For the sake of brevity, we consider a few non-trivial cases.

Case (shift):

Suppose that $C \equiv \langle \langle \mathcal{H}[\mathbf{S_0} k.\ M] | x.N \rangle; \theta \rangle$, l is a fresh label in this configuration, and $C' \equiv \langle M[l/k]; \theta' \rangle$, where $\theta' \coloneqq \theta[l \coloneqq \lambda y.\, \langle \mathcal{H}[\mathbf{return}\ y] | x.N \rangle]$. By the definition of $C \sim D$, we have for some η,

$$D \equiv \big\langle \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}}_{\eta}\big[\mathtt{shift0}\ \big\{\lambda k.\ \underline{M}_{\eta}\big\}\big]; \tau \big\rangle,$$

where $H \equiv \{\mathbf{return}\ x \mapsto \underline{N}_{\eta},\ \mathtt{shift0}\ p\ k \mapsto p!\ k\}$ and θ, τ, and η satisfy the IC.

D evaluates to the following configuration:

$$D' \coloneqq \big\langle \underline{M}_{\eta}[m_H/k]; \tau' \big\rangle,$$

where $\tau' \coloneqq \tau[m_H \coloneqq \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}}_{\eta}[\mathbf{return}\ y]]$ and m_H is a fresh label. Let η' be $\eta[l \coloneqq m_H]$. By Lemma24, we obtain

$$\underline{M}_{\eta}[m_H/k] \equiv \underline{M[l/k]}_{\eta'}.$$

Moreover, since m_H is took as a fresh label in D, we know that

$$\begin{aligned}&\lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}}_\eta[\mathbf{return}\ y]\\ &\equiv \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}}_{\eta'}[\mathbf{return}\ y]\\ &\equiv \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}[\mathbf{return}\ y]}_{\eta'},\end{aligned}$$

and this implies that θ', τ', and η' satisfy the IC.

Therefore, we conclude that

$$\langle M[l/k];\theta'\rangle \sim \left\langle \underline{M}_\eta[m_H/k];\tau'\right\rangle,$$

which completes the current case.

Case (throw):

Suppose that $C \equiv \langle \mathbf{throw}\ l\ V;\theta\rangle$, $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\ y]|x.N\rangle$, and $C' \equiv \langle\langle \mathcal{H}[\mathbf{return}\ V]|x.N\rangle;\theta[l \coloneqq \mathbf{nil}]\rangle$. By the definition of $C \sim D$, we obtain

$$D \equiv \left\langle \mathbf{throw}\ \eta(l)\ \underline{V}_\eta;\tau\right\rangle,$$

for some τ and η such that

$$\tau(\eta(l)) = \lambda y.\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{N},\ \texttt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{\mathcal{H}[\mathbf{return}\ y]}_\eta,$$

where θ, τ, and η satisfy the IC.

D evaluates to

$$\begin{aligned}D' &\coloneqq \left\langle \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{N},\ \texttt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{\mathcal{H}}_\eta\left[\mathbf{return}\ \underline{V}_\eta\right];\tau[\eta(l) \coloneqq \mathbf{nil}]\right\rangle\\ &\equiv \left\langle \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{N},\ \texttt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{\mathcal{H}[\mathbf{return}\ V]}_\eta;\tau[\eta(l) \coloneqq \mathbf{nil}]\right\rangle.\end{aligned}$$

It is trivial that $\theta[l \coloneqq \mathbf{nil}]$, $\tau[\eta(l) \coloneqq \mathbf{nil}]$, and η satisfy the IC. Therefore, we obtain $C' \sim D'$, and this completes the current case.

Lemma 26. $\underline{\mathcal{K}[M]}_\eta \equiv \underline{\mathcal{K}}_\eta[\underline{M}_\eta]$ *for an arbitrary context* K *and computation* M.

Proof. By straightforward induction on K.

Lemma 27. *If* $\langle M;\theta\rangle \sim \langle N;\tau\rangle$ *and* M *can be decomposed into an evaluation context* $\mathcal{C}$ *and a redex* M'*, then there exist some* η *such that* $N \equiv \underline{M}_\eta$ *can be also decomposed into an evaluation context* $\underline{\mathcal{C}}_\eta$ *and a redex* $\underline{M'}_\eta$.

Proof. It is easy to check this by induction on the number of frames of $\mathcal{C}$. Note that the extended macro-translation maps frames to frames and redexes to redexes.

Proposition 7 (Simulation). *Suppose that* $C \sim D$ *and* $C \to_{\mathbf{D}} C'$*, then there exists an* $\mathbf{EFF}_{\text{one}}$ *configuration* D' *such that* $D \to_{\mathbf{E}}^{+} D'$ *and* $C' \sim D'$.

Proof. Since C is not in normal form, there exist M and θ such that $C \equiv \langle M;\theta\rangle$ and according to the semantics of $\mathbf{DEL}_{\mathrm{one}}$, M can be decomposed into an evaluation context $\mathcal{C}$ and a redex M'. By the definition of $C \sim D$, there exist η and τ such that $D \equiv \langle \underline{M}_\eta;\tau\rangle$, and θ, τ, and η satisfy the IC. By Lemma 27, $\underline{M}_\eta$ can also be decomposed into $\underline{\mathcal{C}}_\eta$ and a redex $\underline{M'}_\eta$, namely $\underline{M}_\eta = \underline{\mathcal{C}}_\eta[\underline{M'}_\eta]$.

If $\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \bot$, then we have $(C \equiv \langle \mathcal{C}[M];\theta\rangle) \to_{\mathbf{D}} \bot$. By Lemma 25, $\langle M';\tau\rangle$ also evaluates to $\bot$. Hence, we conclude that $\langle \mathcal{D}[M'];\tau\rangle \to^{+}_{\mathbf{E}} \bot$.

Next, suppose $\langle M';\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M'';\theta'\rangle$. By Lemma 25, we obtain $\langle \underline{M'}_\eta;\tau\rangle \to^{+}_{\mathbf{E}} \langle \underline{M''}_{\eta'};\tau'\rangle$ and $\langle M'';\theta'\rangle \sim \langle \underline{M''}_{\eta'};\tau'\rangle$, where θ', τ', and η' satisfy the IC. Moreover, $\underline{\mathcal{C}}_\eta \equiv \underline{\mathcal{C}}_{\eta'}$ holds since the labels in $\mathrm{Dom}(\eta') \setminus \mathrm{Dom}(\eta)$ are fresh and do not appear in $\mathcal{C}$. Therefore, we conclude that $\langle \mathcal{C}[M''];\theta'\rangle \sim \left\langle \underline{\mathcal{C}[M'']}_{\eta'};\tau'\right\rangle$ by Lemma 26.

To establish the strong macro-expressibility of $\mathbf{EFF}_{\mathrm{one}}$ in $\mathbf{DEL}_{\mathrm{one}}$, we prove four lemmas.

Lemma 28. *We say a* $\mathbf{DEL}_{\mathrm{one}}$*-configuration* $\langle M;\theta\rangle$ *is well-formed if for any continuation label* l *that appears in* M*,* $\theta(l)$ *is defined, and this property is preserved under reduction.*

Proof. By straightforward case analysis on M.

Lemma 29. *Suppose that* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *diverges, i.e., the reduction sequence of* M *is infinite. Then* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ *also diverges.*

Proof. By applying Theorem 7 repeatedly, we also obtain an infinite reduction sequence of $\underline{M}$. Since the reduction of $\mathbf{EFF}_{\mathrm{one}}$ is deterministic, this implies that $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ also diverges.

Lemma 30. *Suppose that* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *gets stuck: there exists a term* M' *and a store* θ *such that* $\langle M;\emptyset\rangle \to^{*}_{\mathbf{D}} \langle M';\theta\rangle$*,* $M' \neq \mathbf{return}\ V$ *for any value* V*, and there is no rule that can reduce* $\langle M';\theta\rangle$*. Then,* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ *also gets stuck.*

Proof. It suffices to show the following statement:

> Suppose that there exist M, N, θ, τ, and η such that $N = \underline{M}_\eta$, $\langle M;\theta\rangle$ is well-formed, and θ, τ, and η satisfy the IC. If $\langle M;\theta\rangle$ is stuck, then $\langle N;\tau\rangle$ is also be stuck.

We prove by induction on the structure of M. However, we treat only four cases. The other cases follow by similar reasoning.

Case 1:

$$M \equiv \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M'$$

Since the $\langle M;\theta\rangle$ is stuck, we know that $V \neq (V_1, V_2)$ for any values V_1 and V_2. Consequently, $\underline{V}_\eta$ also cannot be of the form (W_1, W_2) for any values W_1 and W_2. Therefore, the configuration $\left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M}_\eta;\tau\right\rangle$ is stuck, and this concludes the current case.

Case 2:

$$M \equiv \mathbf{throw}\ V\ W$$

It follows that $V \neq l$ for any continuation label l, since if $V = l$ for some l, then by the well-formedness of $\langle \mathbf{throw}\ V\ W; \theta \rangle$, we know that $l \in \mathrm{Dom}(\theta)$. However, this contradicts that $\langle \mathbf{throw}\ V\ W; \theta \rangle$ is stuck. This implies that $\underline{\mathbf{throw}\ V\ W}_\eta$ is also stuck, which concludes the current case.

Case 3:

$$M \equiv \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$$

Since $\langle M; \theta \rangle$ is stuck, M_1 cannot be of the form $\mathbf{return}\ V$ for any value V. By the induction hypothesis, we obtain that N_1 is stuck and that $N_1 \neq \mathbf{return}\ W$ for any value W. Consequently, the configuration $\left\langle \mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_\eta ; \tau \right\rangle$ is also stuck, which completes the current case.

Case 4:

$$M \equiv \langle M_1 | x.M_2 \rangle$$

By the definition of the translation, we obtain

$$N \equiv \mathbf{with}\ \{\mathbf{return}\ x \mapsto \underline{M_2}_\eta,\ \mathtt{shift0}\ p\ k \mapsto p!\ k\}\ \mathbf{handle}\ \underline{M_1}_\eta.$$

Since $\langle M; \theta \rangle$ is stuck, M_1 cannot be neither of the form $\mathcal{H}[\mathbf{S_0}k.\ L]$ nor of the form $\mathbf{return}\ V$ for any pure context $\mathcal{H}$, computation L, and value V. Then, it follows by induction on M_1 that $\underline{M_1}_\eta$ cannot be neither of the form $\mathcal{P}[\mathtt{op}\ W]$ nor of the form $\mathbf{return}\ W$, for any pure context $\mathcal{P}$, and value W. Therefore, the configuration $\langle N; \tau \rangle$ is also stuck, and this completes the current case.

Lemma 31. *If* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *reaches the error state* $\bot$*, i.e.,* $\langle M; \emptyset \rangle \to_{\mathbf{D}}^{+} \bot$*, then* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ *also reaches* $\bot$*.*

Proof. This lemma directly follows from Theorem 6.

Corollary 3 (Preservation of semantics). $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *terminates if and only if* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ *terminates.*

Proof. We first show the "only if" direction. Suppose that $\langle M; \emptyset \rangle \to_{\mathbf{D}}^{+} \langle \mathbf{return}\ V; \theta \rangle$ for some θ. Lemma 23 yields that $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$. By applying Proposition 7 iteratively, there exists η and τ such that $\langle \underline{M}; \emptyset \rangle \to_{\mathbf{E}}^{+} \langle \mathbf{return}\ \underline{V}_\eta; \tau \rangle$. This implies that $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M}) = V$. We then show the "if" direction. We prove the contrapositive of the proposition: if $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ does not terminate, $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ also does not terminate. There are three cases where $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ does not terminate: (1) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ diverges; (2) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ gets stuck; (3) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ reaches $\bot$. In each case, Lemmas 29, 30, and 31 imply that $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ also diverges, gets stuck, or reaches $\bot$, respectively. Thus, $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ does not terminate, which completes the proof.

D Supplementary Proofs for Chapter 5

In this appendix, we provide the complete proof of Theorems 6 and 7.

D.1 Proof of Theorem 6

Proof.

$$M \coloneqq \begin{pmatrix} \textbf{let } r = \textbf{create } \textbf{inj}_{\mathrm{A}}\,() \textbf{ in} \\ \textbf{let } i = \textbf{get } r \textbf{ in} \\ \textbf{let } \_ = \textbf{set } r\ \textbf{inj}_{\mathrm{B}}\,() \textbf{ in} \\ \textbf{let } k = \textbf{get } r \textbf{ in} \\ \textbf{return } (i,k) \end{pmatrix} \mapsto \begin{pmatrix} \textbf{let } r = \underline{\textbf{create}}\ (\textbf{inj}_{\mathrm{A}}\,()) \textbf{ in} \\ \textbf{let } i = \underline{\textbf{get}}\ r \textbf{ in} \\ \textbf{let } \_ = \underline{\textbf{set}}\ r\ (\textbf{inj}_{\mathrm{B}}\,()) \textbf{ in} \\ \textbf{let } k = \underline{\textbf{get}}\ r \textbf{ in} \\ \textbf{return } (i,k) \end{pmatrix}$$

Consider the above **REF**-Program M. In **REF**, M evaluates to $(\textbf{inj}_{\mathrm{A}}\,(), \textbf{inj}_{\mathrm{B}}\,())$. Suppose that we have a macro-translation $\underline{\cdot}$ from **REF** to $\mathbf{EFF}_{\mathrm{one}}$. Then the evaluation of the translated program $\underline{M}$ terminates with a value $(\textbf{inj}_{\mathrm{A}}\,(), \textbf{inj}_{\mathrm{B}}\,())$. This implies that there exist a value X and store θ such that

$$\langle \mathcal{C}[\underline{\textbf{create}}\ (\textbf{inj}_{\mathrm{A}}\,())]; \emptyset \rangle \rightarrow_{\mathbf{E}}^{+} \langle \mathcal{C}[\textbf{return } X]; \theta \rangle$$

where

$$\mathcal{C} = \big(\textbf{let } r = [\,] \textbf{ in let } i = \underline{\textbf{get}}\ r \textbf{ in let } \_ = \underline{\textbf{set}}\ r\ (\textbf{inj}_{\mathrm{B}}\,()) \textbf{ in let } k = \underline{\textbf{get}}\ r \textbf{ in return } (i,k)\big).$$

We claim that the two evaluations of $\underline{\textbf{get}}\ r$ (with r being substituted for X) yield the same value $\textbf{inj}_{\mathrm{A}}\,()$. The first $\underline{\textbf{get}}\ X$ is evaluated under the configuration $\langle \mathcal{D}[\underline{\textbf{get}}\ X]; \theta \rangle$ where

$$\mathcal{D} = \big(\textbf{let } i = [\,] \textbf{ in let } \_ = \underline{\textbf{set}}\ X\ (\textbf{inj}_{\mathrm{B}}\,()) \textbf{ in let } k = \underline{\textbf{get}}\ X \textbf{ in return } (i,k)\big).$$

Since the reduction of $\underline{M}$ terminates, there exists a store θ' such that

$$\langle \mathcal{D}[\underline{\textbf{get}}\ X]; \theta \rangle \rightarrow_{\mathbf{E}}^{+} \langle \mathcal{D}[\textbf{return } (\textbf{inj}_{\mathrm{A}}\,())]; \theta' \rangle$$

because $\underline{M}$ evaluates to $(\textbf{inj}_{\mathrm{A}}\,(), \textbf{inj}_{\mathrm{B}}\,())$. We shall show that a non-local computation which goes beyond $\underline{\textbf{get}}\ X$ does not happen in this evaluation. But this is trivial; since the evaluation context $\mathcal{D}$ does not have a handler that encloses the hole, there is no way to capture a raised operation in $\mathcal{D}$. All operations invoked in the evaluation should be caught within $\underline{\textbf{get}}\ X$; otherwise, the whole computation would not terminate normally.

This implies that the evaluation of $\mathcal{E}[\underline{\textbf{get}}\ X]$ also yields $\textbf{inj}_{\mathrm{A}}\,()$[14] where

$$\mathcal{E} = (\textbf{let } k = [\,] \textbf{ in return } (\textbf{inj}_{\mathrm{A}}\,(), k)).$$

Therefore, $\underline{M}$ evaluates to $(\textbf{inj}_{\mathrm{A}}\,(), \textbf{inj}_{\mathrm{A}}\,())$.

[14] To be precise, the second reduction of $\underline{\textbf{get}}\ X$ is identical to the first one modulo the labels of continuations.

Consider the following program L:

$$L \coloneqq \left(\begin{array}{l} \textbf{let } r = M \textbf{ in} \\ \textbf{case } r \textbf{ of } \{ \\ \quad (\textbf{inj}_{\mathrm{A}}\,(), \textbf{inj}_{\mathrm{B}}\,()) \mapsto \textbf{return } () \\ \quad \_ \mapsto (\lambda x.\ x!\ x)\ \{\lambda x.\ x!\ x\} \\ \} \end{array}\right)$$

In **REF**, L evaluates to $()$. However, in $\mathbf{EFF}_{\mathrm{one}}$, the evaluation of $\underline{L}$ does not terminate, which is a contradiction. Therefore, there is no (weak) macro-translation from **REF** to $\mathbf{EFF}_{\mathrm{one}}$.

D.2 Proof of Theorem 7

We extend the translation $M \mapsto \underline{M}$ by introducing a partial function η, which maps reference cells onto coroutine labels, as follows:

$$\underline{l}_{\eta} \coloneqq \eta(l).$$

Definition 12. *A binary relation $\langle M; \theta\rangle \sim \langle N; \tau\rangle$ is defined to hold if and only if there exists η such that the following conditions are satisfied:*

1. $N = \underline{M}_{\eta}$
2. *η is injective*
3. $\eta(\mathrm{Dom}(\theta)) \subseteq \mathrm{Dom}(\tau)$
4. *For any $l \in \mathrm{Dom}(\theta)$, if $\theta(l) = V$, then*

$$\tau(\eta(l)) = \big\{\lambda x.\ \textbf{let } y = \textbf{return } x \textbf{ in } \mathit{loop}!\ \underline{V}_{\eta}\ y\big\}.$$

We define θ, τ, and η satisfy the invariant conditions (IC) if the conditions 2 through 4 in the definition are fulfilled.

We prove the following important property of *loop*:

Lemma 32. *For any* **AC** *store θ and value V,*

$$\langle \mathit{loop}!\ V; \theta\rangle \to^{+}_{\mathbf{AC}} \left\langle \left\{ \begin{array}{l} \lambda f.\lambda s.\lambda a.\ \textbf{case } a \textbf{ of}\{ \\ \quad \textbf{inj}_{\mathrm{Get}}\,() \mapsto \textbf{let } y = \textbf{yield } s \textbf{ in } f!\ s\ y \\ \quad \textbf{inj}_{\mathrm{Set}}\,v \mapsto \textbf{let } y = \textbf{yield } () \textbf{ in } f!\ v\ y\} \end{array} \right\}!\ \mathit{loop}\ V; \theta \right\rangle$$

Proof. By straightforward calculation.

Next, we establish a substitution property for the translation.

Lemma 33. *Suppose that a* **REF***-computation M has free variables $x_1, \ldots x_n$. Then for any values $V_1, \ldots V_n$ and store η, the following equality holds:*

$$\underline{M[V_1/x_1, \ldots, V_n/x_n]}_{\eta} \equiv \underline{M}_{\eta}[\underline{V_1}_{\eta}/x_1, \ldots, \underline{V_n}_{\eta}/x_n].$$

Proof. By induction on M.

Lemma 34. *Suppose that $C \sim D$ and $C \to_{\mathbf{E}}^{\beta} C'$, then there exists a* $\mathbf{DEL}_{\mathrm{one}}$ *configuration D' such that $D \to_{\mathbf{D}}^{+} D'$ and $C' \sim D'$.*

Proof. We prove by case analysis on $C \to_{\mathbf{E}}^{\beta} C'$. For brevity, we focus on the reduction rules specific to **REF**.

Case (create):

Suppose that $C \equiv \langle \mathbf{create}\ V; \theta \rangle$ and $C' \equiv \langle \mathbf{return}\ l; \theta[l \coloneqq V] \rangle$ for a fresh label l. By the definition of $C \sim D$, there exists η such that

$$D = \big\langle \mathbf{create}\ \big\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_{\eta}\ y\big\}; \tau \big\rangle.$$

The configuration D evaluates to $D' \equiv \big\langle \mathbf{return}\ l'; \tau\big[l' \coloneqq \big\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_{\eta}\ y\big\}\big] \big\rangle$. Let η' be $\eta[l \coloneqq l']$. By Lemma 33, we have $\mathbf{return}\ l' \equiv \underline{\mathbf{return}\ l}_{\eta'}$. Therefore, we conclude that

$$\langle \mathbf{return}\ l; \theta[l \coloneqq V] \rangle \sim \big\langle \underline{\mathbf{return}\ l}_{\eta'}; \tau\big[l' \coloneqq \big\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_{\eta'}\ y\big\}\big] \big\rangle.$$

Note that V does not contain l since l is a fresh label and the updated stores and η' satisfy the IC.

Case (set):

Suppose that $C \equiv \langle \mathbf{set}\ l\ V; \theta \rangle$ and $C' \equiv \langle \mathbf{return}\ (); \theta[l \coloneqq V] \rangle$ for some $l \in \mathrm{Dom}(\theta)$. By the definition of $C \sim D$, there exists η such that

$$D \equiv \langle \mathbf{resume}\ (\eta(l))\ (\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{eta}); \tau \rangle$$

and

$$\tau(\eta(l)) = \Big\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_{\eta}\ y\Big\}.$$

The reduction of D proceeds as follows:

$$\begin{aligned}
D &\equiv \big\langle \mathbf{resume}\ (\eta(l))\ \big(\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{\eta}\big); \tau \big\rangle \\
&\to_{\mathbf{AC}} \left\langle \begin{array}{l} (\eta(l)) : \Big(\Big\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_{\eta}\ y\Big\}!\ \big(\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{\eta}\big)\Big); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to_{\mathbf{AC}}^{+} \Big\langle (\eta(l)) : \Big(\mathit{loop}!\ \underline{\theta(l)}_{\eta}\ \big(\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{\eta}\big)\Big); \tau[(\eta(l)) \coloneqq \mathbf{nil}] \Big\rangle \\
&\to_{\mathbf{AC}}^{+} \left\langle \begin{array}{l} (\eta(l)) : \left(\begin{array}{l} \left\{\begin{array}{l} \lambda f.\lambda s.\lambda a.\ \mathbf{case}\ a\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ () \mapsto \mathbf{let}\ y = \mathbf{yield}\ s\ \mathbf{in}\ f!\ s\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ f!\ v\ y\} \end{array}\right\}! \\ \mathit{loop}\ \underline{\theta(l)}_{\eta}\ \big(\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{\eta}\big) \end{array}\right); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to_{\mathbf{AC}}^{+} \left\langle \begin{array}{l} (\eta(l)) : \left(\begin{array}{l} \mathbf{case}\ \big(\mathbf{inj}_{\mathrm{Set}}\ \underline{V}_{\eta}\big)\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ () \mapsto \mathbf{let}\ y = \mathbf{yield}\ \underline{\theta(l)}_{\eta}\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_{\eta}\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ \mathit{loop}!\ v\ y\} \end{array}\right); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to_{\mathbf{AC}} \big\langle (\eta(l)) : \big(\mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_{\eta}\ y\big); \tau[(\eta(l)) \coloneqq \mathbf{nil}] \big\rangle \\
&\to_{\mathbf{AC}} \big\langle \mathbf{return}\ (); \tau[(\eta(l)) \coloneqq \big\{\lambda x.\ \mathbf{let}\ y = \mathbf{yield}\ \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_{\eta}\ y\big\}] \big\rangle
\end{aligned}$$

Therefore, we obtain

$$C' \sim \langle \mathbf{return}\ ();\tau[(\eta(l)) \coloneqq \{\lambda x.\ \mathbf{let}\ y = \mathbf{yield}\ \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{V}_\eta\ y\}]\rangle,$$

since the updated stores and η satisfy the IC.

Case (get):

Suppose that $C \equiv \langle \mathbf{get}\ l; \theta\rangle$ and $C' \equiv \langle \mathbf{return}\ V; \theta\rangle$, where $l \in \mathrm{Dom}(\theta)$ and $\theta(l) = V$. By the definition of $C \sim D$, there exists η such that

$$D \equiv \langle \mathbf{resume}\ (\eta(l))\ (\mathbf{inj}_{\mathrm{Get}}\ ());\tau\rangle$$

and

$$\tau(\eta(l)) = \{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y\}.$$

The reduction of D proceeds as follows:

$$\begin{aligned}
D &\equiv \langle \mathbf{resume}\ (\eta(l))\ (\mathbf{inj}_{\mathrm{Get}}\ ());\tau\rangle \\
&\to_{\mathbf{AC}} \left\langle \begin{array}{l} (\eta(l)) : (\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y\}!\ (\mathbf{inj}_{\mathrm{Get}}\ ())); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to^+_{\mathbf{AC}} \left\langle (\eta(l)) : (\mathit{loop}!\ \underline{\theta(l)}_\eta\ (\mathbf{inj}_{\mathrm{Get}}\ ()));\tau[(\eta(l)) \coloneqq \mathbf{nil}] \right\rangle \\
&\to^+_{\mathbf{AC}} \left\langle \begin{array}{l} (\eta(l)) : \left(\begin{array}{l} \left\{ \begin{array}{l} \lambda f.\lambda s.\lambda a.\ \mathbf{case}\ a\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ () \mapsto \mathbf{let}\ y = \mathbf{yield}\ s\ \mathbf{in}\ f!\ s\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ f!\ v\ y\} \end{array} \right\}! \\ \mathit{loop}\ \underline{\theta(l)}_\eta\ (\mathbf{inj}_{\mathrm{Get}}\ ()) \end{array} \right); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to^+_{\mathbf{AC}} \left\langle \begin{array}{l} (\eta(l)) : \left(\begin{array}{l} \mathbf{case}\ (\mathbf{inj}_{\mathrm{Get}}\ ())\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ () \mapsto \mathbf{let}\ y = \mathbf{yield}\ \underline{\theta(l)}_\eta\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ \mathit{loop}!\ v\ y\} \end{array} \right); \\ \tau[(\eta(l)) \coloneqq \mathbf{nil}] \end{array} \right\rangle \\
&\to_{\mathbf{AC}} \left\langle (\eta(l)) : (\mathbf{let}\ y = \underline{\theta(l)}_\eta\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y);\tau[(\eta(l)) \coloneqq \mathbf{nil}] \right\rangle \\
&\to_{\mathbf{AC}} \left\langle \mathbf{return}\ \underline{\theta(l)}_\eta;\tau[(\eta(l)) \coloneqq \{\lambda x.\ \mathbf{let}\ y = \mathbf{yield}\ \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y\}] \right\rangle
\end{aligned}$$

Therefore, we obtain

$$C' \sim \left\langle \mathbf{return}\ \underline{\theta(l)}_\eta;\tau[(\eta(l)) \coloneqq \{\lambda x.\ \mathbf{let}\ y = \mathbf{yield}\ \mathbf{return}\ x\ \mathbf{in}\ \mathit{loop}!\ \underline{\theta(l)}_\eta\ y\}] \right\rangle,$$

since the updated stores and η satisfy the IC.

We define a macro-translation on contexts by mapping holes to holes and let $\underline{\mathcal{K}}_\eta$ denote the translation of a context $\mathcal{K}$.

Lemma 35. $\underline{\mathcal{K}[M]}_\eta \equiv \underline{\mathcal{K}}_\eta[\underline{M}_\eta]$ *for an arbitrary context* K *and computation* M.

Proof. By straightforward induction on K.

Lemma 36. *If $\langle M;\theta\rangle \sim \langle N;\tau\rangle$ and M can be decomposed into an evaluation context $\mathcal{C}$ and a redex M', then there exists some η for which $N = \underline{M}_\eta$ can be also decomposed into an evaluation context $\underline{\mathcal{C}}_\eta$ and a redex $\underline{M'}_\eta$ such that $N = \underline{\mathcal{C}[M']}_\eta$.*

Proof. It is easy to check this by induction on the number of frames of $\mathcal{C}$. Observe that the extended macro-translation maps frames to frames and redexes to redexes.

Proposition 8 (Simulation). *Suppose that $C \sim D$ and $C \to_{\mathbf{R}} C'$, then there exists a* **AC** *configuration D' such that $D \to^{+}_{\mathbf{AC}} D'$ and $C' \sim D'$.*

Proof. By the definition of $C \sim D$, there exist M, θ, τ, and η such that $C \equiv \langle M;\theta\rangle$, $D \equiv \langle \underline{M}_\eta;\tau\rangle$, and θ, τ, and η satisfy the IC. Since C is not in normal form, M can be decomposed into an evaluation context $\mathcal{C}$ and a redex M'. By Lemma 36, $\underline{M}_\eta$ can also be decomposed into $\underline{\mathcal{C}}_\eta$ and a redex $\underline{M'}_\eta$, namely, $\underline{M}_\eta = \underline{\mathcal{C}}_\eta\big[\underline{M'}_\eta\big]$.

Now, suppose $\langle M';\theta\rangle \to^{\beta}_{\mathbf{R}} \langle M'';\theta'\rangle$. By Lemma 34, there exist η' and τ' such that $\big\langle \underline{M'}_\eta;\tau\big\rangle \to^{+}_{\mathbf{AC}} \big\langle \underline{M''}_{\eta'};\tau'\big\rangle$, $\langle M'';\theta'\rangle \sim \big\langle \underline{M''}_{\eta'};\tau'\big\rangle$, and θ', τ', and η' satisfy the IC. Moreover, $\underline{\mathcal{C}}_\eta \equiv \underline{\mathcal{C}}_{\eta'}$ holds since the labels in $\mathrm{Dom}(\eta') \setminus \mathrm{Dom}(\eta)$ are fresh and do not appear in C. Hence, we conclude that $\langle \mathcal{C}[M''];\theta'\rangle \sim \Big\langle \underline{\mathcal{C}[M'']}_{\eta'};\tau'\Big\rangle$ by Lemma 35.

Lemma 37. *For any* **REF***-computation M, $\langle M;\emptyset\rangle \sim \langle \underline{M};\emptyset\rangle$ holds.*

Proof. By induction on M.

Lemma 38. *We say a* **REF***-configuration $\langle M;\theta\rangle$ is well-formed if for any reference cell l that appears in M, $\theta(l)$ is defined, and this property is preserved under reduction.*

Proof. By straightforward case analysis on M.

Lemma 39. *Suppose that* $\mathrm{Eval}_{\mathbf{REF}}(M)$ *diverges, i.e., the reduction sequence of M is infinite. Then* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *also diverges.*

Proof. By applying Theorem 8 repeatedly, we also obtain an infinite reduction sequence of $\underline{M}$. Since the reduction of **AC** is deterministic, this implies that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges.

Lemma 40. *Suppose that* $\mathrm{Eval}_{\mathbf{REF}}(M)$ *gets stuck: there exists a term M' and a store θ such that $\langle M;\emptyset\rangle \to^{*}_{\mathbf{R}} \langle M';\theta\rangle$, $M' \neq$ **return** V for any value V, and there is no rule that can reduce $\langle M';\theta\rangle$. Then,* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *also gets stuck.*

Proof. It suffices to show the following statement:

> Suppose that there exist M, N, θ, τ, and η such that $N = \underline{M}_\eta$, $\langle M;\theta\rangle$ is well-formed, and θ, τ, and η satisfy the IC. If $\langle M;\theta\rangle$ is stuck, then $\langle N;\tau\rangle$ is also be stuck.

We prove by induction on the structure of M. However, we treat only three cases. The other cases follow by similar reasoning.

Case 1:

$$M \equiv \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M'$$

Since the $\langle M; \theta \rangle$ is stuck, we know that $V \neq (V_1, V_2)$ for any values V_1 and V_2. Consequently, $\underline{V}_\eta$ also cannot be of the form (W_1, W_2) for any values W_1 and W_2. Therefore, the configuration $\left\langle \underline{\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M}_\eta ; \tau \right\rangle$ is stuck, and this concludes the current case.

Case 2:

$$M \equiv \mathbf{set}\ V\ W$$

It follows that $V \neq l$ for any continuation label l, since if $V = l$ for some l, then by the well-formedness of $\langle \mathbf{throw}\ V\ W; \theta \rangle$, we know that $l \in \mathrm{Dom}(\theta)$. However, this contradicts that $\langle \mathbf{throw}\ V\ W; \theta \rangle$ is stuck. This implies that $\underline{\mathbf{set}\ V\ W}_\eta$ is also stuck, which concludes the current case.

Case 3:

$$M \equiv \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$$

Since $\langle M; \theta \rangle$ is stuck, M_1 cannot be of the form $\mathbf{return}\ V$ for any value V. By the induction hypothesis, we obtain that N_1 is stuck and that $N_1 \neq \mathbf{return}\ W$ for any value W. Consequently, the configuration $\left\langle \mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_\eta ; \tau \right\rangle$ is also stuck, which completes the current case.

We now present the proof of Theorem 7.

Complete proof of Theorem 7. We prove that $\mathrm{Eval}_{\mathbf{REF}}(M)$ terminates if and only if $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ terminates and show the "only if" direction first. Suppose that $\langle M; \emptyset \rangle \to_{\mathbf{R}}^{+} \langle \mathbf{return}\ V; \theta \rangle$ for some θ. By Lemma 37, we obtain $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$. By repeatedly applying Proposition 8, it follows that there exist η and τ such that

$$\langle \underline{M}; \emptyset \rangle \to_{\mathbf{AC}}^{+} \langle \mathbf{return}\ \underline{V}_\eta ; \tau \rangle.$$

Therefore, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ terminates.

Next, we prove the contrapositive of the "if" direction. There are two cases where $\mathrm{Eval}_{\mathbf{REF}}(M)$ does not terminate: (1) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ diverges, or (2) $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ gets stuck. In each case, Lemmas 39 and 40 imply that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges or gets stuck, respectively. Thus, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ does not terminate, which completes the proof.