Expressive power of one-shot control operators and coroutines

KENTARO KOBAYASHI
University of Tsukuba, Japan
(email: kentaro.kobayashi@acm.org*)*

YUKIYOSHI KAMEYAMA
University of Tsukuba, Japan
(email: 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.

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 [25, 26].

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 on control operators have focused on unrestricted (i.e., *multi-shot*) control operators,[1] 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 ones, as they avoid stack copying [3]. Second, they may alleviate the verification burden by reducing the complexity of reasoning about sensitive resources [6]. 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 the one-shot effect handlers of OCaml

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

DRAFT PAPER UNDER REVIEW

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 was 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 [5]. However, this correspondence relies heavily on mutable state 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., [17]), 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 expressive power of three one-shot control facilities: one-shot effect handlers, one-shot delimited-control operators, and coroutines [5]. We adopt macro-expressibility [8] as the basis for our comparison, since it is well-established in the literature, as in the study of control operators by Forster, Kammar, Lindley, and Pretnar [12]. 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.

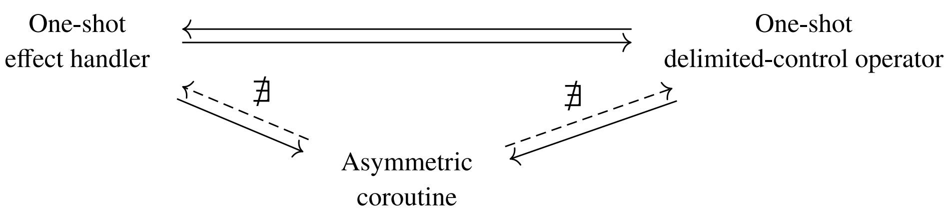

Fig. 1. Macro-expressibility among control operators. Solid lines indicate the existence of a macro-translation; dashed lines indicate its non-existence.

Figure 1 illustrates the macro-expressibility results established in this 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 [15].

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

Fig. 2. Syntax of **MAM**

This is the extended version of our paper presented at APLAS 2025 [19]. It differs from the conference version as follows:

(1) We add all key definitions and several important proof cases that could not be included for lack of space.
(2) We give a new translation from one-shot delimited-control operators to coroutines, which is much simpler than that presented in the conference version, and allows a more concise proof of correctness.
(3) We present a new inexpressibility result in Section 5.2, which completes the comparison of all the calculi introduced in this paper.

The remainder of this paper is organized as follows. In Section 2, we introduce the core calculus and define macro-expressibility. In Section 3, we introduce two calculi for one-shot delimited continuations and coroutines. We first demonstrate that a naïve translation from one-shot delimited continuations to coroutines fails to preserve semantics, analyzing the cause of failure. We then give a refined translation and prove its correctness. In Section 4, we introduce the calculus for one-shot effect handlers, and establish the equi-expressivity of one-shot effect handlers and one-shot delimited continuations. In Section 5, we describe the inexpressibility results. Finally, in Section 7, we conclude the paper. Proofs omitted from the main text appear in the appendices.

2 Core calculus and macro-expressibility

2.1 Core calculus MAM

First, we present the calculus **MAM** (multi-adjunctive metalanguage) by Forster et al. [12], which is designed after Levy's call-by-push-value [20]. This framework subsumes both call-by-value and call-by-name evaluation strategies and provides a uniform foundation to study various computational effects. In this paper, **MAM** serves as the common core calculus for our extensions.

Figure 2 gives the syntax of **MAM**. In this language, values and computations are clearly separated into two different syntactic categories.

Values, ranged over by the metavariables V and W, are the data consumed by computations. As usual, this category includes variables (drawn from a countable set $\mathscr{V}$), the unit value, pairings (V, W) (products), and variants $\mathbf{inj}_{\mathrm{L}}\, V$ (sums, whose tags L range over a finite set $\mathscr{C}$). In addition, a thunk $\{M\}$ is a value that suspends a computation M and reifies it as data.

Computations, ranged over by the metavariables M and N, consume such data and, when they terminate, return a value. A computation returns a value V via the returner **return** V,

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

Fig. 3. Frames and contexts of **MAM**

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

Fig. 4. Beta reduction rules of **MAM**

and two computations can be sequenced by **let** $x = M$ **in** N: once M returns a value, that value is substituted for x in N. The pattern-matching constructs **case** V **of** $(x_1, x_2) \mapsto M$ and **case** V **of** $\{(\mathbf{inj}_{\mathrm{L}_i}\ x_i \mapsto M_i)_i\}$ match on a product and a variant value, respectively. A force term $V!$ runs the computation represented by V; in particular, if V is a thunk $\{M\}$, $\{M\}!$ reduces to the computation M. An abstraction $\lambda x.\ M$ binds a variable x in computation M, and an application $M\ V$ supplies a value V to such an abstraction. Since an abstraction is classified as a computation, to pass it as a value to other computations, it must be suspended as a thunk. Finally, $\langle M, N\rangle$ pairs two computations M and N, and $\mathbf{prj}_i\ M$ takes the i-th component of such a pairing, reducing to that component.

Throughout the paper, we freely use nested patterns and a catch-all clause in the pattern-matching constructs; e.g., we write a term such as **case** V **of** $\{\mathbf{inj}_{\mathrm{L}}\ (x_1, x_2) \mapsto M,\ \_ \mapsto N\}$. This is only a superficial extension: such a term can be desugared into a combination of pattern-matching constructs. We also write _ for a variable that does not occur free in the body, as in **let** $\_ = M$ **in** N.

We consider terms modulo renaming of bound variables as usual. We define the set of **MAM** *programs* to be the set of all computations.

We give the operational semantics of **MAM** using evaluation contexts [9]: we decompose a computation into an evaluation context and a redex, and reduce the latter using beta reduction rules. Figure 3 defines frames and contexts. Evaluation contexts consist of computational frames, while pure contexts comprise only pure frames. We use an evaluation context to locate the redex as usual; a pure context, in turn, identifies the part of an evaluation context that has no (immediate) computational effects. Although these two notions of contexts agree in **MAM** (since it has no computational effects), in each extension of **MAM** introduced below, we add computational frames according to the control abstraction it offers.

Figure 4 defines the beta reduction rules $\to^{\beta}_{\mathbf{M}}$ on computations. $M[V_1/x_1, \ldots, V_n/x_n]$ denotes the simultaneous, capture-avoiding substitution of each x_i with V_i in M. We define the transition relation $\to_{\mathbf{M}}$ on computations by using evaluation contexts as follows:

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

We remark that the decomposition of a computation into an evaluation context and a redex is unique.

Evaluation of a program is defined by: $\mathrm{Eval}_{\mathbf{MAM}}(M) \stackrel{\text{def}}{=} V$ if $M \to_{\mathbf{M}}^{*}$ **return** V.[4] $\mathrm{Eval}_{\mathbf{MAM}}$ is a well-defined partial function, since $\to_{\mathbf{M}}$ is deterministic: there exists a unique maximal reduction sequence starting from M. It is defined at M exactly when the evaluation of M terminates successfully.

Example 2.1. We assume that **MAM** is extended with integers and arithmetic operators.[5] Then, $\mathrm{Eval}_{\mathbf{MAM}}((\lambda f.\ f\,!\,2)\,\{\lambda x.\ x+1\}) = 3$. In particular, the reduction proceeds as follows:

$$\begin{aligned}(\lambda f.\ f\,!\,2)\,\{\lambda x.\ x+1\} &\to_{\mathbf{M}} \{\lambda x.\ x+1\}\,!\,2\\ &\to_{\mathbf{M}} (\lambda x.\ x+1)\,2\\ &\to_{\mathbf{M}} 2+1 \to_{\mathbf{M}} 3.\end{aligned}$$

2.2 Macro-expressibility

To compare the expressive power of one-shot control operators, we use the notion of *macro-expressibility*, which was originally proposed by Felleisen [8], and has been adopted in a number of studies on control operators. This notion serves as a finer basis for comparison than computability, and is especially useful when comparing the expressive power of extensions of a Turing-complete language, such as **MAM**.

Before defining the notion of *macro-expressibility* formally, let us clarify the target of comparison. An extension of **MAM** considered in this paper should contain the syntactic categories of values and computations, and may add finitely many syntactic categories. It may also have additional constructs. For each extension, the set of programs and the evaluation function should be defined explicitly.[6]

Definition 2.2. Let $\mathscr{L}_1$ and $\mathscr{L}_2$ be extensions of **MAM**. A partial function ϕ from $\mathscr{L}_1$-terms to $\mathscr{L}_2$-terms is a (strong) *macro-translation* if and only if the following conditions are satisfied:

(1) If M is an $\mathscr{L}_1$-program, then $\phi(M)$ is defined and an $\mathscr{L}_2$-program;

(2) If F is an n-ary constructor of **MAM**, then for any $\mathscr{L}_1$-terms $M_1, \dots, M_n$,
$$\phi(F(M_1, \dots, M_n)) = F(\phi(M_1), \dots, \phi(M_n));$$

(3) For each n-ary constructor $F \in \mathscr{L}_1 \setminus \mathbf{MAM}$, there is an n-hole syntactic abstraction[7] $\mathcal{A}$ in $\mathscr{L}_2$ such that
$$\phi(F(M_1, \dots, M_n)) = \mathcal{A}[\phi(M_1), \dots, \phi(M_n)]$$
for any $\mathscr{L}_1$-terms $M_1, \dots, M_n$;

(4) For every $\mathscr{L}_1$ program M, $\mathrm{Eval}_{\mathscr{L}_1}(M)$ is defined if and only if $\mathrm{Eval}_{\mathscr{L}_2}(\phi(M))$ is defined.

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 define a *weak* macro-translation by replacing "if and only if" in (4) with "only if".

[4] We write $\to^*$ for the reflexive transitive closure of a binary relation $\to$, and $\to^+$ for its transitive closure.

[5] Henceforth, we freely use this extension when we give examples.

[6] Our treatment of extensions of **MAM** is deliberately made somewhat informal. We could have defined a general notion of languages and macro-expressibility over them, but it would involve complex arguments about binding structures, possibly using the theory of abstract syntax [10]. Such a rigorous treatment is necessary when developing a general theory of macro-expressibility; however, in this paper, we are interested in the expressive power of several fixed calculi.

[7] An n-hole syntactic abstraction in $\mathscr{L}$ is an $\mathscr{L}$-term with n holes.

$$
\begin{array}{rcll}
V, W & ::= & \dots & \text{value} \\
& | & l \in \mathscr{L}_{\mathbf{D}} & \text{continuation label} \\
M, N & ::= & \dots & \text{computation} \\
& | & \mathbf{S_0} k.\ M & \text{shift0} \\
& | & \langle M \,|\, x.N \rangle & \text{dollar} \\
& | & \mathbf{throw}\ V\ W & \text{throw}
\end{array}
$$

$$
\begin{array}{rrcl}
\text{pure frame} & \mathcal{P} & ::= & \dots \\
\text{computational frame} & \mathcal{F} & ::= & \dots \mid \langle [\ ] \,|\, x.N \rangle \\
\text{pure context} & \mathcal{H} & ::= & \dots \\
\text{evaluation context} & \mathcal{C} & ::= & \dots
\end{array}
$$

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

Example 2.3. An example of macro-expressibility is the sequencing $M; N$, which evaluates M first, discards the result, and then evaluates N. Consider extending **MAM** with this sequencing constructor. Then we define a macro-translation ϕ from this extension to **MAM** such that $\phi(M; N) = (\mathbf{let}\ \_ = \phi(M)\ \mathbf{in}\ \phi(N))$ holds. Other constructors, such as the abstraction $\lambda x.\ M$, are translated homomorphically: $\phi(\lambda x.\ M) = \lambda x.\ \phi(M)$.

We remark that macro-expressibility is reflexive and transitive: an identity translation is a macro-translation, and the composition of macro-translations is also a macro-translation.

3 One-shot delimited continuations as asymmetric coroutines

This section investigates the macro-expressibility of one-shot delimited continuations in terms of asymmetric coroutines. Although this expressibility had been considered trivially true, establishing it turns out to be unexpectedly complicated. We will first introduce two calculi: $\mathbf{DEL}_{\text{one}}$ for one-shot delimited continuations and **AC** for asymmetric coroutines. We will then present a simple translation from $\mathbf{DEL}_{\text{one}}$ to **AC**, and explain why this translation fails. Finally, we will give a refined translation which is proved to be a correct macro-translation.

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 *shift0/dollar*. The *shift0/dollar* operator [18, 21] is a variant of *shift0/reset0* [4] and has the same expressive power. We adopt the calculus **DEL** defined by Forster et al. [12], restricting the use of each captured continuation to one-shot.

Figure 5 shows the syntax of $\mathbf{DEL}_{\text{one}}$ as an extension of **MAM**, where the ellipses (. . .) indicate the syntax from **MAM**. A continuation label l is a value ranging over a countable set $\mathscr{L}_{\mathbf{D}}$. It is used to refer to a captured continuation, as described below. The computation $\langle M \,|\, x.N \rangle$ represents a *generalized reset0*, called *dollar*. When evaluated, this term 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. The computation $\mathbf{S_0} k.\ M$ represents the control operator *shift0*. When this term is evaluated, a continuation delimited by the nearest dollar term is captured and assigned a fresh continuation label; k is bound to this label, and the body M is evaluated. If there is no surrounding dollar term, the evaluation gets stuck. The computation $\mathbf{throw}\ V\ W$ represents the invocation of the

$$
\begin{array}{ll}
(\textbf{MAM}) & \dfrac{M \to_{\mathbf{M}}^{\beta} M'}{\langle M;\theta\rangle \to_{\mathbf{D}}^{\beta} \langle M';\theta\rangle} \\
(\text{ret}) & \langle\langle \textbf{return}\, V \,|\, x.M\rangle;\theta\rangle \to_{\mathbf{D}}^{\beta} \langle M[V/x];\theta\rangle \\
(\text{shift}) & \dfrac{l \notin \mathrm{Dom}(\theta)}{\langle\langle \mathcal{H}[\mathbf{S_0} k.\ M] \,|\, x.N\rangle;\theta\rangle \to_{\mathbf{D}}^{\beta} \langle M[l/k];\theta[l := \lambda y.\ \langle \mathcal{H}[\textbf{return}\, y] \,|\, x.N\rangle]\rangle} \\
(\text{throw}) & \dfrac{\theta(l) = \lambda y.\ \langle \mathcal{H}[\textbf{return}\, y] \,|\, x.N\rangle}{\langle \textbf{throw}\, l\, V;\theta\rangle \to_{\mathbf{D}}^{\beta} \langle\langle \mathcal{H}[\textbf{return}\, V] \,|\, x.N\rangle;\theta[l := \textbf{nil}]\rangle} \\
(\text{fail}) & \dfrac{\theta(l) = \textbf{nil}}{\langle \textbf{throw}\, l\, V;\theta\rangle \to_{\mathbf{D}}^{\beta} \bot}
\end{array}
$$

Fig. 6. Beta reduction rules of $\mathbf{DEL_{one}}$

captured continuation associated with V with the argument W. We define $\mathbf{DEL_{one}}$ programs as the set of computations containing no continuation labels.

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

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

The term attempts to invoke the captured continuation k twice, first to compute a and then to compute b, which violates the one-shotness constraint. Since whether a continuation has already been invoked can only be determined at runtime, we detect such violations dynamically, using a *store* to record the content and status of continuations.

A store is a partial function $\theta : \mathscr{L}_{\mathbf{D}} \rightharpoonup \mathsf{computation} \sqcup \{\textbf{nil}\}$. $\mathrm{Dom}(\theta)$ is the set of continuation labels l such that $\theta(l)$ is defined. Note that $\theta(l) = \textbf{nil}$ does *not* mean that $\theta(l)$ is undefined. Instead, $\theta(l) = \textbf{nil}$ means that the continuation labeled l is *invalid*—that is, this continuation has already been invoked. If $\theta(l) = M$ for some computation M, we say that the continuation labeled l is *valid*. The empty set $\emptyset$ represents the store whose domain is empty. We write $\theta[l := M]$ for the usual store update.

We introduce *configurations* to describe the runtime states of programs. A configuration C is either a pair of a $\mathbf{DEL_{one}}$ computation M and a store θ, written as $\langle M;\theta\rangle$, or the error state $\bot$. The beta reduction rules $\to_{\mathbf{D}}^{\beta}$ on configurations are defined in Figure 6.[8] In (shift), we assume that y does not freely occur in the context $\mathcal{H}$. Throughout this paper, we assume similar freshness conditions for the variables introduced in reduction rules and translations.

The reduction rules of $\mathbf{DEL_{one}}$ are defined as follows:

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

[8] Here $\to_{\mathbf{M}}^{\beta}$ is understood as the relation on $\mathbf{DEL_{one}}$ computations obtained by reading the beta-reduction rules of **MAM** in Figure 4 as rules over $\mathbf{DEL_{one}}$ computations. We adopt the same reading for the calculi introduced in later sections.

$$
\begin{array}{rcll}
V, W & ::= & \dots & \text{value} \\
& | & l \in \mathscr{L}_{\mathbf{AC}} & \text{coroutine label} \\
M, N & ::= & \dots & \text{computation} \\
& | & \mathbf{yield}\, V & \text{yield} \\
& | & \mathbf{create}\, V & \text{create} \\
& | & \mathbf{resume}\, V\, W & \text{resume} \\
& | & l : M & \text{labeled computation}
\end{array}
$$

$$
\begin{array}{rcll}
\text{pure frame} & \mathcal{P} & ::= & \dots \\
\text{computational frame} & \mathcal{F} & ::= & \dots \mid l : [\ ] \\
\text{pure context} & \mathcal{H} & ::= & \dots \\
\text{evaluation context} & C & ::= & \dots
\end{array}
$$

Fig. 7. Syntax of **AC**

Evaluation of a $\mathbf{DEL}_{\mathrm{one}}$-program is defined by: $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M) \stackrel{\mathrm{def}}{=} V$ if there exists a store θ such that $\langle M; \emptyset\rangle \to^{*}_{\mathbf{D}} \langle \mathbf{return}\, V; \theta\rangle$. $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}$ is a well-defined partial function, since $\to_{\mathbf{D}}$ is deterministic. [9]

Example 3.1. The evaluation of the program $M \stackrel{\mathrm{def}}{=} \langle (\mathbf{S_0} k.\ ((\mathbf{throw}\, k\, 2) + 3)) | x.x + 4\rangle$ proceeds as follows.

$$
\begin{aligned}
\langle M; \emptyset\rangle &\to_{\mathbf{D}} \langle (\mathbf{throw}\, l\, 2) + 3;\ \emptyset[l := \lambda y.\ \langle \mathbf{return}\, y | x.x + 4\rangle]\rangle \\
&\to_{\mathbf{D}} \langle \langle \mathbf{return}\, 2 | x.x + 4\rangle + 3;\ \emptyset[l := \mathbf{nil}]\rangle \\
&\to_{\mathbf{D}} \langle (2 + 4) + 3;\ \emptyset[l := \mathbf{nil}]\rangle \\
&\to^{+}_{\mathbf{D}} \langle 9;\ \emptyset[l := \mathbf{nil}]\rangle.
\end{aligned}
$$

Thus, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M) = 9$.

3.2 The calculus for asymmetric coroutines

A coroutine is a computation that can suspend its execution and later resume from the point of suspension, in contrast to an ordinary subroutine. De Moura and Ierusalimschy studied several variations of coroutines found in programming languages and introduced two calculi, symmetric coroutines and asymmetric coroutines [5]. We choose asymmetric coroutines because they are the most expressive among several variations of coroutines and "much simpler to manage and understand" from the programmer's perspective [5]. They are also prevalent in modern programming languages, such as Lua and Ruby, while symmetric coroutines are adopted in "classical" languages, such as Simula and Modula-2.

We define the calculus **AC** for asymmetric coroutines, employing the basic constructs of de Moura and Ierusalimschy's calculus, and adapting these constructs to the call-by-push-value style. Figure 7 presents the syntax of **AC**. Besides the constructs 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 countable set of coroutine labels. The labeled computation $l : M$ represents the coroutine l executing the computation M. The

[9] Strictly speaking, there is an ambiguity in the choice of a fresh label, which implies the nondeterminism of the reduction. To avoid this, we fix a linear order on $\mathscr{L}_{\mathbf{D}}$ and take the least label not in $\mathrm{Dom}(\theta)$ as a fresh label. We adopt the same convention for the calculi introduced in later sections.

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

$$
\dfrac{\langle M; \theta\rangle \rightarrow^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle}{\langle C[M]; \theta\rangle \rightarrow_{\mathbf{AC}} \langle C[M']; \theta'\rangle} \qquad\qquad \dfrac{\langle M; \theta\rangle \rightarrow^{\beta}_{\mathbf{AC}} \bot}{\langle C[M]; \theta\rangle \rightarrow_{\mathbf{AC}} \bot}
$$

Fig. 8. Semantics of **AC**

computation **create** V produces a new coroutine whose body is V, creating a fresh label for it. The computation **resume** $V\,W$ invokes a coroutine labeled V, with a parameter W. The computation **yield** V suspends the current coroutine, yielding V to its caller. We define **AC** programs as the set of computations containing no coroutine labels.

Similarly to $\mathbf{DEL}_{\text{one}}$, we use a store to track the content and status of coroutines. 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**. A configuration in **AC** is either 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 $\rightarrow^{\beta}_{\mathbf{AC}}$ and the reduction rules $\rightarrow_{\mathbf{AC}}$. Note that coroutines are inherently one-shot: although a coroutine may be called multiple times by suspending and resuming it, it cannot be duplicated or reused.

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

Example 3.2. The evaluation of the program M, defined by

$$
M \overset{\text{def}}{=} \left(\begin{array}{l}
\mathbf{let}\, f = \mathbf{create} \left\{\begin{array}{l}\lambda i.\, \mathbf{let}\, j = \mathbf{yield}\, i\, \mathbf{in} \\ \quad j + 1\end{array}\right\} \mathbf{in} \\
\mathbf{let}\, a = \mathbf{resume}\, f\, 2\, \mathbf{in} \\
\mathbf{let}\, b = \mathbf{resume}\, f\, 3\, \mathbf{in} \\
\mathbf{return}\, (a, b)
\end{array}\right),
$$

proceeds as follows.

$$
\langle M; \emptyset\rangle \rightarrow_{\mathbf{AC}} \left\langle \left(\begin{array}{l}
\mathbf{let}\, f = \mathbf{return}\, l\, \mathbf{in} \\
\mathbf{let}\, a = \mathbf{resume}\, f\, 2\, \mathbf{in} \\
\mathbf{let}\, b = \mathbf{resume}\, f\, 3\, \mathbf{in} \\
\mathbf{return}\, (a, b)
\end{array}\right);\ \emptyset\left[l := \left\{\begin{array}{l}\lambda i.\, \mathbf{let}\, j = \mathbf{yield}\, i\, \mathbf{in} \\ \quad j + 1\end{array}\right\}\right] \right\rangle
$$

$$
\begin{array}{rcl}
\underline{\langle M \,|\, x.N \rangle} & \stackrel{\text{def}}{=} & \left(\begin{array}{l}\textbf{let}\ z = \textbf{create}\ \{\lambda\_.\ \textbf{let}\ x = \underline{M}\ \textbf{in return}\ \{\lambda\_.\ \underline{N}\}\}\ \textbf{in} \\ \textbf{let}\ res = \textbf{resume}\ z\ ()\ \textbf{in} \\ res!\ z\end{array}\right) \\
\underline{\mathbf{S_0}k.\ L} & \stackrel{\text{def}}{=} & \textbf{yield}\ \{\lambda k.\ \underline{L}\} \\
\underline{\textbf{throw}\ V\ W} & \stackrel{\text{def}}{=} & \left(\begin{array}{l}\textbf{let}\ res = \textbf{resume}\ \underline{V}\ \underline{W}\ \textbf{in} \\ res!\ \underline{V}\end{array}\right)
\end{array}
$$

Fig. 9. Naïve translation from $\mathbf{DEL_{one}}$ to $\mathbf{AC}$

$$
\begin{array}{ll}
\rightarrow_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ a = \textbf{resume}\ l\ 2\ \textbf{in} \\ \textbf{let}\ b = \textbf{resume}\ l\ 3\ \textbf{in} \\ \textbf{return}\ (a, b)\end{array}\right);\ \emptyset\left[l := \left\{\begin{array}{l}\lambda i.\ \textbf{let}\ j = \textbf{yield}\ i\ \textbf{in} \\ \quad j + 1\end{array}\right\}\right] \right\rangle \\
\rightarrow^{+}_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ a = l : \left(\begin{array}{l}\lambda i.\ \textbf{let}\ j = \textbf{yield}\ i\ \textbf{in} \\ \quad j + 1\end{array}\right) 2\ \textbf{in} \\ \textbf{let}\ b = \textbf{resume}\ l\ 3\ \textbf{in} \\ \textbf{return}\ (a, b)\end{array}\right);\ \emptyset[l := \textbf{nil}] \right\rangle \\
\rightarrow_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ a = l : \left(\begin{array}{l}\textbf{let}\ j = \textbf{yield}\ 2\ \textbf{in} \\ j + 1\end{array}\right)\ \textbf{in} \\ \textbf{let}\ b = \textbf{resume}\ l\ 3\ \textbf{in} \\ \textbf{return}\ (a, b)\end{array}\right);\ \emptyset[l := \textbf{nil}] \right\rangle \\
\rightarrow_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ a = \textbf{return}\ 2\ \textbf{in} \\ \textbf{let}\ b = \textbf{resume}\ l\ 3\ \textbf{in} \\ \textbf{return}\ (a, b)\end{array}\right);\ \emptyset\left[l := \left\{\begin{array}{l}\lambda y.\ \textbf{let}\ j = \textbf{return}\ y\ \textbf{in} \\ \quad j + 1\end{array}\right\}\right] \right\rangle \\
\rightarrow_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ b = \textbf{resume}\ l\ 3\ \textbf{in} \\ \textbf{return}\ (2, b)\end{array}\right);\ \emptyset\left[l := \left\{\begin{array}{l}\lambda y.\ \textbf{let}\ j = \textbf{return}\ y\ \textbf{in} \\ \quad j + 1\end{array}\right\}\right] \right\rangle \\
\rightarrow^{+}_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ b = l : \left(\begin{array}{l}\lambda y.\ \textbf{let}\ j = \textbf{return}\ y\ \textbf{in} \\ \quad j + 1\end{array}\right) 3\ \textbf{in} \\ \textbf{return}\ (2, b)\end{array}\right);\ \emptyset[l := \textbf{nil}] \right\rangle \\
\rightarrow_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ b = l : \left(\begin{array}{l}\textbf{let}\ j = \textbf{return}\ 3\ \textbf{in} \\ j + 1\end{array}\right)\ \textbf{in} \\ \textbf{return}\ (2, b)\end{array}\right);\ \emptyset[l := \textbf{nil}] \right\rangle \\
\rightarrow^{+}_{\mathbf{AC}} & \left\langle \left(\begin{array}{l}\textbf{let}\ b = l : \textbf{return}\ 4\ \textbf{in} \\ \textbf{return}\ (2, b)\end{array}\right);\ \emptyset[l := \textbf{nil}] \right\rangle \\
\rightarrow^{+}_{\mathbf{AC}} & \langle \textbf{return}\ (2, 4);\ \emptyset[l := \textbf{nil}] \rangle
\end{array}
$$

Thus, $\mathrm{Eval}_{\mathbf{AC}}(M) = (2, 4)$.

3.3 Naïve translation and its failure

Given the similarity between delimited continuations and coroutines, one may think that it is straightforward to macro-translate $\mathbf{DEL_{one}}$ to $\mathbf{AC}$ with the following correspondence: a dollar term in $\mathbf{DEL_{one}}$ 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 naïve 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 naïve translation works for simple expressions. However, it fails to preserve semantics, as shown below. Consider the following $\mathbf{DEL}_{\text{one}}$-program M:

$$\begin{aligned} M &\stackrel{\text{def}}{=} \langle M' \mid i.\, \mathbf{return}\, i\rangle \\ M' &\stackrel{\text{def}}{=} \left(\begin{array}{l} \mathbf{let}\, j = (\mathbf{S_0} k_1.\ \mathbf{let}\ r_1 = \mathbf{throw}\, k_1\ 10\ \mathbf{in} \\ \qquad\qquad\qquad\quad \mathbf{let}\ r_2 = \mathbf{throw}\, k_1\ 20\ \mathbf{in}\ \mathbf{return}\, r_1)\ \mathbf{in} \\ \mathbf{S_0} k_2.\ \mathbf{return}\, 30 \end{array}\right) \end{aligned}$$

We shall explain how the evaluation of M proceeds in $\mathbf{DEL}_{\text{one}}$. (For brevity, the full content of the store is omitted from the evaluation trace. Only the changes relevant to the main computation are mentioned.) First, the shift0 term $\mathbf{S_0} k_1.\ \cdots$ is invoked, and the pure context surrounding it is captured and stored under a fresh label l_1:

$$M \to_{\mathbf{D}} \left(\begin{array}{l} \mathbf{let}\ r_1 = \mathbf{throw}\, l_1\ 10\ \mathbf{in} \\ \mathbf{let}\ r_2 = \mathbf{throw}\, l_1\ 20\ \mathbf{in}\ \mathbf{return}\, r_1 \end{array}\right) \tag{1}$$
$$[\text{where } l_1 \mapsto \lambda y.\ \langle \mathbf{let}\, j = \mathbf{return}\, y\ \mathbf{in}\ \mathbf{S_0} k_2.\ \mathbf{return}\, 30 |\, i.\mathbf{return}\, i\rangle]$$

Next, $\mathbf{throw}\, l_1$ 10 invokes the captured continuation labeled l_1, invalidating the continuation:

$$\cdots \to^{+}_{\mathbf{D}} \left(\begin{array}{l} \mathbf{let}\ r_1 = \left\langle \begin{array}{l} \mathbf{let}\, j = \mathbf{return}\, 10\ \mathbf{in} \\ \mathbf{S_0} k_2.\ \mathbf{return}\, 30 \end{array} \middle|\ i.\, \mathbf{return}\, i \right\rangle\ \mathbf{in} \\ \mathbf{let}\ r_2 = \mathbf{throw}\, l_1\ 20\ \mathbf{in}\ \mathbf{return}\, r_1 \end{array}\right) \tag{2}$$
$$[\text{where } l_1 \mapsto \mathbf{nil}]$$
$$\to_{\mathbf{D}} \left(\begin{array}{l} \mathbf{let}\ r_1 = \langle \mathbf{S_0} k_2.\ \mathbf{return}\, 30 \mid i.\, \mathbf{return}\, i\rangle\ \mathbf{in} \\ \mathbf{let}\ r_2 = \mathbf{throw}\, l_1\ 20\ \mathbf{in}\ \mathbf{return}\, r_1 \end{array}\right)$$

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):

$$\cdots \to^{+}_{\mathbf{D}} \left(\begin{array}{l} \mathbf{let}\ r_1 = \mathbf{return}\, 30\ \mathbf{in} \\ \mathbf{let}\ r_2 = \mathbf{throw}\, l_1\ 20\ \mathbf{in}\ \mathbf{return}\, r_1 \end{array}\right) \tag{3}$$
$$[\text{where } l_2 \mapsto \lambda y.\ \langle \mathbf{return}\, y |\, i.\mathbf{return}\, i\rangle]$$
$$\to_{\mathbf{D}} \mathbf{let}\ r_2 = \mathbf{throw}\, l_1\ 20\ \mathbf{in}\ \mathbf{return}\, 30 \tag{4}$$

Finally, $\mathbf{throw}\, l_1$ 20 is invoked, but since the continuation labeled l_1 has already been consumed, the evaluation fails.

Therefore, in $\mathbf{AC}$, the evaluation of $\underline{M}$ must not successfully terminate since macro-translations preserve semantics; however, this is not the case. To understand the reason, consider the evaluation trace of $\underline{M}$. First, after applying the translation, a coroutine corresponding to the body of the source dollar term is created, labeled as m, and invoked ($\mathcal{P}$ is a pure context defined as **let** i = [ ] **in return** $\{\lambda\_.\ \textbf{return}\, i\}$):

$$\underline{M} \equiv \begin{pmatrix} \textbf{let}\ z = \textbf{create}\left\{\lambda\_.\ \mathcal{P}\left[\underline{M'}\right]\right\}\ \textbf{in} \\ \textbf{let}\ res = \textbf{resume}\, z\ ()\ \textbf{in} \\ res!\, z \end{pmatrix}$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ res = \\ \quad m : \mathcal{P}\begin{bmatrix} \textbf{let}\ j = \\ \quad \textbf{yield}\left\{\lambda k_1.\ \begin{pmatrix} \textbf{let}\ r_1 = \underline{\textbf{throw}}\, k_1\ 10\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, k_1\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix}\right\}\ \textbf{in} \\ \textbf{yield}\ \{\lambda k_2.\ \textbf{return}\, 30\} \end{bmatrix}\ \textbf{in} \\ res!\, m \end{pmatrix}$$

$$[\text{where } m \mapsto \textbf{nil}]$$

Then, the current coroutine m is suspended, yielding the thunk $\{\lambda k_1.\ \cdots\}$.

$$\cdots \rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ res = \textbf{return}\left\{\lambda k_1.\ \begin{pmatrix} \textbf{let}\ r_1 = \underline{\textbf{throw}}\, k_1\ 10\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, k_1\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix}\right\}\ \textbf{in} \\ res!\, m \end{pmatrix}$$

$$\begin{bmatrix} \text{where} \\ m \mapsto \left\{\lambda y.\ \mathcal{P}\begin{bmatrix} \textbf{let}\ j = \textbf{return}\, y\ \textbf{in} \\ \textbf{yield}\ \{\lambda k_2.\ \textbf{return}\, 30\} \end{bmatrix}\right\} \end{bmatrix}$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_1 = \underline{\textbf{throw}}\, m\ 10\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, m\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix}$$

This term corresponds to (1), where the continuation captured by $\mathbf{S_0}k_1.\ \cdots$ in M, labeled l_1, corresponds to the coroutine labeled m. Next, $\underline{\textbf{throw}}\, m\ 10$ is evaluated. This resumes the coroutine labeled m, mapping m to **nil** in the store, and the evaluation reaches the following term corresponding to (2):

$$\cdots \rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_1 = \begin{pmatrix} \textbf{let}\ res = m : \left(\mathcal{P}\begin{bmatrix} \textbf{let}\ j = \textbf{return}\, 10\ \textbf{in} \\ \textbf{yield}\ \{\lambda k_2.\ \textbf{return}\, 30\} \end{bmatrix}\right)\ \textbf{in} \\ res!\, m \end{pmatrix}\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, m\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix} \tag{5}$$

$$[\text{where } m \mapsto \textbf{nil}]$$

Then, the value 10 is discarded, and the term **yield** $\{\lambda k_2.\ \textbf{return}\, 30\}(= \underline{\mathbf{S_0}k_2.\ \textbf{return}\, 30})$ is invoked:

$$\cdots \rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_1 = \{\lambda k_2.\ \textbf{return}\, 30\}!\, m\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, m\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix} \begin{bmatrix} \text{where} \\ m \mapsto \left\{\lambda y.\ \mathcal{P}[\textbf{return}\, y]\right\} \end{bmatrix}$$

$$\cdots \rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_1 = \textbf{return}\, 30\ \textbf{in} \\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\, m\ 20\ \textbf{in} \\ \textbf{return}\, r_1 \end{pmatrix}$$

In this term, which corresponds to (3), the label m substituted for k_2 is discarded without being consumed. Thus, the invocation of $\underline{\textbf{throw}}\, m\, 20$ (corresponding to that of $\textbf{throw}\, l_1\, 20$ in (4)) succeeds, although it should not, and the evaluation terminates successfully:

$$\dots \to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_2 = \underline{\textbf{throw}}\, m\ 20\ \textbf{in} \\ \textbf{return}\, 30 \end{pmatrix} \begin{bmatrix} \text{where} \\ m \mapsto \{\lambda y.\, \mathcal{P}[\textbf{return}\, y]\} \end{bmatrix}$$

$$\to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_2 = \begin{pmatrix} \textbf{let}\ res = m : (\textbf{let}\ i = \textbf{return}\, 20\ \textbf{in return}\, \{\lambda\_.\ \textbf{return}\, i\})\ \textbf{in} \\ res!\, m \end{pmatrix} \textbf{in} \\ \textbf{return}\, 30 \end{pmatrix}$$

$$[\text{where } m \mapsto \textbf{nil}]$$

$$\to^{+}_{\mathbf{AC}} \begin{pmatrix} \textbf{let}\ r_2 = \{\lambda\_.\ \textbf{return}\, 20\}!\, m\ \textbf{in} \\ \textbf{return}\, 30 \end{pmatrix}$$

$$\to^{+}_{\mathbf{AC}} \textbf{return}\, 30$$

In short, the naïve translation fails because it ignores the difference between $\mathbf{DEL}_{\mathrm{one}}$ and **AC** in how the validity of labels is managed. In $\mathbf{DEL}_{\mathrm{one}}$, validity is *one-way*: once a shift0 invocation creates a fresh label l, this label remains valid—i.e., $\theta(l)$ holds the captured continuation—until the continuation is invoked. After that, $\theta(l)$ becomes **nil** permanently. For example, once the continuation labeled l_1 is invoked in (2), it is kept invalid throughout the evaluation. In **AC**, by contrast, validity is *alternating*: each time coroutine m is suspended and later resumed, $\theta(m)$ alternates between a value (the coroutine m is suspended, hence valid) and **nil** (the coroutine is running, hence invalid).

Consequently, since both l_1 and l_2 are mapped to m, the generation of l_2 corresponds to the suspension of m, and the second invocation of l_1 corresponds to its resumption. This resumption is legitimate in **AC**, as the coroutine m is suspended. Interpreted back in $\mathbf{DEL}_{\mathrm{one}}$'s terms, however, this means that l_1 is illegitimately *reactivated* by the creation of l_2. This is the reason why the naïve translation fails to preserve the semantics.

We remark that it would be harmless if the captured continuation labeled l_2 were consumed (or, equivalently, the thunk returned by **yield** in (5) eventually resumed m), as that would invalidate m again. In our counterexample, however, this is not the case.

This phenomenon had been overlooked for years. In fact, it 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 [17]. 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.[10]

3.4 *Refined translation from* $\mathbf{DEL}_{\mathrm{one}}$ *to* AC

Behind the failure analyzed in the previous section lie two gaps to be addressed:

(1) Each shift0 invocation creates a fresh continuation label, whereas a coroutine label, once created, is reused across resumptions. A single coroutine label therefore has to serve all the continuations captured within one dollar term.

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

(2) The validity of a continuation label is one-way, whereas that of a coroutine label is alternating.

The naïve translation addresses neither.

Our key idea to fill the gaps is to map each continuation label not directly to a coroutine label, but to a *reference cell* that holds both the continuation label's validity and a coroutine label. Specifically, a reference cell holds either $\mathbf{inj}_{\mathrm{Valid}}\, m$, meaning that the continuation is still valid and is realized by the coroutine labeled m, or $\mathbf{inj}_{\mathrm{Invalid}}\, ()$, meaning that it has already been invoked. A continuation is then represented by the label of a cell, while the continuation's validity is recorded in the content of the cell. Thus, invoking a continuation through one copy of its label invalidates all the other copies as well.

Reference cells are not primitive in **AC**, but they can be implemented using coroutines. See the following encoding (this encoding can be extended into a macro-translation from the calculus for reference cells to **AC**; see Section 5):

$$\begin{aligned}
\mathit{ref} &\stackrel{\text{def}}{=} \{\lambda v.\ \mathbf{create}\, \mathit{RefCell}(v)\} \\
\mathit{RefCell}(v) &\stackrel{\text{def}}{=} \left\{\begin{array}{l}\lambda y.\ \mathbf{let}\ q' = \mathbf{return}\, y\ \mathbf{in} \\ \quad\ \mathit{loop}!\, v\, q'\end{array}\right\} \\
\mathit{loop} &\stackrel{\text{def}}{=} \{(\lambda x.\ \mathit{th}!\, \{x!\, x\})\, \{\lambda x.\ \mathit{th}!\, \{x!\, x\}\}\} \\
\mathit{th} &\stackrel{\text{def}}{=} \left\{\begin{array}{l}\lambda f.\lambda s.\lambda q.\ \mathbf{case}\ q\ \mathbf{of}\ \{ \\ \quad (\mathbf{inj}_{\mathrm{Set}}\, v) \mapsto \mathbf{let}\ q' = \mathbf{yield}\, ()\ \mathbf{in}\, f!\, v\, q' \\ \quad (\mathbf{inj}_{\mathrm{Get}}\, _) \mapsto \mathbf{let}\ q' = \mathbf{yield}\, s\ \mathbf{in}\, f!\, s\, q'\}\end{array}\right\} \\
\mathit{get} &\stackrel{\text{def}}{=} \left\{\lambda c.\ \mathbf{resume}\, c\, (\mathbf{inj}_{\mathrm{Get}}\, ())\right\} \\
\mathit{set} &\stackrel{\text{def}}{=} \left\{\lambda c.\ \lambda v.\ \mathbf{resume}\, c\, (\mathbf{inj}_{\mathrm{Set}}\, v)\right\}
\end{aligned}$$

Given a value v, *ref* creates a coroutine $\mathit{RefCell}(v)$ that holds v as its content and runs an infinite loop waiting for a query q. When it is passed a query of the form $\mathbf{inj}_{\mathrm{Get}}\, ()$, it returns the current state, continuing the loop; when passed a query of the form $\mathbf{inj}_{\mathrm{Set}}\, v$, it updates the state to v, returning the unit value $()$.

Using reference cells, we give the refined translation in Figure 10. In the translation of a dollar term, we introduce a thunk zc that creates a reference cell holding the coroutine z tagged with Valid and pass it in place of the coroutine in the naïve translation. This thunk is forced in the body of the translation of a shift0 term to obtain the reference cell. In the translation of a throw term $\underline{\mathbf{throw}\, V\, W}$, we pattern-match on the content of the reference cell labeled as $\underline{V}$. If its content is $\mathbf{inj}_{\mathrm{Valid}}\, m$, then we update the content of the reference cell to $\mathbf{inj}_{\mathrm{Invalid}}\, ()$, which makes every later use of $\underline{V}$ fail; we then resume the coroutine labeled m with $\underline{W}$, passing it a thunk zc' built in the same way as zc. Otherwise, we let the computation fail.

We address the first gap by building zc in the dollar term and zc' in the throw term, each of which creates a new reference cell when forced. Thus, the continuations captured by successive shift0 invocations are represented by distinct reference cells, even though they all refer to the same coroutine z. We address the second gap by never turning the content of a reference cell from $\mathbf{inj}_{\mathrm{Invalid}}\, ()$ back to $\mathbf{inj}_{\mathrm{Valid}}\, m$. Thus, once a reference cell drops the Valid tag, it never regains it, whereas the coroutine m keeps alternating between valid and invalid.

$$\underline{\mathbf{S_0} k.\ M} \stackrel{\text{def}}{=} \textbf{yield}\,\{\lambda x.\ \textbf{let}\ k = x!\ \textbf{in}\ \underline{M}\}$$

$$\underline{\langle M \mid x.N\rangle} \stackrel{\text{def}}{=}$$

$$\left(\begin{array}{l}\textbf{let}\ z = \textbf{create}\\ \quad \left\{\begin{array}{l}\lambda\_.\ \textbf{let}\ x = \underline{M}\ \textbf{in}\\ \qquad \textbf{return}\,\{\lambda\_.\ \underline{N}\}\end{array}\right\}\ \textbf{in}\\ \textbf{let}\ zc = \textbf{return}\,\{\mathit{ref}!\ (\textbf{inj}_{\text{Valid}}\ z)\}\ \textbf{in}\\ \textbf{let}\ \mathit{res} = \textbf{resume}\ z\ ()\ \textbf{in}\\ \mathit{res}!\ zc\end{array}\right)$$

$$\underline{\textbf{throw}\ V\ W} \stackrel{\text{def}}{=}$$

$$\left(\begin{array}{l}\textbf{let}\ zc = \mathit{get}!\ \underline{V}\ \textbf{in}\\ \textbf{case}\ zc\ \textbf{of}\ \{\\ \quad (\textbf{inj}_{\text{Valid}}\ z) \mapsto\\ \qquad \textbf{let}\ \_ = \mathit{set}!\ \underline{V}\ (\textbf{inj}_{\text{Invalid}}\ ())\ \textbf{in}\\ \qquad \textbf{let}\ zc' = \textbf{return}\,\{\mathit{ref}!\ (\textbf{inj}_{\text{Valid}}\ z)\}\ \textbf{in}\\ \qquad \textbf{let}\ \mathit{res} = \textbf{resume}\ z\ \underline{W}\ \textbf{in}\\ \qquad \mathit{res}!\ zc'\\ \quad (\textbf{inj}_{\text{Invalid}}\ \_) \mapsto \mathit{fail}!\\ \}\end{array}\right)$$

where

$$\mathit{fail} \stackrel{\text{def}}{=}$$

$$\left\{\begin{array}{l}\textbf{let}\ z = \textbf{create}\,\{\lambda\_.\ \textbf{return}\ ()\}\ \textbf{in}\\ \textbf{let}\ \_ = \textbf{resume}\ z\ ()\ \textbf{in}\\ \textbf{resume}\ z\ ()\end{array}\right\}$$

$$\mathit{ref} \stackrel{\text{def}}{=} \{\lambda v.\ \textbf{create}\ \mathit{RefCell}(v)\}$$

$$\mathit{RefCell}(v) \stackrel{\text{def}}{=}$$

$$\left\{\begin{array}{l}\lambda y.\ \textbf{let}\ q' = \textbf{return}\ y\ \textbf{in}\\ \qquad \mathit{loop}!\ v\ q'\end{array}\right\}$$

$$\mathit{loop} \stackrel{\text{def}}{=} \{(\lambda x.\ \mathit{th}!\ \{x!\ x\})\ \{\lambda x.\ \mathit{th}!\ \{x!\ x\}\}\}$$

$$\mathit{th} \stackrel{\text{def}}{=}$$

$$\left\{\begin{array}{l}\lambda f.\lambda s.\lambda q.\ \textbf{case}\ q\ \textbf{of}\ \{\\ \quad (\textbf{inj}_{\text{Set}}\ v) \mapsto\\ \qquad \textbf{let}\ q' = \textbf{yield}\ ()\ \textbf{in}\ f!\ v\ q'\\ \quad (\textbf{inj}_{\text{Get}}\ \_) \mapsto\\ \qquad \textbf{let}\ q' = \textbf{yield}\ s\ \textbf{in}\ f!\ s\ q'\}\end{array}\right\}$$

$$\mathit{get} \stackrel{\text{def}}{=} \{\lambda c.\ \textbf{resume}\ c\ (\textbf{inj}_{\text{Get}}\ ())\}$$

$$\mathit{set} \stackrel{\text{def}}{=} \{\lambda c.\ \lambda v.\ \textbf{resume}\ c\ (\textbf{inj}_{\text{Set}}\ v)\}$$

Fig. 10. Refined translation from $\mathbf{DEL_{one}}$ to $\mathbf{AC}$

Now, the evaluation of M in Section 3.3 under the refined translation proceeds as follows:

$$\underline{M} \equiv \left(\begin{array}{l}\textbf{let}\ z = \textbf{create}\,\{\lambda\_.\ \mathcal{P}[\underline{M'}]\}\ \textbf{in}\\ \textbf{let}\ \mathit{res} = \textbf{resume}\ z\ ()\ \textbf{in}\\ \textbf{let}\ zc = \textbf{return}\,\{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\ z\}\ \textbf{in}\\ \mathit{res}!\ zc\end{array}\right)$$

$$\longrightarrow^{+}_{\mathbf{AC}} \left(\begin{array}{l}\textbf{let}\ \mathit{res} =\\ \qquad m : \mathcal{P}\left[\begin{array}{l}\textbf{let}\ j =\\ \qquad \textbf{yield}\left\{\lambda x_1.\left(\begin{array}{l}\textbf{let}\ k_1 = x_1!\ \textbf{in}\\ \textbf{let}\ r_1 = \underline{\textbf{throw}}\ k_1\ 10\ \textbf{in}\\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\ k_1\ 20\ \textbf{in}\\ \textbf{return}\ r_1\end{array}\right)\right\}\ \textbf{in}\\ \textbf{yield}\ \{\lambda x_2.\ \textbf{let}\ k_2 = x_2!\ \textbf{in}\ \textbf{return}\ 30\}\end{array}\right]\ \textbf{in}\\ \mathit{res}!\ \{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\ m\}\end{array}\right)$$

[where $m \mapsto \textbf{nil}$]

$$\longrightarrow^{+}_{\mathbf{AC}} \left\{\lambda x_1.\left(\begin{array}{l}\textbf{let}\ k_1 = x_1!\ \textbf{in}\\ \textbf{let}\ r_1 = \underline{\textbf{throw}}\ k_1\ 10\ \textbf{in}\\ \textbf{let}\ r_2 = \underline{\textbf{throw}}\ k_1\ 20\ \textbf{in}\\ \textbf{return}\ r_1\end{array}\right)\right\}!\ \{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\ m\}$$

$$\left[\begin{array}{l}\text{where}\\ m \mapsto \left\{\lambda y.\ \mathcal{P}\left[\begin{array}{l}\textbf{let}\ j = \textbf{return}\ y\ \textbf{in}\\ \textbf{yield}\ \{\lambda x_2.\ \textbf{let}\ k_2 = x_2!\ \textbf{in}\ \textbf{return}\ 30\}\end{array}\right]\right\}\end{array}\right]$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = \underline{\mathbf{throw}}\ m_1\ 10\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ r_1 \end{pmatrix} \tag{6}$$

$$\left[\text{where } m_1 \mapsto \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\ m)\right]$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = \begin{pmatrix} \mathbf{let}\ \mathit{res} = m : \left(\mathcal{P}\begin{bmatrix} \mathbf{let}\ j = \mathbf{return}\ 10\ \mathbf{in} \\ \mathbf{yield}\ \{\lambda x_2.\ \mathbf{let}\ k_2 = x_2!\ \mathbf{in}\ \mathbf{return}\ 30\} \end{bmatrix}\right)\ \mathbf{in} \\ \mathit{res}!\ \left\{\mathit{ref}!\ \mathbf{inj}_{\mathrm{Valid}}\ m\right\} \end{pmatrix}\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ r_1 \end{pmatrix}$$

$$\left[\text{where } m \mapsto \mathbf{nil}, m_1 \mapsto \mathit{RefCell}(\mathbf{inj}_{\mathrm{Invalid}}\ ())\right]$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = \{\lambda x_2.\ \mathbf{let}\ k_2 = x_2!\ \mathbf{in}\ \mathbf{return}\ 30\}!\ \left\{\mathit{ref}!\ \mathbf{inj}_{\mathrm{Valid}}\ m\right\}\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ r_1 \end{pmatrix}$$

$$[\text{where } m \mapsto \{\lambda y.\ \mathcal{P}[\mathbf{return}\ y]\}]$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = \left(\mathbf{let}\ k_2 = \left\{\mathit{ref}!\ \mathbf{inj}_{\mathrm{Valid}}\ m\right\}!\ \mathbf{in}\ \mathbf{return}\ 30\right)\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ 30 \end{pmatrix}$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = (\mathbf{let}\ k_2 = \mathbf{return}\ m_2\ \mathbf{in}\ \mathbf{return}\ 30)\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ 30 \end{pmatrix}$$

$$\left[\text{where } m_2 \mapsto \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\ m)\right]$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_1 = \mathbf{return}\ 30\ \mathbf{in} \\ \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ 30 \end{pmatrix}$$

$$\rightarrow^{+}_{\mathbf{AC}} \begin{pmatrix} \mathbf{let}\ r_2 = \underline{\mathbf{throw}}\ m_1\ 20\ \mathbf{in} \\ \mathbf{return}\ 30 \end{pmatrix} \tag{7}$$

$$\rightarrow^{+}_{\mathbf{AC}} \perp$$

Thus, the evaluation of $\underline{M}$ results in the error state $\perp$, as it should. There are two points worth noting. First, the two continuations captured in $\underline{M}$ are represented by distinct cells m_1 and m_2, even though both are backed by the same coroutine m. Therefore, the capture of m_2 no longer reactivates m_1, as it did under the naïve translation. Second, the invocation of the first $\underline{\mathbf{throw}}$ in (6) makes the cell m_1 hold $\mathbf{inj}_{\mathrm{Invalid}}\ ()$, and nothing restores it afterwards, so the second throw fails in (7).

3.5 Simulation relation

To prove the correctness of the translation in Figure 10, we use *simulation*. For two transition systems $\mathscr{L}$ and $\mathscr{L}'$, a binary relation on these systems is a simulation relation if the following condition holds:

> If $M \rightarrow_{\mathscr{L}} M'$ and $M \sim N$ hold, then there exists an N' such that $N \rightarrow^{*}_{\mathscr{L}'} N'$ 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 $\mathbf{DEL}_{\text{one}}$ configurations and **AC** configurations that respects the translation in Figure 10. We then use this to show that the translation preserves the semantics.

Constructing such a relation is the main obstacle to the correctness proof. In the refined translation, the validity of a continuation is recorded in the corresponding reference cell, not in the coroutine that realizes it. Therefore, a target term alone does not tell us which source configuration it represents: our simulation relation must relate stores as well as computations.

We do so with *label maps*, which associate a continuation label (e.g., l_1 in our running example) with a reference cell that represents it (e.g., m_1) and a coroutine label that realizes it (e.g., m). We also enforce some *invariant conditions*: several continuation labels may share one coroutine (e.g., l_1 and l_2 share m), but at most one of them is valid, and each reference cell's content follows the validity of its label. Setting up reference cells and coroutines in **AC** needs several steps that have no counterpart in $\mathbf{DEL}_{\text{one}}$. We therefore let the *core* relation relate a source configuration to a *canonical* target representative, and close it under these *administrative* reductions.

Label maps and invariant conditions. First, we define the *label maps*: A label map is a partial function $\eta : \mathscr{L}_{\mathbf{D}} \rightharpoonup \mathscr{L}_{\mathbf{AC}} \times \mathscr{L}_{\mathbf{AC}}$. If $\eta(l) = (mc, m)$, then mc is the reference cell that represents l, and m is the coroutine label that mc may hold as $\mathbf{inj}_{\text{Valid}}\, m$. We then parameterize and extend the translation $\underline{\cdot}$ with η, by translating the source continuation label l to the label mc of the reference cell. We call this extension a *runtime translation* and write it as $\underline{\cdot}_{\eta}$. Similarly, we extend runtime translations to translations on contexts by mapping a hole to a hole.

Using label maps and the extended translation, we define coherence conditions, which constrain a label map against a source store, and invariant conditions, which relate a source store to a target one.

Definition 3.3 (coherence conditions). For a $\mathbf{DEL}_{\text{one}}$ store θ and a label map η, we define $\mathsf{Coh}(\theta, \eta)$ to hold if and only if both of the following conditions are satisfied:

(C1) $\mathrm{Dom}(\eta) = \mathrm{Dom}(\theta)$;

(C2) η is injective in its first component: if $\eta(l) = (mc, m)$ and $\eta(l') = (mc, m')$, then $l = l'$.

Definition 3.4 (invariant conditions). For a $\mathbf{DEL}_{\text{one}}$ store θ, an **AC** store τ, and a runtime label map η, we define $\mathsf{Inv}(\theta, \tau, \eta)$ to hold if and only if the following conditions are satisfied:

(IC1) If $\theta(l) = \mathbf{nil}$ and $\eta(l) = (mc, m)$, then $\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\text{Invalid}}\, ())$;

(IC2) If $\theta(l) = \lambda y.\, \langle \mathcal{H}[\mathbf{return}\, y] | x.M \rangle$ and $\eta(l) = (mc, m)$, then

  (a) $\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\text{Valid}}\, m)$,

  (b) $\tau(m) = \mathrm{Cont}_{\eta}(H, x, M)$, and

  (c) for any $l' \in \mathrm{Dom}(\theta) \setminus \{l\}$ and $mc' \in \mathscr{L}_{\mathbf{AC}}$, if $\eta(l') = (mc', m)$, then $\theta(l') = \mathbf{nil}$,

where

$$\mathrm{Cont}_{\eta}(H, x, M) \stackrel{\text{def}}{=} \left\{\lambda y.\, \mathbf{let}\ x = \underline{\mathcal{H}[\mathbf{return}\, y]}_{\eta}\ \mathbf{in\ return} \left\{\lambda\_.\ \underline{M}_{\eta}\right\}\right\}.$$

Condition (IC1) ensures that a reference cell representing an invalid continuation has $\mathbf{inj}_{\text{Invalid}}\, ()$. Condition (IC2) states that a valid continuation corresponds to a valid reference cell containing a coroutine label that realizes the continuation. Moreover, condition (IC2-c) says that a coroutine label m realizes only one valid source continuation label. In other words, since the second component of η is not necessarily injective, the label m may have more than one corresponding source continuations labels, but condition (IC2-c) ensures that only one of them is valid.

Core relation. We then define the *core relation* that specifies the correspondence between source and target configurations.

Definition 3.5 (core relation). For a label map η, we define a relation $\overset{\eta}{\sim}$ on $\mathbf{DEL}_{\text{one}}$ configurations and **AC** configurations inductively on the structure of $\mathbf{DEL}_{\text{one}}$ computations as follows.

For every constructor E of $\mathbf{DEL}_{\text{one}}$ among

$$\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M, \qquad \mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\ x_i \mapsto m_i)_i\}, \qquad V!, \qquad \mathbf{return}\ V,$$

$$\lambda x.\ M, \qquad \langle M_1, M_2 \rangle, \qquad \mathbf{S_0} k.\ M, \qquad \mathbf{throw}\ V\ W,$$

we have the base clause

$$\langle E; \theta \rangle \overset{\eta}{\sim} \left\langle \underline{E}_\eta; \tau \right\rangle,$$

for any θ and τ.

For other constructs of $\mathbf{DEL}_{\text{one}}$, we have

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

$$\frac{\langle M; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle}{\langle M\, V; \theta \rangle \overset{\eta}{\sim} \left\langle N\, \underline{V}_\eta; \tau \right\rangle}, \qquad \frac{\langle M; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle}{\left\langle \mathbf{prj}_i\ M; \theta \right\rangle \overset{\eta}{\sim} \left\langle \mathbf{prj}_i\ N; \tau \right\rangle},$$

and

$$\frac{\langle M_1; \theta \rangle \overset{\eta}{\sim} \langle N_1; \tau \rangle \qquad \tau(m) = \mathbf{nil} \qquad \forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \mathbf{nil}}{\langle \langle M_1 \,|\, x.M_2 \rangle; \theta \rangle \overset{\eta}{\sim} \left\langle \mathrm{Act}_\eta(N_1, x, M_2, m); \tau \right\rangle},$$

where

$$\mathrm{Act}_\eta(N, x, M, m) \overset{\text{def}}{=} \left(\begin{array}{l} \mathbf{let}\ res = m : \left(\mathbf{let}\ x = N\ \mathbf{in}\ \mathbf{return}\ \left\{\lambda\_.\ \underline{M}_\eta\right\}\right)\ \mathbf{in} \\ res!\ \left\{ref!\ (\mathbf{inj}_{\mathrm{Valid}}\ m)\right\} \end{array} \right).$$

We defined the core relation on configurations, not on computations. This is because, in the clause for the dollar term, the target configuration has an active coroutine m in it, which requires two additional conditions in the premise. These premises are complementary to the invariant conditions. The invariant conditions are concerned with continuations to be invoked—thus suspended coroutines. On the other hand, these two premises are concerned with the current continuation to be captured—thus the active coroutine m.

We extend the core relation onto contexts: $\langle C; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}; \tau \rangle$, where C is a $\mathbf{DEL}_{\text{one}}$ evaluation context, $\mathcal{D}$ is an **AC** evaluation context, θ is a $\mathbf{DEL}_{\text{one}}$ store, and τ is an **AC** store. The definition of $\langle C; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}; \tau \rangle$ has clauses similar to the last four in the definition of $\overset{\eta}{\sim}$, plus a clause for a hole: $\langle [\ ]; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle [\ ]; \tau \rangle$.

Then, we put several side conditions together into a single relation:

Definition 3.6. For $\mathbf{DEL}_{\text{one}}$ and **AC** configurations $C = \langle M; \theta \rangle$ and $D = \langle N; \tau \rangle$, we write $C \overset{\eta}{\sim}_\sharp D$ if and only if all of the following hold:

(1) $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(C)$, which means that all continuation labels occurring in M or in θ are in $\mathrm{Dom}(\theta)$;[11]
(2) $\mathsf{WF}_{\mathbf{AC}}(D)$, which is defined similarly, and which moreover requires that $\tau(m) = \mathbf{nil}$ for every coroutine m running in N, that is, for every m such that N has a subterm of the form $m : N'$;
(3) $\mathsf{Coh}(\theta, \eta)$;
(4) $\mathsf{Inv}(\theta, \tau, \eta)$;
(5) $C \overset{\eta}{\sim} D$.

There are two points worth noting about this definition. First, $C \overset{\eta}{\sim}_\sharp D$ is parameterized by η, which is necessary to record the correspondence between source continuations and target coroutines that are dynamically generated during evaluation. The final simulation relation will be given by existentially quantifying η. Second, the relation mentions only *canonical* **AC** configuration D. For example, the base clause for the dollar term, $\langle\langle M_1 | x.M_2\rangle; \theta\rangle \overset{\eta}{\sim} \left\langle \underline{\langle M_1 | x.M_2\rangle}_\eta ; \tau\right\rangle$, is not included. This is because, by reducing the configuration on the right-hand side, we obtain

$$\left\langle \underline{\langle M_1 | x.M_2\rangle}_\eta ; \tau\right\rangle \to^+_{\mathbf{AC}} \left\langle \mathrm{Act}_\eta(N_1, x, M_2, m); \tau'\right\rangle,$$

for some m and τ', and then we can prove that

$$\langle\langle M_1 | x.M_2\rangle; \theta\rangle \overset{\eta}{\sim} \left\langle Act_\eta(N_1, x, M_2, m); \tau'\right\rangle.$$

Put differently, we consider the target configurations *modulo administrative reductions*, which leads to the following definition of the simulation relation:

Definition 3.7 (simulation relation). For $\mathbf{DEL}_{\mathrm{one}}$ and **AC** configurations C and D, define $C \sim D$ if and only if either $C = D = \bot$, or there exist a label map η and an **AC** configuration D' such that

$$D \to^*_{\mathbf{AC}} D' \quad \text{and} \quad C \overset{\eta}{\sim}_\sharp D'.$$

We remark that $C \sim D$ has the following properties, the latter of which ensures that we can *canonicalize* the target configuration.

Lemma 3.8. *(1) For every $\mathbf{DEL}_{\mathrm{one}}$ program M, $\langle M; \emptyset\rangle \sim \left\langle \underline{M}; \emptyset\right\rangle$ holds.*
*(2) Assume $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M; \theta\rangle)$, $\mathsf{WF}_{\mathbf{AC}}(\left\langle \underline{M}_\eta; \tau\right\rangle)$, $\mathsf{Coh}(\theta, \eta)$, and $\mathsf{Inv}(\theta, \tau, \eta)$. Then, there exist an **AC** computation N and an **AC** store τ' such that $\left\langle \underline{M}_\eta; \tau\right\rangle \to^*_{\mathbf{AC}} \langle N; \tau'\rangle$ and $\langle M; \theta\rangle \overset{\eta}{\sim}_\sharp \langle N; \tau'\rangle$.*

3.6 Correctness of the refined translation

We prove that our refined translation and the simulation relation form a simulation in three steps. First, we prove the simulation of beta-reduction (Proposition 3.9), then its lift onto the general reduction (Proposition 3.10), and finally the simulation of the general reduction (Theorem 3.11). The preservation of semantics follows from it as a corollary. We shall give all the key propositions and proof sketches for important cases. The full proof appears in Appendix A.2.

Proposition 3.9. *Assume $\langle M; \theta\rangle \overset{\eta}{\sim}_\sharp \langle N; \tau\rangle$. Then:*

[11] See Appendix A.1.1 for the precise definition. We remark that well-formedness is preserved by reduction.

(1) If $\langle M;\theta\rangle \to_{\mathbf{D}}^{\beta} \bot$*, then*

$$\langle N;\tau\rangle \to_{\mathbf{AC}}^{+} \bot.$$

(2) If $\langle M;\theta\rangle \to_{\mathbf{D}}^{\beta} \langle M';\theta'\rangle$ *then there exist* N'*,* τ'*,* η' *such that*

$$\langle N;\tau\rangle \to_{\mathbf{AC}}^{+} \langle N';\tau'\rangle \quad \textit{and} \quad \langle M';\theta'\rangle \overset{\eta'}{\sim}_{\sharp} \langle N';\tau'\rangle.$$

Proof. By case analysis on the beta-reduction rule used in the reduction of $\langle M;\theta\rangle$. We write out the cases for (throw) and (fail).

Case (throw):

Suppose $M = \mathbf{throw}\, l\, V$ and $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\, y] | x.M_0\rangle$. Then

$$\langle M;\theta\rangle \to_{\mathbf{D}} \langle\langle \mathcal{H}[\mathbf{return}\, V] | x.M_0\rangle;\theta'\rangle, \qquad \theta' \overset{\text{def}}{=} \theta[l := \mathbf{nil}].$$

The last rule used to derive $\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle$ is the base clause for **throw**. Hence $N \equiv \underline{\mathbf{throw}\, l\, V}_{\eta}$. By (C1), there are target labels mc and m such that $\eta(l) = (mc, m)$. By (IC2),

$$\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\text{Valid}}\, m), \qquad \tau(m) = \text{Cont}_{\eta}(H, x, M_0),$$

and for any continuation label l' and coroutine label mc', if $l' \neq l$ and $\eta(l') = (mc', m)$, then $\theta(l') = \mathbf{nil}$. Put

$$\tau' \overset{\text{def}}{=} \tau[mc := \mathit{RefCell}(\mathbf{inj}_{\text{Invalid}}\, ()),\ m := \mathbf{nil}].$$

By calculation, we have

$$\left\langle \underline{\mathbf{throw}\, l\, V}_{\eta};\tau\right\rangle \to_{\mathbf{AC}}^{+} \left\langle \text{Act}_{\eta}(\underline{\mathcal{H}[\mathbf{return}\, V]}_{\eta}, x, M_0, m);\tau'\right\rangle,$$

and $\underline{\langle \mathcal{H}[\mathbf{return}\, V] | x.M_0\rangle}_{\eta} \equiv \text{Act}_{\eta}(\underline{\mathcal{H}[\mathbf{return}\, V]}_{\eta}, x, M_0, m)$.

We next check the hypotheses of Lemma 3.8(2) for $\mathcal{H}[\mathbf{return}\, V]$ as M with θ', τ' and η. $\mathsf{WF}_{\mathbf{DEL}_{\text{one}}}(\ldots)$ and $\mathsf{WF}_{\mathbf{AC}}(\ldots)$ are immediate. $\mathsf{Coh}(\theta',\eta)$ follows from $\mathsf{Coh}(\theta,\eta)$ because $\text{Dom}(\theta') = \text{Dom}(\theta)$. To prove $\mathsf{Inv}(\theta',\tau',\eta)$, we show (IC1) as follows: The label l is invalid in θ' and mc has the invalid tag in τ'. All other invalid continuation labels in θ' are invalid in θ, and mapped to invalid cells by η and (IC1) for θ. Thus (IC1) holds. We then show (IC2) as follows: If l' is valid in θ', then $l' \neq l$, and by (C2) of $\mathsf{Coh}(\theta',\eta)$, the first component of $\eta(l')$ is not mc. Moreover, the coroutine realizing l' cannot be m: if so, (IC2-c) of $\mathsf{Inv}(\theta,\tau,\eta)$ would imply $\theta(l') = \mathbf{nil}$. Thus the store contents relevant to l' are unchanged, and (IC2) follows.

Hence, we can apply Lemma 3.8(2) for $M = \mathcal{H}[\mathbf{return}\, V]$ with θ', τ', and η, to obtain

$$\left\langle \underline{\mathcal{H}[\mathbf{return}\, V]}_{\eta};\tau'\right\rangle \to_{\mathbf{AC}}^{*} \langle N';\tau''\rangle \quad \text{and} \quad \langle \mathcal{H}[\mathbf{return}\, V];\theta'\rangle \overset{\eta}{\sim}_{\sharp} \langle N';\tau''\rangle,$$

for some **AC** computation N' and **AC** store τ''. This reduction takes place in the body of the active coroutine m, so it lifts to

$$\left\langle \text{Act}_{\eta}(\underline{\mathcal{H}[\mathbf{return}\, V]}_{\eta}, x, M_0, m);\tau'\right\rangle \to_{\mathbf{AC}}^{*} \left\langle \text{Act}_{\eta}(N', x, M_0, m);\tau''\right\rangle,$$

and hence $\langle N;\tau\rangle \to_{\mathbf{AC}}^{+} \left\langle \text{Act}_{\eta}(N', x, M_0, m);\tau''\right\rangle$.

It remains to show $\langle\langle \mathcal{H}[\mathbf{return}\, V] | x.M_0\rangle;\theta'\rangle \overset{\eta}{\sim}_{\sharp} \left\langle \text{Act}_{\eta}(N', x, M_0, m);\tau''\right\rangle$. The two well-formedness conditions, $\mathsf{Coh}(\theta',\eta)$, and $\mathsf{Inv}(\theta',\tau'',\eta)$ are immediately derived from the relation $\overset{\eta}{\sim}_{\sharp}$ obtained above. For the core relation, we apply the dollar-term rule of the core relation to $\langle \mathcal{H}[\mathbf{return}\, V]; \theta' \rangle \overset{\eta}{\sim} \langle N'; \tau'' \rangle$, so it suffices to check its two side conditions

$$\tau''(m) = \mathbf{nil} \quad \text{and} \quad \forall l', mc'.\, \eta(l') = (mc', m) \Rightarrow \theta'(l') = \mathbf{nil}.$$

The former holds because the coroutine m keeps running throughout the lifted reduction above, and $\mathsf{WF}_{\mathbf{AC}}$, which is preserved by reduction, forces $\tau''(m)$ to be **nil** (see the proof of Proposition A.26 in the appendix for the details). As for the latter, (IC2-c) of $\mathsf{Inv}(\theta, \tau, \eta)$ tells us that the label l is the only valid continuation label defined in θ that is associated with the coroutine m. This label is no longer valid in θ', which implies the condition above.

Case (fail):

Suppose $M = \mathbf{throw}\, l\, V$ and $\theta(l) = \mathbf{nil}$. Then $\langle M; \theta \rangle \to_{\mathbf{D}}^{\beta} \bot$. The last rule used to derive $\langle M; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle$ is the base clause for **throw**, so $N \equiv \underline{\mathbf{throw}\, l\, V}_{\eta}$. By (C1), $\eta(l) = (mc, m)$ is defined. Since $\theta(l) = \mathbf{nil}$, (IC1) gives

$$\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\mathrm{Invalid}}\, ()).$$

Therefore, by calculation, we obtain

$$\left\langle \underline{\mathbf{throw}\, l\, V}_{\eta}; \tau \right\rangle \to_{\mathbf{AC}}^{+} \langle \mathit{fail}!; \tau \rangle \to_{\mathbf{AC}}^{+} \bot.$$

This concludes the proof. □

Proposition 3.10. *Suppose* $\langle M; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle$, $\langle C; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}; \tau \rangle$, $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle C[M]; \theta \rangle)$, *and* $\mathsf{WF}_{\mathbf{AC}}(\langle \mathcal{D}[N]; \tau \rangle)$.

(1) If $\langle M; \theta \rangle \to_{\mathbf{D}}^{\beta} \bot$, *then*

$$\langle \mathcal{D}[N]; \tau \rangle \to_{\mathbf{AC}}^{+} \bot.$$

(2) If $\langle M; \theta \rangle \to_{\mathbf{D}}^{\beta} \langle M'; \theta' \rangle$ *then there exist* N', τ', η' *such that*

$$\langle N; \tau \rangle \to_{\mathbf{AC}}^{+} \langle N'; \tau' \rangle, \qquad \langle C[M']; \theta' \rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}[N']; \tau' \rangle, \qquad \langle C; \theta' \rangle \overset{\eta'}{\sim}_{\mathsf{c}} \langle \mathcal{D}; \tau' \rangle.$$

Proof. By induction on the derivation of $\langle C; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}; \tau \rangle$. The base case is Proposition 3.9. For the inductive step, the cases for **MAM** frames trivially hold. The only nontrivial case is that for a dollar frame, where we have to carefully check that the side conditions involving an active coroutine are preserved through reduction (See the proof of Proposition A.27 in the appendix). □

Theorem 3.11 (simulation). *If* $C \sim D$ *and* $C \to_{\mathbf{D}} C'$, *then there exists an* **AC** *configuration* D' *such that*

$$D \to_{\mathbf{AC}}^{+} D' \quad \text{and} \quad C' \sim D'.$$

Proof. Since $C \to_{\mathbf{D}} C'$, the configuration C is not $\bot$. By the definition of $\sim$, there exist an **AC** computation P, an **AC** store τ, and a label map η such that

$$D \to_{\mathbf{AC}}^{*} \langle P; \tau \rangle \quad \text{and} \quad C \overset{\eta}{\sim}_{\sharp} \langle P; \tau \rangle.$$

Write $C = \langle M_0; \theta \rangle$. By the definition of $\to_{\mathbf{D}}$, the source step decomposes into an evaluation context and a beta-reduction step. Thus, there exist a $\mathbf{DEL}_{\mathrm{one}}$ evaluation context C_0 and a $\mathbf{DEL}_{\mathrm{one}}$ computation M such that $M_0 \equiv C_0[M]$ and either

$$\langle M; \theta \rangle \to_{\mathbf{D}}^{\beta} \bot \quad \text{and} \quad C' = \bot,$$

or there exist a $\mathbf{DEL}_{\mathrm{one}}$ computation M' and a $\mathbf{DEL}_{\mathrm{one}}$ store θ' such that

$$\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} \langle M'; \theta' \rangle \quad \text{and} \quad C' = \langle C_0[M']; \theta' \rangle.$$

From $\langle C_0[M]; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle P; \tau \rangle$, we obtain in particular $\langle C_0[M]; \theta \rangle \overset{\eta}{\sim} \langle P; \tau \rangle$. By induction on C_0, we can check that there exist an **AC** evaluation context $\mathcal{D}_0$ and an **AC** computation N such that $P \equiv \mathcal{D}_0[N]$, $\langle C_0; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0; \tau \rangle$, and $\langle M; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle$. From the side conditions of $\langle C_0[M]; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle \mathcal{D}_0[N]; \tau \rangle$, we obtain $\langle M; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle$.

We now apply Proposition 3.10 to $\langle M; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle$ and $\langle C_0; \theta \rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0; \tau \rangle$. First, suppose $\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} \bot$. Proposition 3.10 gives $\langle P; \tau \rangle = \langle \mathcal{D}_0[N]; \tau \rangle \to^{+}_{\mathbf{AC}} \bot$. Combining this with $D \to^{*}_{\mathbf{AC}} \langle P; \tau \rangle$ yields $D \to^{+}_{\mathbf{AC}} \bot$. As $C' = \bot$, we have $C' \sim \bot$ by the definition of the simulation relation.

Next suppose $\langle M; \theta \rangle \to^{\beta}_{\mathbf{D}} \langle M'; \theta' \rangle$. Proposition 3.10 yields an **AC** computation N', an **AC** store τ', and a label map η' such that $\langle N; \tau \rangle \to^{+}_{\mathbf{AC}} \langle N'; \tau' \rangle$ and $\langle C_0[M']; \theta' \rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}_0[N']; \tau' \rangle$. By the definition of $\to_{\mathbf{AC}}$, we obtain $\langle \mathcal{D}_0[N]; \tau \rangle \to^{+}_{\mathbf{AC}} \langle \mathcal{D}_0[N']; \tau' \rangle$. Since $P \equiv \mathcal{D}_0[N]$, composing with $D \to^{*}_{\mathbf{AC}} \langle P; \tau \rangle$ gives $D \to^{+}_{\mathbf{AC}} \langle \mathcal{D}_0[N']; \tau' \rangle$. Also, because $C' = \langle C_0[M']; \theta' \rangle$, we obtain $C' \sim \langle \mathcal{D}_0[N']; \tau' \rangle$ by the definition of the simulation relation. □

Using this theorem, we prove the preservation of semantics and the strong macro-expressibility.

Theorem 3.12 (preservation of semantics). *For every* $\mathbf{DEL}_{\mathrm{one}}$ *program* M, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *is defined.*

Proof. We prove the two directions separately.

First suppose that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is defined. Then there exist a $\mathbf{DEL}_{\mathrm{one}}$-value V and a $\mathbf{DEL}_{\mathrm{one}}$-store θ such that $\langle M; \emptyset \rangle \to^{*}_{\mathbf{D}} \langle \mathbf{return}\, V; \theta \rangle$. By Lemma 3.8, $\langle M; \emptyset \rangle \sim \left\langle \underline{M}; \emptyset \right\rangle$. Repeatedly applying Theorem 3.11 along the source reduction sequence, we obtain an **AC**-configuration D such that

$$\left\langle \underline{M}; \emptyset \right\rangle \to^{*}_{\mathbf{AC}} D \quad \text{and} \quad \langle \mathbf{return}\, V; \theta \rangle \sim D.$$

By the definition of $\sim$, there exist an **AC**-computation N, an **AC**-store τ, and a label map η such that

$$D \to^{*}_{\mathbf{AC}} \langle N; \tau \rangle \quad \text{and} \quad \langle \mathbf{return}\, V; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle,$$

the latter of which yields $\langle \mathbf{return}\, V; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle$. By the definition of $\overset{\eta}{\sim}$, N equals $\mathbf{return}\, \underline{V}_{\eta}$. Hence $\left\langle \underline{M}; \emptyset \right\rangle \to^{*}_{\mathbf{AC}} \left\langle \mathbf{return}\, \underline{V}_{\eta}; \tau \right\rangle$, and therefore $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is defined.

We prove the converse direction by contraposition. Assume that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is undefined. Since $\to_{\mathbf{D}}$ is deterministic, exactly one of the following alternatives holds.

(1) The evaluation diverges.
(2) The evaluation reaches the error state $\bot$: $\langle M; \emptyset \rangle \to^{+}_{\mathbf{D}} \bot$.
(3) The evaluation reaches a stuck configuration: there exist a $\mathbf{DEL}_{\mathrm{one}}$-computation M' and a $\mathbf{DEL}_{\mathrm{one}}$-store θ such that $\langle M; \emptyset \rangle \to^{*}_{\mathbf{D}} \langle M'; \theta \rangle$, the configuration $\langle M'; \theta \rangle$ is stuck, and M' is not of the form $\mathbf{return}\, V$.

We can prove that, in each case, the evaluation of $\underline{M}$ diverges, reaches the error state, or gets stuck, respectively (See Appendix A.2.4 for the details). Therefore, in all cases, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is undefined. □

$$
\begin{array}{rcll}
V, W & ::= & \dots & \text{value} \\
 & | & l \quad (\in \mathscr{L}_{\mathbf{E}}) & \text{continuation label} \\
M, N & ::= & \dots & \text{computation} \\
 & | & \mathsf{op}\, V \quad (\mathsf{op} \in \mathscr{O}) & \text{operation call} \\
 & | & \textbf{with } H \textbf{ handle } M & \text{handle} \\
 & | & \textbf{throw}\, V\, W & \text{throw} \\
H & ::= & \{\textbf{return}\ x \mapsto M_{\mathrm{ret}},\ (\mathsf{op}_i\ p_i\ k_i \mapsto M_i)_i\} & \text{handler}^{12}
\end{array}
$$

$$
\begin{array}{rrcl}
\text{pure frame} & \mathcal{P} & ::= & \dots \\
\text{computational frame} & \mathcal{F} & ::= & \dots \mid \textbf{with } H \textbf{ handle } [\ ] \\
\text{pure context} & \mathcal{H} & ::= & \dots \\
\text{evaluation context} & \mathcal{C} & ::= & \dots
\end{array}
$$

Fig. 11. Syntax of $\mathbf{EFF}_{\mathrm{one}}$

THEOREM 3.13 (STRONG MACRO-EXPRESSIBILITY). *The translation $M \mapsto \underline{M}$ of Figure 10 is a strong macro-translation from* $\mathbf{DEL}_{\mathrm{one}}$ *to* $\mathbf{AC}$.

PROOF. The translation maps every $\mathbf{DEL}_{\mathrm{one}}$ program to an $\mathbf{AC}$ program and is homomorphic on the $\mathbf{MAM}$ constructors by definition. Every constructor peculiar to $\mathbf{DEL}_{\mathrm{one}}$ is translated by a fixed syntactic abstraction, as shown in Figure 10. The preservation of semantics is shown by Theorem 3.12. □

4 One-shot effect handlers as asymmetric coroutines

Since Plotkin and Pretnar's proposal [26], effect handlers have been actively studied in recent years. This section extends our results to effect handlers; namely, we establish the macro-expressibility of one-shot effect handlers in terms of asymmetric coroutines. Since macro-expressibility is transitive, 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 11 presents the syntax of $\mathbf{EFF}_{\mathrm{one}}$, which is a one-shot variant of the calculus $\mathbf{EFF}$ for effect handlers defined by Forster et al. [12]. A continuation label l (drawn from a countable set $\mathscr{L}_{\mathbf{E}}$) is a runtime representation of a one-shot continuation, as in $\mathbf{DEL}_{\mathrm{one}}$. The computation $\mathsf{op}\, V$ is an operation invocation with an argument V, where the operation op ranges over a finite set $\mathscr{O}$. A handler $\{\textbf{return}\ x \mapsto M_{\mathrm{ret}},\ (\mathsf{op}_i\ p_i\ k_i \mapsto M_i)_i\}$ consists of the *return clause* $\textbf{return}\, x \mapsto M_{\mathrm{ret}}$, and for each operation symbol op_i, the *operational clause* $\mathsf{op}_i\ p_i\ k_i \mapsto M_i$. The computation **with** H **handle** M installs a handler H and evaluates M under it. The computation **throw** $V\, W$, when evaluated, invokes the continuation associated with V with the argument W if V is a continuation label. We define $\mathbf{EFF}_{\mathrm{one}}$ programs as the set of computations containing no continuation labels.

In our formulation, every handler must handle all operations in $\mathscr{O}$. This is not an essential restriction, as any handler that handles only some of the operations in $\mathscr{O}$ can be rewritten[13] to satisfy this condition.

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

[13]Given such a handler H, we add a clause $\mathsf{op}\ p\ k \mapsto \textbf{let}\ r = \mathsf{op}\ p\ \textbf{in throw}\ k\ r$ to H for each unhandled operation op.

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

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

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

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

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

Fig. 12. Beta reduction rules of $\mathbf{EFF}_{\mathrm{one}}$

$$
\begin{array}{rcl}
\underline{\mathbf{with}\ H\ \mathbf{handle}\ M} & \stackrel{\mathrm{def}}{=} & \left\langle \left\langle \underline{M} \middle| H^{\mathrm{ret}} \right\rangle \{H^{\mathrm{ops}}\} \middle| z.\mathbf{return}\,z \right\rangle \\
& \text{where} & (\{\mathbf{return}\,x \mapsto M_{\mathrm{ret}},\ \ldots\})^{\mathrm{ret}} = x.\lambda\_.\ \underline{M_{\mathrm{ret}}} \\
& & (\{\mathbf{return}\,x \mapsto M_{\mathrm{ret}},\ (\mathtt{op}_i\ p_i\ k_i \mapsto M_i)_i\})^{\mathrm{ops}} \\
& & \quad = \lambda c.\ \mathbf{case}\ c\ \mathbf{of}\ \{(\mathbf{inj}_{\mathtt{op}_i}\ (p_i, k_i) \mapsto \underline{M_i})_i\} \\
\underline{\mathtt{op}\,V} & \stackrel{\mathrm{def}}{=} & \mathbf{S_0}k.\ \lambda h.\ (\mathbf{let}\ y = (\mathbf{S_0}k'.\ h!\,\mathbf{inj}_{\mathtt{op}}\ (\underline{V}, k'))\ \mathbf{in} \\
& & \qquad\qquad \mathbf{throw}\ k\ y\ h) \\
\underline{\mathbf{throw}\,V\,W} & \stackrel{\mathrm{def}}{=} & \mathbf{throw}\ \underline{V}\ \underline{W}
\end{array}
$$

Fig. 13. Translation from $\mathbf{EFF}_{\mathrm{one}}$ to $\mathbf{DEL}_{\mathrm{one}}$

Similarly to $\mathbf{DEL}_{\mathrm{one}}$, captured continuations can be invoked at most once during evaluation. Hence, the evaluation of $\mathbf{with}\ H\ \mathbf{handle}\ (\mathtt{error}\ 10)$ where

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

must result in an error. $\mathbf{EFF}_{\mathrm{one}}$ also uses a store, which a partial function $\theta : \mathscr{L}_{\mathbf{E}} \rightharpoonup \mathsf{computation} \sqcup \{\mathbf{nil}\}$, to enforce one-shotness

A configuration C is either a pair of an $\mathbf{EFF}_{\mathrm{one}}$ 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 12. The reduction of $\mathbf{EFF}_{\mathrm{one}}$ is defined as follows:

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

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

4.2 Macro-translation from $\mathbf{EFF}_{\mathrm{one}}$ to $\mathbf{DEL}_{\mathrm{one}}$

We present a translation from $\mathbf{EFF}_{\mathrm{one}}$ to $\mathbf{DEL}_{\mathrm{one}}$ in Figure 13, which is inspired by Forster et al.'s translation [12]. A handle computation **with** H **handle** M is translated using two nested dollar terms: an outer one that delimits the whole translated computation, and an inner one, whose additional part H^{ret} is built from the return clause of H, that delimits $\underline{M}$. The inner dollar term, once evaluated, is applied to the thunked *dispatcher* $\{H^{\mathrm{ops}}\}$ built from the operational clauses of H. An operation invocation $\mathsf{op}\,V$ is translated to a computation that performs two shift0's in succession. The first captures the continuation up to the inner delimiter, binding k; the resulting abstraction is then applied to the dispatcher, binding h. The second captures the continuation **let** $y = [\ ]$ **in throw** $k\ y\ h$ up to the outer delimiter, binding k', and calls h with the pair, tagged with op, of the argument $\underline{V}$ and k'—the label of this second captured continuation. A source throw term is translated to the target throw term homomorphically. Specifically, when an operational clause invokes its continuation variable, which is bound to k', the target throw term invokes k', which in turn immediately invokes k, reinstalling the dispatcher h.

The use of two successive shift0's is where our translation departs from that of Forster et al. Adapting their translation to $\mathbf{EFF}_{\mathrm{one}}$ constructs, we obtain the following:

$$\begin{aligned}
\underline{\textbf{with}\ H\ \textbf{handle}\ M} &\stackrel{\mathrm{def}}{=} \left\langle \underline{M} \middle| H^{\mathrm{ret}} \right\rangle \{H^{\mathrm{ops}}\}, \\
\underline{\mathsf{op}\,V} &\stackrel{\mathrm{def}}{=} \mathbf{S_0}k.\ \lambda h.\ h!\,(\mathbf{inj}_{\mathsf{op}}\,(\underline{V}, \{\lambda y.\ \textbf{throw}\,k\ y\ h\})), \\
\underline{\textbf{throw}\,V\,W} &\stackrel{\mathrm{def}}{=} \underline{V}!\,\underline{W},
\end{aligned}$$

where a source continuation label is represented by the thunk $\{\lambda y.\ \textbf{throw}\,k\ y\ h\}$, which is a value that can be consumed by the (U) rule of **MAM**. This translation would still be a weak macro-translation, but not a strong one in our setting, since a source program that forces a captured continuation gets stuck in $\mathbf{EFF}_{\mathrm{one}}$ whereas its translation does not.[14] Therefore, we represent an $\mathbf{EFF}_{\mathrm{one}}$ continuation label by a genuine $\mathbf{DEL}_{\mathrm{one}}$ continuation label instead of a thunk. This is the reason why we perform two successive shift0's: the second one captures the computation that Forster et al.'s translation encapsulates as a thunk above.

Simulation relation. As in Section 3.5, we prove the correctness of the translation by constructing a simulation relation between $\mathbf{EFF}_{\mathrm{one}}$ configurations and $\mathbf{DEL}_{\mathrm{one}}$ configurations. While Forster et al. defined a simulation relation using their translation directly, we must take into account labels and stores that are necessary to capture one-shotness. Therefore, our simulation relation must relate the stores as well as the computations, so that it records which captured continuations in $\mathbf{DEL}_{\mathrm{one}}$ correspond to a given $\mathbf{EFF}_{\mathrm{one}}$ captured continuation.

As in Section 3.5, we use *label maps*: here a label map is a partial function $\eta : \mathscr{L}_{\mathbf{E}} \rightharpoonup \mathscr{L}_{\mathbf{D}} \times \mathscr{L}_{\mathbf{D}}$ that sends an $\mathbf{EFF}_{\mathrm{one}}$ continuation label l to a pair $\eta(l) = (\eta(l)_1, \eta(l)_2)$ of $\mathbf{DEL}_{\mathrm{one}}$ continuation labels. Note that, in our translation, an $\mathbf{EFF}_{\mathrm{one}}$ continuation label l is represented by the label passed to the dispatcher, namely the one bound to k'. We thus let $\eta(l)_1$ be that label, generated by the second shift0, and $\eta(l)_2$ be the one bound to k, generated by the first shift0. We then parameterize and extend the translation with η by $\underline{l}_\eta \stackrel{\mathrm{def}}{=} \eta(l)_1$, provided $\eta(l)$

[14]For instance, take $H \equiv \{\textbf{return}\ x \mapsto \textbf{return}\,x,\ \mathsf{op}\,p\ k \mapsto k!\,()\}$. Then, **with** H **handle** $\mathsf{op}\,()$ gets stuck in $\mathbf{EFF}_{\mathrm{one}}$, because the continuation label substituted for k is not a thunk, whereas its translation evaluates to $()$.

is defined. As in Section 3.5, we call this extension a *runtime translation* and write it as $\underline{\cdot}_\eta$. We also extend runtime translations to translations on contexts by mapping a hole to a hole.

Using the label map and the runtime translation, we define *coherence conditions* and *invariant conditions*.

Definition 4.1 (coherence conditions). For an $\mathbf{EFF}_{\text{one}}$ store θ and a label map η, we define $\mathsf{Coh}(\theta, \eta)$ to hold if and only if the following conditions are satisfied.

(C1) $\mathrm{Dom}(\theta) = \mathrm{Dom}(\eta)$;
(C2) for all $l, l' \in \mathrm{Dom}(\theta)$ and $i, j \in \{1, 2\}$, if $\eta(l)_i = \eta(l')_j$, then $l = l'$ and $i = j$.

Definition 4.2 (invariant conditions). For an $\mathbf{EFF}_{\text{one}}$ store θ, a $\mathbf{DEL}_{\text{one}}$ store τ, and a label map η, we define $\mathsf{Inv}(\theta, \tau, \eta)$ to hold if and only if the following conditions are satisfied for every $l \in \mathrm{Dom}(\theta)$.

(IC1) $\eta(l)_1 \in \mathrm{Dom}(\tau)$ and $\eta(l)_2 \in \mathrm{Dom}(\tau)$;
(IC2) if $\theta(l) = \mathbf{nil}$, then $\tau(\eta(l)_1) = \mathbf{nil}$;
(IC3) if $\theta(l) = \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, y]$ for an $\mathbf{EFF}_{\text{one}}$ handler H and an $\mathbf{EFF}_{\text{one}}$ pure context $\mathcal{H}$, then

$$\tau(\eta(l)_1) = \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ \mathbf{throw}\, \eta(l)_2\ y\ \left\{H_\eta^{\text{ops}}\right\} \middle|\, z.\mathbf{return}\, z \right\rangle,$$
$$\tau(\eta(l)_2) = \lambda y.\ \left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_\eta \middle| H_\eta^{\text{ret}} \right\rangle,$$

where H_η^{ops} (resp. H_η^{ret}) denotes the translation of the operational clauses (resp. return clause) of H with respect to η.

Condition (IC2) ensures that an invalid source continuation corresponds to an invalid target continuation, and condition (IC3) states that a valid source continuation is realized by a pair of target continuations.

We can now define the simulation relation.

Definition 4.3 (simulation relation). For an $\mathbf{EFF}_{\text{one}}$ configuration C and a $\mathbf{DEL}_{\text{one}}$ configuration D, we write $C \sim D$ if and only if either $C = D = \bot$, or there exist an $\mathbf{EFF}_{\text{one}}$ computation M, a $\mathbf{DEL}_{\text{one}}$ computation N, an $\mathbf{EFF}_{\text{one}}$ store θ, a $\mathbf{DEL}_{\text{one}}$ store τ, and a label map η such that

(1) $C = \langle M; \theta \rangle$ and $D = \langle N; \tau \rangle$;
(2) $N \equiv \underline{M}_\eta$;
(3) $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(C)$, which means that every continuation label occurring in M or in θ must be in $\mathrm{Dom}(\theta)$;[15]
(4) $\mathsf{Coh}(\theta, \eta)$ and $\mathsf{Inv}(\theta, \tau, \eta)$.

In contrast to Section 3.5, we relate a source computation to its runtime translation directly; in other words, we need not consider administrative reductions. The essential difference is that, in the translation from $\mathbf{DEL}_{\text{one}}$ to $\mathbf{AC}$, a dollar term administratively creates a coroutine, which forces us to relate only *canonical* target representatives, whereas here every $\mathbf{DEL}_{\text{one}}$ continuation label is created by a shift0 term that originates from an operation invocation in the source.

[15] See Appendix B.1.1 for the precise definition. We remark that well-formedness is preserved by reduction.

Correctness of the translation. We shall prove the correctness of the translation in Figure 13. We first prove that $C \sim D$ forms a simulation.

Proposition 4.4 (simulation). *The following statements hold.*

(1) If $C \sim D$ 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 $C' \sim D'$.

(2) (1) can be lifted onto the general reduction: if $C \sim D$ 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. We prove (1) by case analysis on $C \to^{\beta}_{\mathbf{E}} C'$, giving only nontrivial cases. The other cases follow by routine arguments, and (2) follows from (1) together with the fact that an $\mathbf{EFF}_{\mathrm{one}}$ configuration not in normal form decomposes into an evaluation context and a redex, and that this decomposition is preserved by the translation. The details are given in the proofs of Lemma B.9 and Proposition B.12 in the appendix.

Case (op):

Suppose $C = \langle \mathbf{with}\ H\ \mathbf{handle}\ (\mathcal{H}[\mathsf{op}\,V]);\theta\rangle$ and $C \to_{\mathbf{E}} C' = \left\langle M_{\mathsf{op}}[V/p, l/k];\theta'\right\rangle$, where

$$H = \{\mathbf{return}\ x \mapsto M_{\mathrm{ret}},\ \ldots(\mathsf{op}\ p\ k \mapsto M_{\mathsf{op}})\ldots\},$$

$$\theta' \stackrel{\text{def}}{=} \theta[l := \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\,x]],$$

and l is a fresh label. By the definition of $C \sim D$, there exist τ and η such that

$$D = \left\langle \left\langle \left\langle N \middle| H^{\mathrm{ret}}_{\eta} \right\rangle \{H^{\mathrm{ops}}_{\eta}\} \middle| z.\mathbf{return}\,z \right\rangle;\ \tau \right\rangle,$$

where

$$N \equiv \underline{\mathcal{H}}_{\eta}\left[\begin{array}{l}\mathbf{S_0}k.\ \lambda h.\ (\mathbf{let}\ y = (\mathbf{S_0}k'.\ h!\,\mathbf{inj}_{\mathsf{op}}\ (\underline{V}_{\eta}, k'))\ \mathbf{in}\\ \qquad\qquad\quad \mathbf{throw}\ k\ y\ h)\end{array}\right].$$

This configuration evaluates to

$$D' \stackrel{\text{def}}{=} \left\langle \underline{M_{\mathsf{op}}}_{\eta}\left[\underline{V}_{\eta}/p, m/k\right];\tau'\right\rangle$$

where m and n are fresh, and

$$\tau' \stackrel{\text{def}}{=} \tau\left[\begin{array}{l} m := \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\,x\ \mathbf{in}\ \mathbf{throw}\,n\,y\,\{H^{\mathrm{ops}}_{\eta}\} \middle| z.\mathbf{return}\,z\right\rangle,\\ n := \lambda y.\ \left\langle \underline{\mathcal{H}}_{\eta}[\mathbf{return}\,y] \middle| H^{\mathrm{ret}}_{\eta}\right\rangle \end{array}\right].$$

Let η' be $\eta[l := (m,n)]$. By the substitution lemma (Lemma B.8 in the appendix), it follows that

$$\underline{M_{\mathsf{op}}}_{\eta}[\underline{V}_{\eta}/p, m/k] \equiv \underline{M_{\mathsf{op}}[V/p, l/k]}_{\eta'}.$$

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

$$C' \sim D',$$

witnessed by η'. It is straightforward to check that $\mathsf{Coh}(\theta', \eta')$ and $\mathsf{Inv}(\theta', \tau', \eta')$ hold. Here, the freshness of l, m, and n gives (C1) and (C2).

Case (throw):

Suppose $C = \langle \mathbf{throw}\, l\, V; \theta \rangle$ and $C' = \langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, V]; \theta' \rangle$, where $\theta(l) = \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, x]$ and $\theta' \stackrel{\text{def}}{=} \theta[l := \mathbf{nil}]$. By the definition of $C \sim D$, there exist τ and η such that $D = \left\langle \mathbf{throw}\, \eta(l)_1\, \underline{V}_\eta; \tau \right\rangle$,

$$\tau(\eta(l)_1) = \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in\ throw}\, \eta(l)_2\, y\ \left\{H^{\text{ops}}_\eta\right\} \middle|\, z.\mathbf{return}\, z \right\rangle,$$

and

$$\tau(\eta(l)_2) = \lambda y.\ \left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle.$$

Thus, D evaluates to

$$D' = \left\langle \left\langle \left(\left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle \left[\underline{V}_\eta / y\right] \right) \left\{H^{\text{ops}}_\eta\right\} \middle| z.\mathbf{return}\, z \right\rangle; \tau' \right\rangle,$$

where $\tau' \stackrel{\text{def}}{=} \tau[\eta(l)_1 := \mathbf{nil}, \eta(l)_2 := \mathbf{nil}]$. Then, from the substitution lemma, we obtain

$$\left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle [\underline{V}_\eta / y] \equiv \left\langle \underline{\mathcal{H}[\mathbf{return}\, V]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle.$$

Therefore, we conclude that

$$C' \sim D',$$

witnessed by η, since

$$\underline{\mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, V]}_\eta \equiv \left\langle \left\langle \underline{\mathcal{H}[\mathbf{return}\, V]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle \left\{H^{\text{ops}}_\eta\right\} \middle| z.\mathbf{return}\, z \right\rangle.$$

Note that $\mathsf{Coh}(\theta', \eta)$ holds since $\mathrm{Dom}(\theta') = \mathrm{Dom}(\theta)$, and that $\mathsf{Inv}(\theta', \tau', \eta)$ also holds.

Case (fail):

Suppose $C = \langle \mathbf{throw}\, l\, V; \theta \rangle$, $\theta(l) = \mathbf{nil}$, and $C \to^{\beta}_{\mathbf{E}} \perp$. By the definition of $C \sim D$, there exist τ and η such that $D = \left\langle \mathbf{throw}\, \eta(l)_1\, \underline{V}_\eta; \tau \right\rangle$ and $\tau(\eta(l)_1) = \mathbf{nil}$. Hence, the invocation of $\eta(l)_1$ fails and D evaluates to $\perp$, which concludes the current case.

This concludes the proof. □

Proposition 4.5 (preservation of semantics). *For every* $\mathbf{EFF}_{\text{one}}$ *program* M, $\mathrm{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ *is defined.*

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. By induction on M, we obtain $\langle M; \emptyset \rangle \sim \left\langle \underline{M}; \emptyset \right\rangle$. By applying Proposition 4.4 iteratively, there exist τ and η such that $\left\langle \underline{M}; \emptyset \right\rangle \to^*_{\mathbf{D}} \left\langle \mathbf{return}\, \underline{V}_\eta; \tau \right\rangle$. This implies that $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ is defined. We then show the "if" direction. We prove the contrapositive of the proposition: if $\mathrm{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ is undefined, $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ is also undefined. There are three cases where $\mathrm{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ is undefined: the evaluation of M (1) diverges, (2) gets stuck, or (3) reaches $\perp$. We can check that in each case the evaluation of $\underline{M}$ diverges, gets stuck, or reaches $\perp$, respectively (Lemmas B.13, B.14, and B.15 in the appendix). Thus, $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ is undefined, which completes the proof. □

Theorem 4.6. $\mathbf{EFF}_{\text{one}}$ *is macro-expressible in* $\mathbf{DEL}_{\text{one}}$.

Proof. The translation in Figure 13 preserves the semantics by Proposition 4.5, so all we have to do is to check the syntactic conditions. 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 acts homomorphically on them. Also, Figure 13 shows clearly that the program constructs peculiar to $\mathbf{EFF}_{\text{one}}$ are expressed by syntactic abstractions of $\mathbf{DEL}_{\text{one}}$. Therefore, the translation is a macro-translation. □

$V, W ::= \ldots$	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. 14. Syntax of **REF**

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

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

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

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

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

Fig. 15. Semantics of **REF**

From Theorem 4.6 together with Theorem 3.13 and the transitivity of macro-expressibility, we get the following result.

THEOREM 4.7. $\mathbf{EFF}_{\mathrm{one}}$ *is macro-expressible in* **AC**.

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

We can also show the macro-expressibility of $\mathbf{DEL}_{\mathrm{one}}$ in $\mathbf{EFF}_{\mathrm{one}}$, using Forster et al.'s translation for the multi-shot $shift_0/dollar$.

THEOREM 4.8. *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* $\mathbf{EFF}_{\mathrm{one}}$.

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

We omit the proof, since the argument is quite similar to that in the previous subsection. See Appendix B.2 for the full proof.

5 One-shot effect handlers cannot macro-express asymmetric coroutines

Now that asymmetric coroutines can 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, which is witnessed by *reference cells*.

First, we introduce **REF**, a calculus with ML-style reference cells. Its syntax is defined in Figure 14; the definition of frames and contexts is omitted since they are the same as those of **MAM**. **REF** programs are the computations with no reference cells. Similarly to 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 15 presents the semantics of **REF**.

5.1 Non-macro-expressibility of REF *in terms of* $\mathbf{DEL}_{\text{one}}$

First, we prove that $\mathbf{DEL}_{\text{one}}$ cannot weakly macro-express **REF**:

THEOREM 5.1. *There is no weak macro-translation from* **REF** *to* $\mathbf{DEL}_{\text{one}}$.

To prove the theorem, we need the following lemma:

LEMMA 5.2. *If* $\langle \mathbf{let}\ x = M\ \mathbf{in}\ N; \theta\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, V; \theta''\rangle$, *then there exist a value* V' *and a store* θ' *such that* $\langle M; \theta\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, V'; \theta'\rangle$ *and* $\langle N[V'/x]; \theta'\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, V; \theta''\rangle$.

PROOF. By induction on the length of the reduction sequence. Since the frame $\mathbf{let}\ x = [\,]\ \mathbf{in}\ N$ contains no dollar frame, redexes for the (shift) rule can occur only within M. The frame is preserved until M becomes $\mathbf{return}\, V'$, as the only rule that can consume the frame is (F). □

PROOF OF THEOREM 5.1. Let $\Omega \stackrel{\text{def}}{=} (\lambda x.\ x!\,x)\ \{\lambda x.\ x!\,x\}$, and define

$$\Delta_{\mathrm{A,B}}(x, y) \stackrel{\text{def}}{=} \left(\begin{array}{l}\mathbf{case}\ (x, y)\ \mathbf{of}\ \{\\ \quad (\mathbf{inj}_{\mathrm{A}}\ \_, \mathbf{inj}_{\mathrm{B}}\ \_) \mapsto \mathbf{return}\ ()\\ \quad \_ \mapsto \Omega\}\end{array}\right).$$

Consider the **REF** program

$$M_{\neq} \stackrel{\text{def}}{=} \left(\begin{array}{l}\mathbf{let}\ r = \mathbf{create}\,\mathbf{inj}_{\mathrm{A}}\ ()\ \mathbf{in}\\ \mathbf{let}\ i = \mathbf{get}\ r\ \mathbf{in}\\ \mathbf{let}\ \_ = \mathbf{set}\ r\ \mathbf{inj}_{\mathrm{B}}\ ()\ \mathbf{in}\\ \mathbf{let}\ j = \mathbf{get}\ r\ \mathbf{in}\\ \Delta_{\mathrm{A,B}}(i, j)\end{array}\right).$$

In **REF**, the first **get** returns $\mathbf{inj}_{\mathrm{A}}\ ()$ and the second **get** returns $\mathbf{inj}_{\mathrm{B}}\ ()$, and therefore $\mathrm{Eval}_{\mathbf{REF}}(M_{\neq})$ is defined.

Suppose that there exists a weak macro-translation $\underline{\cdot}$ from **REF** to $\mathbf{DEL}_{\text{one}}$, and let $\mathcal{A}_{\text{create}}$, $\mathcal{A}_{\text{get}}$, $\mathcal{A}_{\text{set}}$ be the syntactic abstractions corresponding to **create**, **get**, and **set**, respectively. Then, since $\underline{\cdot}$ acts homomorphically on **MAM** constructores, the translation $\underline{M_{\neq}}$ has the shape

$$\left(\begin{array}{l}\mathbf{let}\ r = \mathcal{A}_{\text{create}}[\mathbf{inj}_{\mathrm{A}}\ ()]\ \mathbf{in}\\ \mathbf{let}\ i = \mathcal{A}_{\text{get}}[r]\ \mathbf{in}\\ \mathbf{let}\ \_ = \mathcal{A}_{\text{set}}[r, \mathbf{inj}_{\mathrm{B}}\ ()]\ \mathbf{in}\\ \mathbf{let}\ j = \mathcal{A}_{\text{get}}[r]\ \mathbf{in}\\ \Delta_{\mathrm{A,B}}(i, j)\end{array}\right),$$

and contains no continuation label. Since $\mathrm{Eval}_{\mathbf{REF}}(M_{\neq})$ is defined, so is $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}\left(\underline{M_{\neq}}\right)$: there exists a store θ such that

$$\left\langle \underline{M_{\neq}}; \emptyset\right\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\ (); \theta\rangle.$$

Applying Lemma 5.2 repeatedly to this reduction sequence (note that there is no dollar frame enclosing $\underline{M_{\neq}}$) yields values X, V_1, W, V_2 and stores $\theta_0, \theta_1, \theta_2, \theta_3$ such that

$$\langle \mathcal{A}_{\text{create}}[\mathbf{inj}_{\mathrm{A}}\ ()]; \emptyset\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, X; \theta_0\rangle,$$
$$\left\langle \mathcal{A}_{\text{get}}[X]; \theta_0\right\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, V_1; \theta_1\rangle,$$
$$\langle \mathcal{A}_{\text{set}}[X, \mathbf{inj}_{\mathrm{B}}\ ()]; \theta_1\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\, W; \theta_2\rangle,$$

$$\left\langle \mathcal{A}_{\mathrm{get}}[X]; \theta_2 \right\rangle \to^{*}_{\mathbf{D}} \langle \mathbf{return}\, V_2; \theta_3 \rangle, \tag{8}$$

and moreover the evaluation of $\Delta_{\mathrm{A,B}}(V_1, V_2)$ terminates successfully. Thus, $V_1 \equiv \mathbf{inj}_{\mathrm{A}}\, V_1'$ and $V_2 \equiv \mathbf{inj}_{\mathrm{B}}\, V_2'$ for some values V_1' and V_2'.

As Figure 6 shows, a reduction of $\mathbf{DEL}_{\mathrm{one}}$ can only store a continuation under a fresh label or invalidate an existing one; unlike the (set) rule of **REF**, it never replaces the content of a valid entry. Hence, when restricted to the labels reachable from $\mathcal{A}_{\mathrm{get}}[X]$, θ_2 differs from θ_0 only in that some of the entries are invalidated. Since the reduction sequence in (8) results in a value V_2, this implies that no step of that sequence depends on the difference between θ_0 and θ_2. Thus, from the determinism of the reduction, we obtain $V_1 = V_2$.[16] However, this contradicts $V_1 \equiv \mathbf{inj}_{\mathrm{A}}\, V_1'$ and $V_2 \equiv \mathbf{inj}_{\mathrm{B}}\, V_2'$. Therefore, there is no weak macro-translation from **REF** to $\mathbf{DEL}_{\mathrm{one}}$. □

We remark that this proof does not depend on the one-shotness of continuations. Thus, by a similar argument, we can prove that multi-shot delimited-control handlers (thus multi-shot effect handlers) also cannot macro-express reference cells. This proves, in the setting of macro-expressibility, the conjecture of Kammar and Pretnar that reference cells cannot be implemented by effect handlers alone [16].

On the other hand, **AC** can macro-express **REF**, using the encoding introduced in Section 3.4:

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

The simulation relation, $\langle M; \theta \rangle \sim \langle N; \tau \rangle$, holds if and only if there exists a partial function $\eta : \mathscr{L}_{\mathbf{R}} \rightharpoonup \mathscr{L}_{\mathbf{AC}}$ such that:

(1) $N \equiv \underline{M}_{\eta}$
(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[17]

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

Note that (4) specifies the correspondence between source reference cells and target coroutines.

Theorem 5.3. **REF** *is strongly macro-expressible in* **AC**.

Proof. We can show that $M \mapsto \underline{M}$ is a strong macro-translation from **REF** to **AC**. The proof proceeds similarly to other correctness proofs (see Appendix C.2 for the details). □

Theorem 5.4. $\mathbf{DEL}_{\mathrm{one}}$ *cannot weakly macro-express* **AC**. *Moreover,* $\mathbf{EFF}_{\mathrm{one}}$ *also cannot weakly macro-express* **AC**.

Proof. Suppose that a weak macro-translation from **AC** to $\mathbf{DEL}_{\mathrm{one}}$ exists. Then, by composing it with a (strong) macro-translation from **REF** to **AC** (Theorem 5.3), we obtain a

[16] To be precise, V_1 is identical to V_2 modulo freshly generated labels. Rigorously proving this part requires cumbersome analysis of stores; see Appendix C.1 for the details.

[17] *loop* is defined in Figure 10.

weak macro-translation from **REF** to $\mathbf{DEL}_{\mathrm{one}}$, which contradicts Theorem 5.1. The latter statement follows from Theorem 4.6. □

One may find Theorem 5.4 counter-intuitive, since effect handlers (and delimited-control operators) are considered a universal tool for expressing various computational effects, such as coroutines. This gap arises from the strict condition which macro-expressibility imposes on translations: a macro-translation should transform programs *locally* and *homomorphically*. This prevents a translation from introducing a *global* handler—thus effect handlers cannot macro-express global effects such as reference cells.

5.2 Non-macro-expressibility of $\mathbf{DEL}_{\mathrm{one}}$ *in terms of* **REF**

The results obtained so far are that while $\mathbf{DEL}_{\mathrm{one}}$ and $\mathbf{EFF}_{\mathrm{one}}$ are equi-expressive, **AC** is strictly more expressive than both; the difference is witnessed by the non macro-expressibility $\mathbf{REF} \not\rightarrow \mathbf{DEL}_{\mathrm{one}}$. In this section, to complete the picture, we consider the converse direction and prove the following theorem:

THEOREM 5.5. *There is no weak macro-translation from* $\mathbf{DEL}_{\mathrm{one}}$ *to* **REF**.

Since macro-expressibility is transitive, this theorem implies the following result.

THEOREM 5.6. *There is no weak macro-translation from* $\mathbf{EFF}_{\mathrm{one}}$ *to* **REF***, and there is no weak macro-translation from* **AC** *to* **REF**.

The gist of the proof of Theorem 5.5 is that **REF** lacks *control effects*. The proof proceeds as follows:

(1) In **REF**, for any evaluation context $\mathcal{E}$, the following computation cannot terminate successfully:
$$\mathcal{E}[\mathbf{let}\ \_ = M\ \mathbf{in}\ \Omega],$$
where Ω is the diverging computation $(\lambda x.\ x\,!\,x)\{\lambda x.\ x\,!\,x\}$. To successfully terminate the computation, we need to escape from the **let** frame when evaluating M, but **REF** does not have any facility for this, in contrast to $\mathbf{DEL}_{\mathrm{one}}$.

(2) Therefore, for an arbitrary **REF** context C that expects a value and arbitrary computations M_i, **REF** cannot distinguish the computations $C[\{\mathbf{let}\ \_ = M_i\ \mathbf{in}\ \Omega\}]$ from one another.

(3) On the other hand, in $\mathbf{DEL}_{\mathrm{one}}$, the computations of the form
$$\left\langle \left\{\mathbf{let}\ \_ = \mathbf{S_0} k.\ \mathbf{return}\ (\mathbf{inj}_{\mathrm{L}_i}\ ())\ \mathbf{in}\ \Omega\right\}! \middle|\, x.\mathbf{return}\, x \right\rangle,$$
parameterized by variant labels L_i, can be distinguished.

(4) Thus, if there exists a weak macro-translation, **REF** must be able to distinguish the computations of the form
$$\mathcal{A}_{\mathrm{dollar},x}\left[\left\{\mathbf{let}\ \_ = \mathcal{A}_{\mathrm{shift0},k}\left[\mathbf{return}\ (\mathbf{inj}_{L_i}\ ())\right]\ \mathbf{in}\ \Omega\right\}!, \mathbf{return}\, x\right],$$
where $\mathcal{A}_{\mathrm{dollar},x}$ is the syntactic abstraction corresponding to a source dollar binding x in its additional part, and $\mathcal{A}_{\mathrm{shift0},k}$ is the one corresponding to a source shift0 binding k in its body. However, this contradicts (2).

To avoid clutter, we only sketch the proof of (2) and omit the proofs of the other statements, which are straightforward to check; see Appendix D for the details.

Proposition 5.7. *Let* $\mathcal{L} \stackrel{\text{def}}{=} (\mathbf{let}\ \_ = [\ ]\ \mathbf{in}\ \Omega)$. *For any* **REF** *computations* M *and* N, *the thunks* $\{\mathcal{L}[M]\}$ *and* $\{\mathcal{L}[N]\}$ *are observationally equivalent; that is, for every context* C *that expects a value,* $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[M]\}])$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[N]\}])$ *is defined.*

Proof. It suffices to show the "only if" direction, since the converse direction follows by symmetry. Suppose that $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[M]\}])$ is defined and let $\sim$ be the congruence relation on **REF** configurations generated by $\{\mathcal{L}[M']\} \sim \{\mathcal{L}[N']\}$ for any **REF** computations M' and N'. Then we can prove that $\sim$ forms a lock-step simulation, except for a case where a thunk of the form $\{\mathcal{L}[M']\}$ is forced; such a step results in a configuration $\langle \mathcal{E}[\mathcal{L}[M']]; \theta\rangle$ for some evaluation context $\mathcal{E}$ and store θ, whose evaluation never terminates successfully by (1), but this contradicts the assumption that $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[M]\}])$ is defined. Therefore, by replacing each occurrence of $\{\mathcal{L}[M']\}$ with the corresponding $\{\mathcal{L}[N']\}$ in each configuration in the reduction sequence of $C[\{\mathcal{L}[M]\}]$, we obtain a reduction sequence of $C[\{\mathcal{L}[N]\}]$ that terminates successfully. Thus $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[N]\}])$ is defined. □

6 Related work

Expressive power of control operators. As a finer notion than computability, Felleisen proposed the notion of *macro-expressibility* to rigorously compare the expressive power of two programming languages when one is an extension of the other [8]. This notion was later adjusted to compare two languages when both are extensions of a common language. Riecke and Thielecke compared first-class continuations (*call/cc*) and exceptions in a typed language and proved that they cannot macro-express each other [27]. Shan relaxed macro-expressibility into *top-macro-expressibility* and compared *shift/reset* with dynamic delimited-control operators such as *control/prompt* [28].

More recently, Forster et al. compared three abstractions for user-defined effects: effect handlers, monads, and delimited-control operators (in particular, *shift0/dollar*) [11, 12]. They proved that, in an untyped setting, the three are all equi-expressive. They also compared the expressive power in a typed setting, and proved that typed monads can macro-express typed delimited-control operators but not typed effect handlers. Following Forster et al.'s approach, Piróg et al. established typed equivalence between effect handlers and delimited-control operators in a polymorphic type system [24]. Ikemori et al. extended this result to the equivalence of labeled effect handlers and labeled delimited-control operators [14].

However, all of these previous studies concentrated on multi-shot control operators, and to our knowledge, the present paper is the first study to conduct a systematic comparison of the relative expressive power 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 [15], but they did not provide a formal justification. Meanwhile, earlier work by de Moura and Ierusalimschy examined the expressive power of symmetric coroutines, asymmetric coroutines, and one-shot subcontinuations in the presence of mutable state [5]. In contrast, we study the extensions of a base calculus without mutable states, which isolates the expressive power of each control operator itself: it is exactly the ability to macro-express global effects such as reference cells that separates coroutines from delimited-control operators, since macro-translations into $\mathbf{DEL}_{\text{one}}$ are local and hence cannot introduce a dollar enclosing the entire program (Section 5.1).

$$
\begin{tikzcd}
 & \mathbf{AC} & \\
\mathbf{EFF}_{\mathrm{one}} \arrow[ur] \arrow[r, shift left] & \mathbf{DEL}_{\mathrm{one}} \arrow[u] \arrow[l, shift left] & \mathbf{REF} \arrow[ul]
\end{tikzcd}
$$

Fig. 16. Macro-expressibility obtained in this work. An arrow indicates the existence of a strong macro-translation in that direction, while the absence of an arrow indicates the non-existence of a weak macro-translation.

One-shot continuations and control operators. Historically, the notion of one-shot continuations was formalized in the Scheme community by Bruggeman et al. [3], who introduced them as an efficient yet less expressive variant of general (multi-shot) first-class continuations, motivated by the desire to avoid the stack-copying overhead upon invocation. Their implementation of one-shot continuations, named *call/1cc*, is available in Chez Scheme [7].

Recently, one-shot continuations have been attracting attention as a means of integrating control abstractions into mainstream programming languages. Since version 5.0, OCaml has implemented one-shot effect handlers because they allow smooth integration into OCaml's runtime system [29]. Phipps-Costin et al. proposed WasmFX, which extends WebAssembly with one-shot effect handlers, enabling type-safe non-local control features such as async/await [23]. Although WasmFX gives types to continuations, it enforces one-shotness dynamically by aborting execution on a second invocation; OCaml likewise raises a runtime error. Our calculi model this discipline directly, and we give a first formal account of what such one-shot control abstractions can and cannot macro-express.

The one-shot usage of continuations has also been studied from a type-theoretic and semantic perspective. Van Rooij and Krebbers presented Affect, an affine type-and-effect system equipped with effect handlers and reference cells [31]. In Affect, one-shot and multi-shot continuations coexist, and their usage is statically tracked by the type system, in contrast to our untyped setting, where one-shotness is enforced dynamically. From a semantic perspective, Berdine et al. demonstrated that various control behaviors, such as exceptions, labeled jumping, and coroutines, can be CPS-transformed into a target calculus with linear types [2]. Notably, they pointed out that, in these transforms, continuations are *used* linearly. Based on this observation, Hasegawa [13] showed that Berdine et al.'s linear CPS is a special instance of the monadic transforms of the computational lambda calculus [22] into the linear lambda calculus, and proved its full completeness. It would be interesting to analyze the expressive power of one-shot control operators from this semantic perspective.

7 Conclusion

This study investigated the expressive power of one-shot control operators, summarized in Figure 16, resulting in the following two key findings:

(1) One-shot delimited-control operators and one-shot effect handlers can be macro-expressed by asymmetric coroutines.
(2) The converse direction does not hold, and this non-macro-expressibility is witnessed by reference cells.

Previously, the former result was considered trivial; however, we found a gap in the literature and fixed the problem by introducing reference cells, which can be simulated using coroutines, and using them to manage the validity of continuations. As this result demonstrates, establishing such a connection requires rigorous analysis.

The latter result depends on the non-macro-expressibility of reference cells in terms of effect handlers. Kammar and Pretnar conjectured that effect handlers alone cannot implement reference cells, while establishing that they can macro-express dynamically scoped state [16]. We proved their conjecture in the macro-expressibility setting, and the proof exploited the *locality* of macro-translations.

This non-macro-expressibility result suggests that macro-expressibility is too strict as a measure for comparing expressive power. We conjecture that, under a suitable relaxation of the locality of macro-translations, $\mathbf{EFF}_{\text{one}}$ 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, as Shan did for comparing the expressive power of delimited-control operators [28]. We leave the exploration of this idea for future work.

Comparing the expressive power in a typed setting is an important direction to explore. For effect handlers and delimited-control operators, there are a number of proposals for their type systems, including linear/affine ones [30, 31]. Such type systems are particularly interesting here, since they would enforce one-shotness statically, whereas our calculi detect its violation dynamically. As Forster et al.'s results indicate, moving to a typed setting can change the relationships obtained in the untyped setting [12]; whether the macro-expressibility results in Figure 16 carry over remains open.

Another important direction is to design type systems for one-shot control operators, in the spirit of Berdine et al.'s linearly used continuations [2], whose examples already include a restricted form of coroutines. The asymmetric coroutines considered here, however, are much more general, and the design space for their type systems is correspondingly broad—thus requiring a systematic approach. An instance of such an approach is Anton and Thiemann's type system for coroutines [1], which is derived via a translation to a calculus with the *shift/reset* operators and reference cells. Whether a type system can be derived in the same spirit from our macro-translations remains to be investigated.

Acknowledgments

The authors were supported by JSPS KAKENHI Grant Number JP23K24819, Japan. Kentaro Kobayashi is supported by JST BOOST, Grant Number JPMJBS2414, Japan.

References

[1] Konrad Anton and Peter Thiemann. 2010. Towards Deriving Type Systems and Implementations for Coroutines. In *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6461)*, Kazunori Ueda (Ed.). Springer, 63–79. doi:10.1007/978-3-642-17164-2_6

[2] Josh Berdine, Peter W. O'Hearn, Uday S. Reddy, and Hayo Thielecke. 2002. Linear Continuation-Passing. *High. Order Symb. Comput.* 15, 2-3 (2002), 181–208. doi:10.1023/A:1020891112409

[3] Carl Bruggeman, Oscar Waddell, and R. Kent Dybvig. 1996. Representing Control in the Presence of One-Shot Continuations. In *Proceedings of the ACM SIGPLAN'96 Conference on Programming Language Design and Implementation (PLDI), Philadephia, Pennsylvania, USA, May 21-24, 1996*, Charles N. Fischer (Ed.). ACM, 99–107. doi:10.1145/231379.231395

[4] Olivier Danvy and Andrzej Filinski. 1989. A functional abstraction of typed contexts. DIKU Rapport 89/12.

[5] Ana Lúcia de Moura and Roberto Ierusalimschy. 2009. Revisiting coroutines. *ACM Trans. Program. Lang. Syst.* 31, 2 (2009), 6:1–6:31. doi:10.1145/1462166.1462167

[6] Paulo Emílio de Vilhena and François Pottier. 2021. A separation logic for effect handlers. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–28. doi:10.1145/3434314

[7] R. Kent Dybvig. 2006. The development of Chez Scheme. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006, Portland, Oregon, USA, September 16-21, 2006*, John H. Reppy and Julia Lawall (Eds.). ACM, 1–12. doi:10.1145/1159803.1159805

[8] Matthias Felleisen. 1991. On the Expressive Power of Programming Languages. *Sci. Comput. Program.* 17, 1-3 (1991), 35–75. doi:10.1016/0167-6423(91)90036-W

[9] Matthias Felleisen and Daniel P. Friedman. 1987. A Reduction Semantics for Imperative Higher-Order Languages. In *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)*, J. W. de Bakker, A. J. Nijman, and Philip C. Treleaven (Eds.). Springer, 206–223. doi:10.1007/3-540-17945-3_12

[10] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 193–202. doi:10.1109/LICS.1999.782615

[11] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2017. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *Proc. ACM Program. Lang.* 1, ICFP (2017), 13:1–13:29. doi:10.1145/3110257

[12] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2019. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *J. Funct. Program.* 29 (2019), e15. doi:10.1017/S0956796819000121

[13] Masahito Hasegawa. 2002. Linearly Used Effects: Monadic and CPS Transformations into the Linear Lambda Calculus. In *Functional and Logic Programming, 6th International Symposium, FLOPS 2002, Aizu, Japan, September 15-17, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2441)*, Zhenjiang Hu and Mario Rodríguez-Artalejo (Eds.). Springer, 167–182. doi:10.1007/3-540-45788-7_10

[14] Kazuki Ikemori, Youyou Cong, and Hidehiko Masuhara. 2023. Typed Equivalence of Labeled Effect Handlers and Labeled Delimited Control Operators. In *International Symposium on Principles and Practice of Declarative Programming, PPDP 2023, Lisboa, Portugal, October 22-23, 2023*, Santiago Escobar and Vasco T. Vasconcelos (Eds.). ACM, 4:1–4:13. doi:10.1145/3610612.3610616

[15] Roshan P. James and Amr Sabry. 2011. Yield: Mainstream Delimited Continuations. In *Proceedings on the 1st International Workshop on Theory and Practice of Delimited Continuations (TPDC 2011), 2011*. 12 pages.

[16] Ohad Kammar and Matija Pretnar. 2017. No value restriction is needed for algebraic effects and handlers. *J. Funct. Program.* 27 (2017), e7. doi:10.1017/S0956796816000320

[17] Satoru Kawahara and Yukiyoshi Kameyama. 2020. One-Shot Algebraic Effects as Coroutines. In *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)*, Aleksander Byrski and John Hughes (Eds.). Springer, 159–179. doi:10.1007/978-3-030-57761-2_8

[18] Oleg Kiselyov and Chung-chieh Shan. 2007. A Substructural Type System for Delimited Continuations. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4583)*, Simona Ronchi Della Rocca (Ed.). Springer, 223–239. doi:10.1007/978-3-540-73228-0_17

[19] Kentaro Kobayashi and Yukiyoshi Kameyama. 2025. Expressive Power of One-Shot Control Operators and Coroutines. In *Programming Languages and Systems - 23rd Asian Symposium, APLAS 2025, Bengaluru, India, October 27-30, 2025, Proceedings (Lecture Notes in Computer Science, Vol. 16201)*, Alex Potanin (Ed.). Springer, 88–106. doi:10.1007/978-981-95-3585-9_5

[20] Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer.

[21] Marek Materzok and Dariusz Biernacki. 2012. A Dynamic Interpretation of the CPS Hierarchy. In *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7705)*, Ranjit Jhala and Atsushi Igarashi (Eds.). Springer, 296–311. doi:10.1007/978-3-642-35182-2_21

[22] Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE

Computer Society, 14–23. doi:10.1109/LICS.1989.39155

[23] Luna Phipps-Costin, Andreas Rossberg, Arjun Guha, Daan Leijen, Daniel Hillerström, K. C. Sivaramakrishnan, Matija Pretnar, and Sam Lindley. 2023. Continuing WebAssembly with Effect Handlers. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 460–485. doi:10.1145/3622814

[24] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2019. Typed Equivalence of Effect Handlers and Delimited Control. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, Dortmund, Germany, June 24-30, 2019 (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:16. doi:10.4230/LIPICS.FSCD.2019.30

[25] Gordon D. Plotkin and John Power. 2003. Algebraic Operations and Generic Effects. *Appl. Categorical Struct.* 11, 1 (2003), 69–94. doi:10.1023/A:1023064908962

[26] Gordon D. Plotkin and Matija Pretnar. 2009. Handlers of Algebraic Effects. In *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)*, Giuseppe Castagna (Ed.). Springer, 80–94. doi:10.1007/978-3-642-00590-9_7

[27] Jon G. Riecke and Hayo Thielecke. 1999. Typed Exeptions and Continuations Cannot Macro-Express Each Other. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1644)*, Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen (Eds.). Springer, 635–644. doi:10.1007/3-540-48523-6_60

[28] Chung-chieh Shan. 2007. A static simulation of dynamic delimited control. *High. Order Symb. Comput.* 20, 4 (2007), 371–401. doi:10.1007/S10990-007-9010-4

[29] K. C. Sivaramakrishnan, Stephen Dolan, Leo White, Tom Kelly, Sadiq Jaffer, and Anil Madhavapeddy. 2021. Retrofitting effect handlers onto OCaml. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 206–221. doi:10.1145/3453483.3454039

[30] Wenhao Tang, Daniel Hillerström, Sam Lindley, and J. Garrett Morris. 2024. Soundly Handling Linearity. *Proc. ACM Program. Lang.* 8, POPL (2024), 1600–1628. doi:10.1145/3632896

[31] Orpheas van Rooij and Robbert Krebbers. 2025. Affect: An Affine Type and Effect System. *Proc. ACM Program. Lang.* 9, POPL (2025), 126–154. doi:10.1145/3704841

A Supplementary proofs for Section 3

A.1 Simulation relation

A.1.1 Well-formedness

Definition A.1. For a $\mathbf{DEL}_{\mathrm{one}}$ computation M, let $\mathrm{LB}(M)$ be the set of continuation labels occurring in M. For a $\mathbf{DEL}_{\mathrm{one}}$ store θ, define

$$\mathrm{LB}(\theta) \overset{\mathrm{def}}{=} \bigcup\{\mathrm{LB}(\theta(l)) \mid l \in \mathrm{Dom}(\theta), \theta(l) \neq \mathbf{nil}\}.$$

For a $\mathbf{DEL}_{\mathrm{one}}$ configuration $C = \langle M; \theta\rangle$, define

$$\mathrm{LB}(C) \overset{\mathrm{def}}{=} \mathrm{LB}(M) \cup \mathrm{LB}(\theta).$$

For **AC** computations, stores, and configurations, define these functions accordingly.

Definition A.2 ($\mathbf{DEL}_{\mathrm{one}}$ well-formedness). A $\mathbf{DEL}_{\mathrm{one}}$ configuration $\langle M; \theta\rangle$ is *well-formed*, written $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M; \theta\rangle)$, if and only if

$$\mathrm{LB}(\langle M; \theta\rangle) \subseteq \mathrm{Dom}(\theta).$$

Proposition A.3 (preservation of $\mathbf{DEL}_{\mathrm{one}}$ well-formedness). *If* $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M; \theta\rangle)$ *and* $\langle M; \theta\rangle \to_{\mathbf{D}} \langle M'; \theta'\rangle$*, then* $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M'; \theta'\rangle)$.

Proof. We prove this by case analysis on the $\mathbf{DEL}_{\mathrm{one}}$ beta-reduction rule used in the source step. The surrounding context itself is unchanged by the reduction, so it suffices to check that the reduct M' contains only labels already defined in the updated store.

In the **MAM** cases, no continuation label is generated and the store is unchanged. Every continuation label occurring in the reduct already occurs in the redex, hence is contained in $\mathrm{Dom}(\theta)$ by the premise.

In the (ret) case,

$$\langle\langle \mathbf{return}\, V \,|\, x.N\rangle; \theta\rangle \rightarrow_{\mathbf{D}} \langle N[V/x]; \theta\rangle,$$

and every label in $N[V/x]$ already occurs in N or in V. Hence it already occurs in the premise and belongs to $\mathrm{Dom}(\theta)$.

In the (shift) case,

$$\langle\langle \mathcal{H}[\mathbf{S_0} k.\ M] \,|\, x.N\rangle; \theta\rangle \rightarrow_{\mathbf{D}} \langle M[l/k]; \theta[l := \lambda y.\ \langle \mathcal{H}[\mathbf{return}\, y] \,|\, x.N\rangle]\rangle,$$

where l is fresh. All old labels remain in $\mathrm{Dom}(\theta)$, and the only new label that can occur in either the current computation or the new store entry is l, which is added to the store domain by the reduction.

In the (throw) case,

$$\langle \mathbf{throw}\, l\, V; \theta\rangle \rightarrow_{\mathbf{D}} \langle\langle \mathcal{H}[\mathbf{return}\, V] \,|\, x.N\rangle; \theta[l := \mathbf{nil}]\rangle,$$

with $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\, y] \,|\, x.N\rangle$. The store domain is unchanged, and every label in H, V, or N already occurs either in the current computation or in the store entry for $\theta(l)$ of the premise. Hence all labels in the reduct lie in $\mathrm{Dom}(\theta[l := \mathbf{nil}])$.

The (fail) case does not occur. □

*Definition A.4 (**AC** well-formedness).* For an **AC** term E, let $AL(E)$ be the set of active labels, i.e., labels occurring in a subterm of the form $l : M$. An **AC** configuration $\langle N; \tau\rangle$ is *well-formed*, written $\mathsf{WF}_{\mathbf{AC}}(\langle N; \tau\rangle)$, if and only if

(1) $\mathsf{WF}_{\mathbf{AC}}(N)$ holds;
(2) $AL(V) = \emptyset$ for every value $V \in \mathrm{Im}(\tau)$;
(3) for every $l \in AL(N)$, $\tau(l) = \mathbf{nil}$.

Here, $\mathsf{WF}_{\mathbf{AC}}(N)$ is a predicate inductively defined by the following rules

$$\frac{AL(V) \cup AL(M) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M)} \qquad \frac{AL(V) \cup (\cup_i AL(M_i)) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L_i}}\ x_i \mapsto M_i)_i\})}$$

$$\frac{AL(V) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(V!)} \qquad \frac{AL(V) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{return}\ V)} \qquad \frac{\mathsf{WF}_{\mathbf{AC}}(M) \qquad AL(N) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{let}\ x = M\ \mathbf{in}\ N)}$$

$$\frac{AL(M) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\lambda x.\ M)} \qquad \frac{\mathsf{WF}_{\mathbf{AC}}(M) \qquad AL(N) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(M\ V)} \qquad \frac{AL(M) \cup AL(N) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\langle M, N\rangle)}$$

$$\frac{\mathsf{WF}_{\mathbf{AC}}(M)}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{prj}_i\ M)} \qquad \frac{\mathsf{WF}_{\mathbf{AC}}(M) \qquad l \notin AL(M)}{\mathsf{WF}_{\mathbf{AC}}(l : M)} \qquad \frac{AL(V) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{create}\ V)}$$

$$\frac{AL(V) \cup AL(W) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{resume}\ V\ W)} \qquad \frac{AL(V) = \emptyset}{\mathsf{WF}_{\mathbf{AC}}(\mathbf{yield}\ V)}$$

Proposition A.5 (preservation of **AC** well-formedness). *Suppose that $\langle C[M]; \theta\rangle$ is well-formed and $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$. Then, $\langle C[M']; \theta'\rangle$ is also well-formed.*

Proof. We prove the proposition by case analysis on $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$.

Case (**MAM**)**.** We prove one representative case. 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 C, we conclude that $\langle C[M[V/x]]; \theta\rangle$ is well-formed.

Case (create)**.** Suppose that $\langle M; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$ is

$$\langle \mathbf{create}\ V; \theta\rangle \to^{\beta}_{\mathbf{AC}} \langle \mathbf{return}\ l; \theta[l := V]\rangle,$$

where l is a fresh label. Since l is fresh, $l \notin AL(C)$. The well-formedness of $\mathbf{create}\ V$ ensures $AL(V) = \emptyset$. By induction on C, we obtain that $\langle C[\mathbf{return}\ l]; \theta[l := V]\rangle$ 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 $\langle C[\mathbf{resume}\ l\ V]; \theta\rangle$, it follows that $\theta(l) = \mathbf{nil}$, but this is a contradiction. Therefore, l is distinct in $\langle C[l : (\theta(l)!\ V)]; \theta[l := \mathbf{nil}]\rangle$ and mapped to $\mathbf{nil}$ in $\theta[l := \mathbf{nil}]$. From the well-formedness of $\langle C[\mathbf{resume}\ l\ M]; \theta\rangle$, 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 C.

Case (fail)**.** This case does not occur, as $\langle C[M]; \theta\rangle$ reduces to $\langle C[M']; \theta'\rangle$, not $\perp$.

Case (ret)**.** This case is immediate from the definition of well-formedness.

Case (yield)**.** Suppose $\langle M; \theta\rangle \rightarrow^{\beta}_{\mathbf{AC}} \langle M'; \theta'\rangle$ is

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

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

Lemma A.6. *For an* **AC** *computation N and its subcomputation N', if* $\mathsf{WF}_{\mathbf{AC}}(N)$*, then* $\mathsf{WF}_{\mathbf{AC}}(N')$*. Moreover, for a store τ,* $\mathsf{WF}_{\mathbf{AC}}(\langle N; \tau\rangle)$ *implies* $\mathsf{WF}_{\mathbf{AC}}(\langle N'; \tau\rangle)$.

Proof. The former statement can be proven by routine induction on the structure of N. Since $AL(N') \subseteq AL(N)$, it is straightforward to check the other side conditions. □

Since the well-formedness of both calculi is preserved under reduction, we do not explicitly mention it in the subsequent proofs.

A.1.2 Invariants

Definition A.7 (label map). A label map is a partial function

$$\eta : \mathscr{L}_{\mathbf{D}} \rightharpoonup \mathscr{L}_{\mathbf{A}} \times \mathscr{L}_{\mathbf{A}}.$$

If $\eta(l) = (mc, m)$, then mc is the reference cell that represents l, and m is the coroutine label that mc may hold as $\mathbf{inj}_{\mathrm{Valid}}\, m$. We parameterize and extend the translation with η, by translating l to mc. We call this extension a *runtime translation* and denote it by $\underline{\cdot}_{\eta}$. Similarly, we can extend the runtime translation to a translation on contexts by mapping a hole to a hole.

Definition A.8 (store-map coherence). For a $\mathbf{DEL}_{\mathrm{one}}$ store θ and a label map η, we define $\mathsf{Coh}(\theta, \eta)$ to hold if and only if the following conditions are satisfied:

(C1) $\mathrm{Dom}(\eta) = \mathrm{Dom}(\theta)$;
(C2) the first projection of η is injective: if $\eta(l) = (mc, m)$ and $\eta(l') = (mc, m')$, then $l = l'$.

Definition A.9. For a pure $\mathbf{DEL}_{\mathrm{one}}$ context H and a computation M with a free variable x, define

$$\mathrm{Cont}_{\eta}(H, x, M) \stackrel{\mathrm{def}}{=} \left\{\lambda y.\ \mathbf{let}\ x = \underline{\mathcal{H}[\mathbf{return}\, y]}_{\eta}\ \mathbf{in\ return}\left\{\lambda\_.\ \underline{M}_{\eta}\right\}\right\}.$$

Definition A.10 (invariant conditions). For a $\mathbf{DEL}_{\mathrm{one}}$ store θ, an **AC** store τ, and a runtime label map η, we define $\mathsf{Inv}(\theta, \tau, \eta)$ to hold if and only if the following conditions are satisfied.

(IC1) If $\theta(l) = \mathbf{nil}$ and $\eta(l) = (mc, m)$, then $\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\mathrm{Invalid}}\, ())$.
(IC2) If $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\, y] \,|\, x.M\rangle$ and $\eta(l) = (mc, m)$, then
  (a) $\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\, m)$,
  (b) $\tau(m) = \mathrm{Cont}_{\eta}(H, x, M)$,
  (c) if $l' \neq l$ and $\eta(l') = (mc', m)$, then $\theta(l') = \mathbf{nil}$.

Condition (IC1) ensures that a reference cell representing an invalid continuation has $\mathbf{inj}_{\mathrm{Invalid}}\, ()$. Condition (IC2) states that a valid continuation corresponds to a valid reference cell containing a coroutine label that realizes the continuation. Moreover, condition (IC2-c) says that a coroutine label m realizes only one valid source continuation label. In other words, since the second component of η is not necessarily injective, the label m may have more than one corresponding source continuations labels, but condition (IC2-c) ensures that only one of them is valid.

A.1.3 Simulation relation

Definition A.11. For coroutine label m, an **AC** computation N, and a $\mathbf{DEL}_{\text{one}}$ computation M with a free variable x, define

$$\mathrm{Act}_\eta(N, x, M, m) \stackrel{\text{def}}{=} \begin{pmatrix} \textbf{let}\ \mathit{res} = m : \left(\textbf{let}\ x = N\ \textbf{in return}\left\{\lambda\_.\ \underline{M}_\eta\right\}\right)\ \textbf{in} \\ \mathit{res}!\ \left\{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\, m\right\} \end{pmatrix},$$

and for an **AC** evaluation context $\mathcal{D}$,

$$\mathrm{ActCtx}_\eta(\mathcal{D}, x, M, m) \stackrel{\text{def}}{=} \begin{pmatrix} \textbf{let}\ \mathit{res} = m : \left(\textbf{let}\ x = \mathcal{D}\ \textbf{in return}\left\{\lambda\_.\ \underline{M}_\eta\right\}\right)\ \textbf{in} \\ \mathit{res}!\ \left\{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\, m\right\} \end{pmatrix}$$

Definition A.12 (core relation). For a label map η, define a relation $\overset{\eta}{\sim} \subseteq \mathrm{Conf}_{\mathbf{DEL}_{\text{one}}} \times \mathrm{Conf}_{\mathbf{AC}}$ inductively as follows.

For every constructor E among

$$\textbf{case}\ V\ \textbf{of}\ (x_1, x_2) \mapsto M, \qquad \textbf{case}\ V\ \textbf{of}\ \{(\textbf{inj}_{\mathrm{l_i}}\, x_i \mapsto m_i)_i\}, \qquad V!, \qquad \textbf{return}\, V,$$

$$\lambda x.\ M, \qquad \langle M_1, M_2\rangle, \qquad \mathbf{S_0}k.\ M, \qquad \textbf{throw}\, V\, W,$$

we have the structural clause

$$\langle E; \theta\rangle \ \overset{\eta}{\sim}\ \left\langle \underline{E}_\eta; \tau\right\rangle,$$

for any θ and τ.

In addition,

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

$$\frac{\langle M; \theta\rangle \ \overset{\eta}{\sim}\ \langle N; \tau\rangle}{\langle M\, V; \theta\rangle \ \overset{\eta}{\sim}\ \left\langle N\, \underline{V}_\eta; \tau\right\rangle}, \qquad \frac{\langle M; \theta\rangle \ \overset{\eta}{\sim}\ \langle N; \tau\rangle}{\left\langle \textbf{prj}_i\, M; \theta\right\rangle \ \overset{\eta}{\sim}\ \left\langle \textbf{prj}_i\, N; \tau\right\rangle},$$

and the dollar clause:

$$\frac{\langle M_1; \theta\rangle \ \overset{\eta}{\sim}\ \langle N_1; \tau\rangle \qquad \tau(m) = \textbf{nil} \qquad \forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \textbf{nil}}{\langle\langle M_1 \,|\, x.M_2\rangle; \theta\rangle \ \overset{\eta}{\sim}\ \left\langle \mathrm{Act}_\eta(N_1, x, M_2, m); \tau\right\rangle}.$$

Definition A.13 (core relation on contexts). For a label map η, define a relation $\overset{\eta}{\sim}_{\mathsf{c}} \subseteq \mathrm{CConf}_{\mathbf{DEL}_{\text{one}}} \times \mathrm{CConf}_{\mathbf{AC}}$ inductively as follows.

$$\langle [\,]; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \langle [\,]; \tau\rangle, \qquad \frac{\langle C; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \langle \mathcal{D}; \tau\rangle}{\langle \textbf{let}\ x = C\ \textbf{in}\ M_2; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \left\langle \textbf{let}\ x = \mathcal{D}\ \textbf{in}\ \underline{M_2}_\eta; \tau\right\rangle},$$

$$\frac{\langle C; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \langle \mathcal{D}; \tau\rangle}{\langle C\, V; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \left\langle \mathcal{D}\, \underline{V}_\eta; \tau\right\rangle}, \qquad \frac{\langle C; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \langle \mathcal{D}; \tau\rangle}{\left\langle \textbf{prj}_i\, C; \theta\right\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \left\langle \textbf{prj}_i\, \mathcal{D}; \tau\right\rangle},$$

$$\frac{\langle C; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \langle \mathcal{D}; \tau\rangle \qquad \tau(m) = \textbf{nil} \qquad \forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \textbf{nil}}{\langle\langle C \,|\, x.M_2\rangle; \theta\rangle \ \overset{\eta}{\sim}_{\mathsf{c}}\ \left\langle \mathrm{ActCtx}_\eta(\mathcal{D}, x, M_2, m); \tau\right\rangle}.$$

We defined the core relation on configurations, not on computations. This is because the target configuration has an active coroutine m in it, which additionally requires the two conditions on the source and target stores. These premises are complementary to the invariant conditions. The invariant conditions are concerned with continuations to be invoked—thus suspended coroutines. On the other hand, these two premises are concerned with the current continuation to be captured—thus the active coroutine m.

We could have included the following structural clause:

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

However, we deliberately excluded it because if we reduce the target configuration, we obtain[18]

$$\left\langle \underline{\langle M_1 \,|\, x.M_2\rangle}_\eta ;\tau \right\rangle \to^{+}_{\mathbf{AC}} \left\langle \mathrm{Act}_\eta(N_1, x, M_2, m);\tau' \right\rangle,$$

for some m and τ', and

$$\langle\langle M_1 \,|\, x.M_2\rangle;\theta\rangle \overset{\eta}{\sim} \left\langle \mathrm{Act}_\eta(N_1, x, M_2, m);\tau' \right\rangle.$$

In the simulation proof, we first *canonicalize* the target configuration and deal only with such resulting target representatives—which makes, somewhat surprisingly, the proof simpler.

Definition A.14 (core relation with side conditions). For $\mathbf{DEL}_{\mathrm{one}}$ and **AC** configurations $C = \langle M;\theta\rangle$ and $D = \langle N;\tau\rangle$, we write $C \overset{\eta}{\sim}_\sharp D$ if and only if all of the following hold:

(1) $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(C)$;
(2) $\mathsf{WF}_{\mathbf{AC}}(D)$;
(3) $\mathsf{Coh}(\theta,\eta)$;
(4) $\mathsf{Inv}(\theta,\tau,\eta)$;
(5) $C \overset{\eta}{\sim} D$.

Definition A.15 (simulation relation). For $\mathbf{DEL}_{\mathrm{one}}$ and **AC** configurations C and D, define $C \sim D$ if and only if either $C = D = \bot$, or there exist a label map η and an **AC** configuration D' such that

$$D \to^{*}_{\mathbf{AC}} D' \quad \text{and} \quad C \overset{\eta}{\sim}_\sharp D'.$$

A.2 Proof

A.2.1 Lemmas

Lemma A.16 (substitution). *For any* $\mathbf{DEL}_{\mathrm{one}}$ *term* E *with free variables* $x_1, \ldots, x_n$*,* $\mathbf{DEL}_{\mathrm{one}}$ *values* $V_1, \ldots, V_n$*, and label map* η*,*

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

For any $\mathbf{DEL}_{\mathrm{one}}$ *pure context* H *and* $\mathbf{DEL}_{\mathrm{one}}$ *computation* M*, and label map* η*,*

$$\underline{\mathcal{H}}_\eta[\underline{M}_\eta] \equiv \underline{\mathcal{H}[M]}_\eta.$$

Proof. Both are proved by induction on E and $\mathcal{H}$, respectively. □

Lemma A.17. *The following AC reductions hold.*

[18] We prove a more general statement in Section A.2.2.

(1) For fresh h,

$$\langle ref!\,V;\tau\rangle \to^{+}_{\mathbf{AC}} \langle \mathbf{return}\,l;\tau[h := RefCell(V)]\rangle,$$

where l is a fresh label.

(2) If $\tau(l) = RefCell(V)$, then

$$\langle get!\,();\tau\rangle \to^{+}_{\mathbf{AC}} \langle \mathbf{return}\,V;\tau\rangle,$$

(3) If $\tau(l) = RefCell(V)$, then

$$\langle set!\,W;\tau\rangle \to^{+}_{\mathbf{AC}} \langle \mathbf{return}\,();\tau[h := RefCell(W)]\rangle,$$

(4) $\langle fail!;\tau\rangle \to^{+}_{\mathbf{AC}} \bot$.

Proof. By routine calculation. □

Lemma A.18 (pure contexts). *If $\mathcal{H}$ is a pure $\mathbf{DEL}_{\text{one}}$ context, then for any* **AC** *context $\mathcal{D}$,*

$$\langle \mathcal{H};\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle \qquad \text{iff} \qquad \mathcal{D} \equiv \underline{H}_{\eta}.$$

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

Lemma A.19. *If $\langle C;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle$ and $\langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle$, then*

$$\langle C[M];\theta\rangle \overset{\eta}{\sim} \langle \mathcal{D}[N];\tau\rangle.$$

Proof. We prove the lemma by induction on the derivation of

$$\langle C;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle.$$

If both contexts are holes, then the conclusion is exactly the second premise.

Suppose the last rule introduces a let-frame. Then

$$C = (\mathbf{let}\ x = C_0\ \mathbf{in}\ M_2), \qquad \mathcal{D} = (\mathbf{let}\ x = \mathcal{D}_0\ \mathbf{in}\ \underline{M_2}_{\eta}),$$

and $\langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0;\tau\rangle$. By the induction hypothesis,

$$\langle C_0[M];\theta\rangle \overset{\eta}{\sim} \langle \mathcal{D}_0[N];\tau\rangle.$$

Applying the let-clause of the core relation gives the required conclusion. The application-frame and projection-frame cases are identical, using the corresponding clauses of the core relation.

Finally suppose the last rule introduces a dollar frame. Then

$$C = \langle C_0 \,|\, x.M_2\rangle, \qquad \mathcal{D} = \mathrm{ActCtx}_{\eta}(\mathcal{D}_0, x, M_2, m),$$

with

$$\langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0;\tau\rangle, \qquad \tau(m) = \mathbf{nil},$$

and

$$\forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \mathbf{nil}.$$

By the induction hypothesis,

$$\langle C_0[M];\theta\rangle \overset{\eta}{\sim} \langle \mathcal{D}_0[N];\tau\rangle.$$

The dollar clause of the core relation, together with the two side conditions above, yields

$$\langle\langle C_0[M] \,|\, x.M_2\rangle;\theta\rangle \overset{\eta}{\sim} \big\langle \mathrm{Act}_{\eta}(\mathcal{D}_0[N], x, M_2, m);\tau\big\rangle.$$

Since $\mathrm{Act}_\eta(\mathcal{D}_0[N], x, M_2, m) \equiv (\mathrm{ActCtx}_\eta(\mathcal{D}_0, x, M_2, m))[N]$, this is the desired judgement. □

Lemma A.20. *If*

$$\langle C[M];\theta\rangle \overset{\eta}{\sim} \langle P;\tau\rangle,$$

then there exist a target evaluation context $\mathcal{D}$ and a target computation N such that

$$P \equiv \mathcal{D}[N], \qquad \langle C;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle, \qquad \langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle.$$

Proof. We prove the lemma by induction on the structure of C.

If $C = [\,]$, take $\mathcal{D} = [\,]$ and $N = P$. The contextual relation for holes is immediate, and the third conclusion is the premise.

Suppose $C = (\mathbf{let}\ x = C_0\ \mathbf{in}\ M_2)$. Then the last rule used in the derivation of $\langle \mathbf{let}\ x = C_0[M]\ \mathbf{in}\ M_2;\theta\rangle \overset{\eta}{\sim} \langle P;\tau\rangle$ must be the let-clause of the core relation. Therefore

$$P \equiv \mathbf{let}\ x = P_0\ \mathbf{in}\ \underline{M_2}_\eta$$

for some P_0, and

$$\langle C_0[M];\theta\rangle \overset{\eta}{\sim} \langle P_0;\tau\rangle.$$

Applying the induction hypothesis to this gives $\mathcal{D}_0$ and N such that $P_0 \equiv \mathcal{D}_0[N]$, $\langle C,;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D},;\tau\rangle$, and $\langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle$. Take

$$\mathcal{D} = \mathbf{let}\ x = \mathcal{D}_0\ \mathbf{in}\ \underline{M_2}_\eta.$$

The application and projection cases are the same.

Suppose $C = \langle C_0 | x.M_2\rangle$. Then the last rule of the core relation derivation must be the dollar clause. Hence for some m and P_0,

$$P \equiv \mathrm{Act}_\eta(P_0, x, M_2, m), \quad \langle C_0[M];\theta\rangle \overset{\eta}{\sim} \langle P_0;\tau\rangle,$$

and the side conditions

$$\tau(m) = \mathbf{nil}, \qquad \forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \mathbf{nil}$$

hold. Applying the induction hypothesis to the subconfiguration involving P_0 gives $\mathcal{D}_0$ and N such that

$$P_0 \equiv \mathcal{D}_0[N], \qquad \langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0;\tau\rangle, \qquad \langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle.$$

Take

$$\mathcal{D} = \mathrm{ActCtx}_\eta(\mathcal{D}_0, x, M_2, m).$$

Applying the contextual dollar clause to $\langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0;\tau\rangle$ is justified by $(*)$, and the equality $P \equiv \mathcal{D}[N]$ follows from the definition of ActCtx. □

Lemma A.21. *Let V be a $\mathbf{DEL}_{\mathrm{one}}$-value.*

(1) If $\underline{V}_\eta \equiv ()$, then $V = ()$.
(2) If $\underline{V}_\eta \equiv (W_1, W_2)$, then $V = (V_1, V_2)$ for some V_1, V_2, with $W_i \equiv \underline{V_i}_\eta$.
(3) If $\underline{V}_\eta \equiv \mathbf{inj}_{\mathrm{L}}\ W$, then $V = \mathbf{inj}_{\mathrm{L}}\ V'$ for some V', with $W \equiv \underline{V'}_\eta$.
(4) If $\underline{V}_\eta \equiv \{P\}$, then $V = \{M\}$ for some M, with $P \equiv \underline{M}_\eta$.
(5) If $\underline{V}_\eta \equiv l$ for some coroutine label l, then $V = l'$ for some continuation label l' and $\eta(l') = (l, m)$ for some m.

Proof. By case analysis on the syntax of V. The translation is homomorphic on all ordinary constructors, and the only source values mapped to target labels are source continuation labels. □

Lemma A.22. *If* $\langle M;\theta\rangle \overset{\eta}{\sim} \langle \mathbf{return}\, W;\tau\rangle$*, then there exists a* $\mathbf{DEL}_{\mathrm{one}}$ *value* V *such that* $M = \mathbf{return}\, V$ *and* $W \equiv \underline{V}_{\eta}$.

Proof. By case analysis on the derivation of $\langle M;\theta\rangle \overset{\eta}{\sim} \langle \mathbf{return}\, W;\tau\rangle$. The only rule where the outermost constructor of the target **AC** computation is **return** is the structural clause for **return**. □

Lemma A.23. *Suppose* $\mathcal{E}$ *is a pure* **AC** *context and*

$$\langle M;\theta\rangle \overset{\eta}{\sim} \langle \mathcal{E}[\mathbf{yield}\,\{P\}];\tau\rangle.$$

Then there exist a pure $\mathbf{DEL}_{\mathrm{one}}$ *context* $\mathcal{H}$*, a variable* k*, and a source computation* M_1 *such that*

$$M = \mathcal{H}[\mathbf{S_0}k.\ M_1], \qquad \mathcal{E} \equiv \underline{\mathcal{H}}_{\eta}, \qquad P \equiv \lambda x.\ \mathbf{let}\ k = x!\ \mathbf{in}\ \underline{M_1}_{\eta}.$$

Proof. We prove the lemma by induction on the derivation of $\langle M;\theta\rangle \overset{\eta}{\sim} \langle \mathcal{E}[\mathbf{yield}\,\{P\}];\tau\rangle$. If the last rule is the structural clause for $\mathbf{S_0}k.\ M_1$, then take $\mathcal{H} = [\ ]$. The target computation is exactly $\mathbf{yield}\left\{\lambda x.\ \mathbf{let}\ k = x!\ \mathbf{in}\ \underline{M_1}_{\eta}\right\}$.

The structural clauses for product matching, variant matching, force, return, abstraction, lazy pairing, and throw are impossible because their target computations cannot be written as a pure-context plugged with a yield term.

In the let case, the target has the form $\mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_{\eta}$. Since the whole target computation is $\mathcal{E}[\mathbf{yield}\,\{P\}]$ with E pure, the occurrence of **yield** must lie in N_1. Apply the induction hypothesis to the premise relating the distinguished source subcomputation to N_1, and then extend the source pure context and the target pure context by the same let-frame. The application and projection cases are identical.

The dollar clause is impossible. Its target has the form $\mathrm{Act}_{\eta}(N_1, x, M_2, m)$, which contains an active labeled computation and therefore cannot be written as a pure context plugged with a yield term. □

A.2.2 Canonicalization

Proposition A.24 (canonicalization). *Assume* $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M;\theta\rangle)$*,* $\mathsf{WF}_{\mathbf{AC}}(\left\langle \underline{M}_{\eta};\tau\right\rangle)$*,* $\mathsf{Coh}(\theta,\eta)$*, and* $\mathsf{Inv}(\theta,\tau,\eta)$*. Then, there exist an* **AC** *computation* N *and an* **AC** *store* τ' *such that*

$$\left\langle \underline{M}_{\eta};\tau\right\rangle \to^{*}_{\mathbf{AC}} \langle N;\tau'\rangle$$

and

$$\langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau'\rangle, \qquad \mathsf{WF}_{\mathbf{AC}}(\langle N;\tau'\rangle), \qquad \mathsf{Inv}(\theta,\tau',\eta),$$

i.e., $\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau'\rangle$.

Proof. By induction on M. In the cases where M has one of the forms covered by the structural clauses of $\overset{\eta}{\sim}$, take $N \equiv \underline{M}_{\eta}$ and $\tau' = \tau$. If $M = \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2$, $M = M_1\, V$, or $M = \mathbf{prj}_i\, M_1$, apply the induction hypothesis to M_1 and then rebuild the constructor by the corresponding structural clause of $\overset{\eta}{\sim}$.

Finally, suppose $M = \langle M_1 | x.M_2\rangle$. The translation $\underline{\langle M_1 | x.M_2\rangle}_\eta$ evaluates to

$$N \stackrel{\text{def}}{=} \left(\begin{array}{l}\textbf{let } \mathit{res} = m : \left(\textbf{let } x = \underline{M_1}_\eta \textbf{ in return } \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\right) \textbf{ in} \\ \mathit{res}!\ \left\{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\ m\right\}\end{array}\right),$$

where m is a fresh coroutine label mapped to **nil** in the store. Let τ' be $\tau[m := \textbf{nil}]$. It is straightforward to check $\mathsf{Inv}(\theta, \tau', \eta)$ holds. Apply the induction hypothesis to M_1. Then, we obtain

$$\left\langle \underline{M_1}_\eta; \tau' \right\rangle \to^*_{\textbf{AC}} \langle N_1; \tau''\rangle$$

and

$$\langle M_1; \theta\rangle \overset{\eta}{\sim} \langle N_1; \tau''\rangle, \qquad \mathsf{WF}_{\textbf{AC}}(\langle N_1; \tau''\rangle), \qquad \mathsf{Inv}(\theta, \tau'', \eta),$$

for some **AC** computation N_1 and **AC** store τ''.

To obtain

$$\langle M; \theta\rangle \overset{\eta}{\sim}_\sharp \left\langle \left(\begin{array}{l}\textbf{let } \mathit{res} = m : \left(\textbf{let } x = N_1 \textbf{ in return } \left\{\lambda\_.\ \underline{M_2}_\eta\right\}\right) \textbf{ in} \\ \mathit{res}!\ \left\{\mathit{ref}!\ \textbf{inj}_{\text{Valid}}\ m\right\}\end{array}\right); \tau'' \right\rangle,$$

it suffices to check the two side conditions of the dollar clause. The condition $\tau''(m) = \textbf{nil}$ holds because m is active in every configuration occurring in the reduction of $\underline{M_1}_\eta$ under the frame $m : [\ ]$, because that reduction preserves well-formedness by Proposition A.5, and because the third clause of **AC** well-formedness forces the content of an active label to be **nil**. The condition $\forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \textbf{nil}$ holds vacuously, since m is fresh and thus occurs in no value of η. □

Corollary A.25 (initial simulation). *For every* $\textbf{DEL}_{\text{one}}$ *program M,*

$$\langle M; \emptyset\rangle \sim \left\langle \underline{M}; \emptyset\right\rangle.$$

Proof. $\langle M; \emptyset\rangle$ is trivially well-formed, and the empty label map satisfies Coh and Inv. Checking the well-formedness of $\left\langle \underline{M}; \emptyset\right\rangle$ is straightforward since the computation of the form $l : M$ is not in the image of the translation and so $AL(\underline{M}) = \emptyset$. Then, apply Proposition A.24. □

A.2.3 Simulation

Proposition A.26 (simulation of beta-reduction). *Assume* $\langle M; \theta\rangle \overset{\eta}{\sim}_\sharp \langle N; \tau\rangle$. *Then:*

(1) If $\langle M; \theta\rangle \to^\beta_{\textbf{D}} \bot$, *then*

$$\langle N; \tau\rangle \to^+_{\textbf{AC}} \bot.$$

(2) If $\langle M; \theta\rangle \to^\beta_{\textbf{D}} \langle M'; \theta'\rangle$ *then there exist* N', τ', η' *such that*

$$\langle N; \tau\rangle \to^+_{\textbf{AC}} \langle N'; \tau'\rangle \quad \textit{and} \quad \langle M'; \theta'\rangle \overset{\eta'}{\sim}_\sharp \langle N'; \tau'\rangle.$$

Proof. We prove by case analysis on the beta-reduction rule used in the source.

Case (MAM). These cases are routine. We spell out one representative case. If $M \equiv (\textbf{let}\ x = \textbf{return}\ V\ \textbf{in}\ M_0)$, then the target must be

$$N \equiv (\textbf{let}\ x = \textbf{return}\ \underline{V}_{\eta}\ \textbf{in}\ \underline{M_0}_{\eta}).$$

Reducing this computation under τ yields

$$\left\langle \underline{M_0}_{\eta}[\underline{V}_{\eta}/x]; \tau \right\rangle = \left\langle \underline{M_0[V/x]}_{\eta}; \tau \right\rangle$$

by Lemma A.16. Then, canonicalize this by Proposition A.24 to obtain the desired conclusion.

Case (ret)**.** Similar to the **MAM** case.

Case (throw)**.** Suppose $M = \textbf{throw}\ l\ V$ and $\theta(l) = \lambda y.\ \langle \mathcal{H}[\textbf{return}\ y] \,|\, x.M_0 \rangle$. Then

$$\langle M; \theta \rangle \to_{\mathbf{D}} \langle \langle \mathcal{H}[\textbf{return}\ V] \,|\, x.M_0 \rangle; \theta' \rangle, \qquad \theta' \stackrel{\text{def}}{=} \theta[l := \textbf{nil}].$$

Since $\langle M; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle$, the last rule used to derive the core relation is the structural clause for **throw**. Hence $N \equiv \underline{\textbf{throw}\ l\ V}_{\eta}$. By (C1), there are target labels mc and m such that $\eta(l) = (mc, m)$. By (IC2),

$$\tau(mc) = \mathit{RefCell}(\textbf{inj}_{\text{Valid}}\ m), \qquad \tau(m) = \text{Cont}_{\eta}(H, x, M_0),$$

and, if $l' \neq l$ and $\eta(l') = (mc', m)$, then $\theta(l') = \textbf{nil}$. Put

$$\tau' \stackrel{\text{def}}{=} \tau[mc := \mathit{RefCell}(\textbf{inj}_{\text{Invalid}}\ ()),\ m := \textbf{nil}].$$

By the definition of the translation of **throw**, Lemma A.17, and the **AC** rule for **resume**, we have

$$\left\langle \underline{\textbf{throw}\ l\ V}_{\eta}; \tau \right\rangle \to^{+}_{\mathbf{AC}} \left\langle \text{Act}_{\eta}(\underline{\mathcal{H}[\textbf{return}\ V]}_{\eta}, x, M_0, m); \tau' \right\rangle.$$

In the last step, the resumed coroutine m runs $\text{Cont}_{\eta}(H, x, M_0)!\,\underline{V}_{\eta}$, and Lemma A.16 identifies the body of the coroutine with

$$\textbf{let}\ x = \underline{\mathcal{H}[\textbf{return}\ V]}_{\eta}\ \textbf{in return}\ \left\{ \lambda\_.\ \underline{M_0}_{\eta} \right\}.$$

We next check the hypotheses of Proposition A.24 for the source reduct $\mathcal{H}[\textbf{return}\ V]$, the stores θ' and τ', and the label map η; the well-formedness conditions are immediate. First, $\mathsf{Coh}(\theta', \eta)$ holds because $\text{Dom}(\theta') = \text{Dom}(\theta)$, so (C1) is preserved and (C2) is unaffected. Next, $\mathsf{Inv}(\theta', \tau', \eta)$ holds as follows. The label l is now invalid and its reference cell mc has the invalid tag, since $\theta'(l) = \textbf{nil}$ and $\tau'(mc) = \mathit{RefCell}(\textbf{inj}_{\text{Invalid}}\ ())$. All other invalid labels in θ' keep their old invalid cells. Thus (IC1) for $\mathsf{Inv}(\theta', \tau', \eta)$ holds. If l' is valid in θ', then $l' \neq l$, and by (C2), the first component of $\eta(l')$ is not mc. Moreover its associated coroutine cannot be m; if so, (IC2-c) for the old valid label l would imply $\theta(l') = \textbf{nil}$. Thus the store contents relevant to l' are unchanged from τ, and (IC2) follows from $\mathsf{Inv}(\theta, \tau, \eta)$.

By applying Proposition A.24 with M taken as $\mathcal{H}[\textbf{return}\ V]$, we obtain an **AC** computation N' and an **AC** store τ'' such that

$$\left\langle \underline{\mathcal{H}[\textbf{return}\ V]}_{\eta}; \tau' \right\rangle \to^{*}_{\mathbf{AC}} \langle N'; \tau'' \rangle \quad \text{and} \quad \langle \mathcal{H}[\textbf{return}\ V]; \theta' \rangle \overset{\eta}{\sim}_{\sharp} \langle N'; \tau'' \rangle.$$

This reduction takes place in the body of the active coroutine m, so it lifts to

$$\left\langle \text{Act}_{\eta}(\underline{\mathcal{H}[\textbf{return}\ V]}_{\eta}, x, M_0, m); \tau' \right\rangle \to^{*}_{\mathbf{AC}} \left\langle \text{Act}_{\eta}(N', x, M_0, m); \tau'' \right\rangle,$$

and hence $\langle N; \tau \rangle \to^{+}_{\mathbf{AC}} \left\langle \text{Act}_{\eta}(N', x, M_0, m); \tau'' \right\rangle$.

It remains to show $\langle\langle\mathcal{H}[\mathbf{return}\,V]\,|\,x.M_0\rangle;\theta'\rangle \overset{\eta}{\sim}_\sharp \left\langle \mathrm{Act}_\eta(N',x,M_0,m);\tau''\right\rangle$. The two well-formedness conditions, $\mathsf{Coh}(\theta',\eta)$, and $\mathsf{Inv}(\theta',\tau'',\eta)$ are all supplied by $\langle\mathcal{H}[\mathbf{return}\,V];\theta'\rangle \overset{\eta}{\sim}_\sharp \langle N';\tau''\rangle$ obtained above; note that it is τ'', not τ', for which the invariant conditions are thereby established. For the core relation, we apply the dollar clause to $\langle\mathcal{H}[\mathbf{return}\,V];\theta'\rangle \overset{\eta}{\sim} \langle N';\tau''\rangle$, so it suffices to check its two side conditions

$$\tau''(m) = \mathbf{nil} \quad \text{and} \quad \forall l', mc'.\ \eta(l') = (mc',m) \Rightarrow \theta'(l') = \mathbf{nil}.$$

The former holds as follows. The label m is active in every configuration occurring in the lifted reduction above, since each step of that reduction takes place inside the frame $m : [\,]$. Moreover, the first of these configurations is well-formed, because $\tau'(m) = \mathbf{nil}$ by the definition of τ', and hence so are all the others by Proposition A.5. The third clause of **AC** well-formedness thus gives $\tau''(m) = \mathbf{nil}$. As for the latter, (IC2-c) of $\mathsf{Inv}(\theta,\tau,\eta)$ tells us that the label l is the only valid continuation label defined in θ that is associated with the coroutine m. This label is no longer valid in θ', which implies that the condition above is satisfied.

Case (shift)**.** Suppose

$$M \equiv \langle\mathcal{H}[\mathbf{S_0}k.\ M_1]\,|\,x.M_2\rangle,$$

where H is a $\mathbf{DEL}_{\mathrm{one}}$ pure context. Let l be the fresh continuation label generated by the source reduction, and put

$$\theta' \overset{\mathrm{def}}{=} \theta[l := \lambda y.\ \langle\mathcal{H}[\mathbf{return}\,y]\,|\,x.M_2\rangle].$$

Then

$$\langle\langle\mathcal{H}[\mathbf{S_0}k.\ M_1]\,|\,x.M_2\rangle;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M_1[l/k];\theta'\rangle.$$

Since $\langle M;\theta\rangle \overset{\eta}{\sim}_\sharp \langle N;\tau\rangle$, the last rule used to derive the core relation is the structural rule for the dollar clause. Hence there exist an **AC** computation N_0 and a coroutine label m such that

$$N \equiv \mathrm{Act}_\eta(N_0,x,M_2,m),$$

$$\langle\mathcal{H}[\mathbf{S_0}k.\ M_1];\theta\rangle \overset{\eta}{\sim} \langle N_0;\tau\rangle, \qquad \tau(m) = \mathbf{nil},$$

and

$$\forall l_0, mc_0.\ \eta(l_0) = (mc_0,m) \Rightarrow \theta(l_0) = \mathbf{nil}.$$

By Lemma A.20 applied to the pure context H, together with Lemma A.18, we may write

$$N_0 \equiv \underline{\mathcal{H}}_\eta\left[\mathbf{yield}\left\{\lambda u.\ \mathbf{let}\ k = u!\ \mathbf{in}\ \underline{M_1}_\eta\right\}\right],$$

where u is chosen fresh.

We now compute the corresponding target reduction. In the active coroutine m, (yield) occurs under the pure target context

$$\mathbf{let}\ x = \underline{\mathcal{H}}_\eta\ \mathbf{in\ return}\left\{\lambda\_.\ \underline{M_2}_\eta\right\}.$$

Therefore,

$$\langle N;\tau\rangle \to^{+}_{\mathbf{AC}} \left\langle\left\{\lambda u.\ \mathbf{let}\ k = u!\ \mathbf{in}\ \underline{M_1}_\eta\right\}!\ \{ref!\,\mathbf{inj}_{\mathrm{Valid}}\,m\};\tau[m := \mathrm{Cont}_\eta(H,x,M_2)]\right\rangle.$$

Reducing the force of the yielded thunk gives

$$\left\langle\mathbf{let}\ k = \{ref!\,\mathbf{inj}_{\mathrm{Valid}}\,m\}!\ \mathbf{in}\ \underline{M_1}_\eta;\tau[m := \mathrm{Cont}_\eta(H,x,M_2)]\right\rangle.$$

By Lemma A.17, for a fresh coroutine label mc, we have

$$\begin{aligned}&\left\langle\left\{ref\,!\,\mathbf{inj}_{\mathrm{Valid}}\,m\right\}!;\tau[m := \mathrm{Cont}_\eta(H,x,M_2)]\right\rangle\\ &\to^+_{\mathbf{AC}} \left\langle\mathbf{return}\,mc;\tau[m := \mathrm{Cont}_\eta(H,x,M_2),\ mc := \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\,m)]\right\rangle.\end{aligned}$$

Hence the whole target computation reduces to

$$\left\langle\underline{M_1}_\eta[mc/k];\tau[m := \mathrm{Cont}_\eta(H,x,M_2),\ mc := \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\,m)]\right\rangle.$$

Define

$$\eta' \stackrel{\mathrm{def}}{=} \eta[l := (mc,m)]$$

and

$$\tau' \stackrel{\mathrm{def}}{=} \tau[m := \mathrm{Cont}_\eta(H,x,M_2),\ mc := \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\,m)].$$

Since l is fresh, it does not occur in M_1, H, or M_2. Therefore Lemma A.16 gives

$$\underline{M_1}_\eta[mc/k] \equiv \underline{M_1[l/k]}_{\eta'}.$$

Moreover,

$$\mathrm{Cont}_\eta(H,x,M_2) \equiv \mathrm{Cont}_{\eta'}(H,x,M_2),$$

again because l is fresh for H and M_2.

We verify the side conditions for

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

First, $\mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle M_1[l/k];\theta'\rangle)$ follows from the freshness of l and from preservation of $\mathbf{DEL}_{\mathrm{one}}$-well-formedness. Next, $\mathsf{Coh}(\theta',\eta')$ holds: (C1) because both θ and η are extended by the same fresh label l, and (C2) because the new first component mc is fresh. We now check $\mathsf{Inv}(\theta',\tau',\eta')$. For the new label l, we have

$$\theta'(l) = \lambda y.\ \langle\mathcal{H}[\mathbf{return}\,y] \mid x.M_2\rangle, \qquad \eta'(l) = (mc,m),$$

and by definition of τ',

$$\tau'(mc) = \mathit{RefCell}(\mathbf{inj}_{\mathrm{Valid}}\,m), \qquad \tau'(m) = \mathrm{Cont}_\eta(H,x,M_2) = \mathrm{Cont}_{\eta'}(H,x,M_2).$$

Thus (IC2-a) and (IC2-b) hold for l. If $l_0 \neq l$ and $\eta'(l_0) = (mc_0,m)$, then $l_0 \in \mathrm{Dom}(\eta)$. The side condition of the outer dollar clause gives $\theta(l_0) = \mathbf{nil}$, hence $\theta'(l_0) = \mathbf{nil}$. Therefore (IC2-c) also holds for the new valid label l.

For an old label $l_0 \in \mathrm{Dom}(\eta)$, the only store entries changed from τ to τ' are those at m and at the fresh label mc. If l_0 is invalid, then (IC1) remains true because its first component is not mc, by freshness, and its old cell is unchanged. If l_0 is valid, then its associated coroutine cannot be m: otherwise the side condition

$$\forall l_0, mc_0.\ \eta(l_0) = (mc_0,m) \Rightarrow \theta(l_0) = \mathbf{nil}$$

would force $\theta(l_0) = \mathbf{nil}$, a contradiction. Hence the target entries relevant to such an old valid label are unchanged, and (IC2) follows from $\mathsf{Inv}(\theta,\tau,\eta)$. Thus $\mathsf{Inv}(\theta',\tau',\eta')$ holds.

Finally, apply Proposition A.24 to $M_1[l/k]$, θ', τ', and η'. We obtain some **AC** computation N' and **AC** store τ'' such that

$$\left\langle\underline{M_1[l/k]}_{\eta'};\tau'\right\rangle \to^*_{\mathbf{AC}} \langle N';\tau''\rangle$$

and

$$\langle M_1[l/k];\theta'\rangle \stackrel{\eta'}{\sim}_\sharp \langle N';\tau''\rangle.$$

Combining the reductions above gives

$$\langle N;\tau\rangle \to^{+}_{\mathbf{AC}} \langle N';\tau''\rangle.$$

Case (fail)**.** Suppose $M = \mathbf{throw}\, l\, V$ and $\theta(l) = \mathbf{nil}$. Then $\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \bot$. As in the previous case, the core relation must be the structural clause for **throw**, so $N \equiv \underline{\mathbf{throw}\, l\, V}_{\eta}$. By (C1), write $\eta(l) = (mc, m)$. Since $\theta(l) = \mathbf{nil}$, (IC1) gives

$$\tau(mc) = \mathit{RefCell}(\mathbf{inj}_{\mathrm{Invalid}}\,()).$$

Therefore, by the definition of the translation and Lemma A.17,

$$\left\langle \underline{\mathbf{throw}\, l\, V}_{\eta};\tau\right\rangle \to^{+}_{\mathbf{AC}} \langle \mathit{fail}!;\tau\rangle \to^{+}_{\mathbf{AC}} \bot.$$

□

Proposition A.27. *Assume*

$$\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle, \qquad \langle C;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle, \qquad \mathsf{WF}_{\mathbf{DEL}_{\mathrm{one}}}(\langle C[M];\theta\rangle),$$

$$\mathsf{WF}_{\mathbf{AC}}(\langle \mathcal{D}[N];\tau\rangle).$$

Then,

(1) *If* $\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \bot$, *then*

$$\langle \mathcal{D}[N];\tau\rangle \to^{+}_{\mathbf{AC}} \bot.$$

(2) *If* $\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M';\theta'\rangle$ *then there exist* N', τ', η' *such that*

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

Proof. We prove the proposition by induction on the derivation of

$$\langle C;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D};\tau\rangle.$$

If $C = [\,]$ and $\mathcal{D} = [\,]$, the result is exactly Proposition 3.9. Indeed, in the non-error case, Proposition 3.9 gives

$$\langle N;\tau\rangle \to^{+}_{\mathbf{AC}} \langle N';\tau'\rangle \qquad \text{and} \qquad \langle M';\theta'\rangle \overset{\eta'}{\sim}_{\sharp} \langle N';\tau'\rangle.$$

The contextual relation

$$\langle [\,];\theta'\rangle \overset{\eta'}{\sim}_{\mathsf{c}} \langle [\,];\tau'\rangle$$

follows from the empty-context clause.

If the last rule of the derivation of the contextual relation is a pure-frame rule, the induction hypothesis gives the desired target reduction and the desired relation for the immediate subcontext. We then rebuild the same pure frame by the corresponding clause of the contextual core relation and by the corresponding clause of the core relation on computations. The source well-formedness of the rebuilt configuration follows from preservation of $\mathbf{DEL}_{\mathrm{one}}$-well-formedness, and the target well-formedness follows from preservation of **AC**-well-formedness along the target reduction. The conditions Coh and Inv are supplied by the induction hypothesis and do not depend on the outer pure frame. This proves the pure-frame cases.

It remains to consider the dollar-frame case. Thus the last rule is of the form

$$\frac{\langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \langle \mathcal{D}_0;\tau\rangle \qquad \tau(m) = \mathbf{nil} \qquad \forall l, mc.\ \eta(l) = (mc, m) \Rightarrow \theta(l) = \mathbf{nil}}{\langle \langle C_0 \,|\, x.M_2\rangle;\theta\rangle \overset{\eta}{\sim}_{\mathsf{c}} \left\langle \mathrm{ActCtx}_{\eta}(\mathcal{D}_0, x, M_2, m);\tau\right\rangle}.$$

Therefore

$$C = \langle C_0 |\, x.M_2 \rangle, \qquad \mathcal{D} = \mathrm{ActCtx}_\eta(\mathcal{D}_0, x, M_2, m).$$

First suppose that

$$\langle M; \theta \rangle \to^\beta_{\mathbf{D}} \bot.$$

By the induction hypothesis applied to the premise

$$\langle C_0; \theta \rangle \overset{\eta}{\sim}_{\mathsf{C}} \langle \mathcal{D}_0; \tau \rangle,$$

we get

$$\langle \mathcal{D}_0[N]; \tau \rangle \to^+_{\mathbf{AC}} \bot.$$

Since

$$\mathcal{D}[N] = (\mathrm{ActCtx}_\eta(\mathcal{D}_0, x, M_2, m))[N] = \mathrm{Act}_\eta(\mathcal{D}_0[N], x, M_2, m),$$

contextual closure of $\to_{\mathbf{AC}}$ yields

$$\langle \mathcal{D}[N]; \tau \rangle \to^+_{\mathbf{AC}} \bot.$$

This proves the error part in the dollar-frame case.

Now suppose that

$$\langle M; \theta \rangle \to^\beta_{\mathbf{D}} \langle M'; \theta' \rangle.$$

Applying the induction hypothesis to the premise

$$\langle C_0; \theta \rangle \overset{\eta}{\sim}_{\mathsf{C}} \langle \mathcal{D}_0; \tau \rangle$$

gives target data N', τ', and η' such that

$$\langle N; \tau \rangle \to^+_{\mathbf{AC}} \langle N'; \tau' \rangle,$$

$$\langle C_0[M']; \theta' \rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}_0[N']; \tau' \rangle,$$

and

$$\langle C_0; \theta' \rangle \overset{\eta'}{\sim}_{\mathsf{C}} \langle \mathcal{D}_0; \tau' \rangle.$$

We have to rebuild the outer dollar frame. For this, it suffices to prove that the two side conditions of the dollar clause are still valid:

$$\tau'(m) = \mathbf{nil}$$

and

$$\forall l, mc.\ \eta'(l) = (mc, m) \Rightarrow \theta'(l) = \mathbf{nil}.$$

We verify $(*)$ and $(**)$ by inspecting the source beta reduction rule used in

$$\langle M; \theta \rangle \to^\beta_{\mathbf{D}} \langle M'; \theta' \rangle.$$

For the **MAM** cases and for the (ret) case, the label map is not extended with any source continuation label associated with the outer coroutine m, and no store entry associated with m is modified. Hence both $(*)$ and $(**)$ follow immediately from the premises of the outer dollar frame.

Consider the (throw) case. Let the valid label used by the throw be l_0, and write

$$\eta(l_0) = (mc_0, m_0).$$

In the target reduction, mc_0 is invalidated, and the coroutine m_0 is set to **nil**. We claim that $m_0 \neq m$. If $m_0 = m$, then the side condition of the outer dollar frame would imply

$$\theta(l_0) = \mathbf{nil},$$

contradicting the assumption of the (throw) rule. Hence the throw does not change the store entry at m, and it does not create any new label mapped to m. Therefore $(*)$ and $(**)$ are preserved.

Finally consider the (shift) case. In this case the local redex has the form

$$\langle \mathcal{H}[\mathbf{S_0}k.\ M_1] \,|\, y.M_0 \rangle,$$

and the proof of Proposition 3.9 introduces a fresh label l_{new} and extends the label map by

$$\eta'(l_{\mathrm{new}}) = (mc_{\mathrm{new}}, m_0),$$

where m_0 is the coroutine label used by the inner dollar clause for this redex. We claim that $m_0 \neq m$. Indeed, if $m_0 = m$, then the target configuration $\langle \mathcal{D}[N]; \tau \rangle$ would contain two active occurrences of the same coroutine label m: one from the outer dollar frame and one from the inner dollar frame. This contradicts the assumed well-formedness

$$\mathsf{WF}_{\mathbf{AC}}(\langle \mathcal{D}[N]; \tau \rangle).$$

Thus $m_0 \neq m$. Since the newly added source label is associated with m_0, not with m, condition $(**)$ is not affected by the fresh extension of η. Moreover, the store update performed by the shift changes the entry at m_0, not at m. Hence $(*)$ is also preserved.

Therefore, we can now rebuild the contextual relation for the outer dollar frame:

$$\langle \langle C_0 \,|\, x.M_2 \rangle; \theta' \rangle \overset{\eta'}{\sim}_{\mathsf{c}} \left\langle \mathrm{ActCtx}_{\eta'}(\mathcal{D}_0, x, M_2, m); \tau' \right\rangle.$$

Since the extension from η to η' is on labels fresh for the outer context and for M_2, we have

$$\underline{M_2}_{\eta'} \equiv \underline{M_2}_{\eta}.$$

Hence the target context above is the same target context $\mathcal{D}$, now considered with the updated label map and store.

From the induction hypothesis we also have

$$\langle C_0[M']; \theta' \rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}_0[N']; \tau' \rangle.$$

In particular,

$$\langle C_0[M']; \theta' \rangle \overset{\eta'}{\sim} \langle \mathcal{D}_0[N']; \tau' \rangle.$$

Using the dollar clause of the core relation together with $(*)$ and $(**)$, we obtain

$$\langle \langle C_0[M'] \,|\, x.M_2 \rangle; \theta' \rangle \overset{\eta'}{\sim} \left\langle \mathrm{Act}_{\eta'}(\mathcal{D}_0[N'], x, M_2, m); \tau' \right\rangle.$$

This is exactly

$$\langle C[M']; \theta' \rangle \overset{\eta'}{\sim} \langle \mathcal{D}[N']; \tau' \rangle.$$

Finally, the well-formedness conditions required for

$$\langle C[M']; \theta' \rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}[N']; \tau' \rangle$$

follow from preservation of source and target well-formedness, while $\mathsf{Coh}(\theta', \eta')$ and $\mathsf{Inv}(\theta', \tau', \eta')$ are inherited from the induction hypothesis applied to

$$\langle \langle C_0 \,|\, x.M_2 \rangle; \theta' \rangle \overset{\eta'}{\sim}_{\mathsf{c}} \left\langle \mathrm{ActCtx}_{\eta'}(\mathcal{D}_0, x, M_2, m); \tau' \right\rangle.$$

Therefore,

$$\langle C[M'];\theta'\rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}[N'];\tau'\rangle$$

holds, as required. □

Theorem A.28 (simulation). *If $C \sim D$ and $C \to_{\mathbf{D}} C'$, then there exists an* **AC** *configuration D' such that*

$$D \to^{+}_{\mathbf{AC}} D' \quad \textit{and} \quad C' \sim D'.$$

Proof. Since $C \to_{\mathbf{D}} C'$, the configuration C is not $\perp$. By the definition of $\sim$, there exist an **AC** computation P, an **AC** store τ, and a label map η such that

$$D \to^{*}_{\mathbf{AC}} \langle P;\tau\rangle \quad \text{and} \quad C \overset{\eta}{\sim}_{\sharp} \langle P;\tau\rangle.$$

Write $C = \langle M_0;\theta\rangle$. By the definition of $\to_{\mathbf{D}}$, the source step decomposes into an evaluation context and a beta-reduction step. Thus, there exist a $\mathbf{DEL}_{\text{one}}$ evaluation context C_0 and a $\mathbf{DEL}_{\text{one}}$ computation M such that

$$M_0 \equiv C_0[M]$$

and either

$$\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \perp \quad \text{and} \quad C' = \perp,$$

or there exist a $\mathbf{DEL}_{\text{one}}$ computation M' and a $\mathbf{DEL}_{\text{one}}$ store θ' such that

$$\langle M;\theta\rangle \to^{\beta}_{\mathbf{D}} \langle M';\theta'\rangle \quad \text{and} \quad C' = \langle C_0[M'];\theta'\rangle.$$

From

$$\langle C_0[M];\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle P;\tau\rangle$$

we obtain in particular

$$\langle C_0[M];\theta\rangle \overset{\eta}{\sim} \langle P;\tau\rangle.$$

By Lemma A.20, there exist an **AC** evaluation context $\mathcal{D}_0$ and an **AC** computation N such that

$$P \equiv \mathcal{D}_0[N],$$
$$\langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathrm{c}} \langle \mathcal{D}_0;\tau\rangle, \qquad \langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle.$$

The side conditions contained in $\langle C_0[M];\theta\rangle \sim_{\sharp} \langle \mathcal{D}_0[N];\tau\rangle$ restrict to the subconfiguration; thus,

$$\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle.$$

Moreover, the well-formedness assumptions required by Proposition 3.10 follow from $\langle C_0[M];\theta\rangle \sim_{\sharp} \langle \mathcal{D}_0[N];\tau\rangle$. We now apply Proposition 3.10 to

$$\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle \quad \text{and} \quad \langle C_0;\theta\rangle \overset{\eta}{\sim}_{\mathrm{c}} \langle \mathcal{D}_0;\tau\rangle.$$

First suppose

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

Proposition 3.10(1) gives

$$\langle P;\tau\rangle = \langle \mathcal{D}_0[N];\tau\rangle \to^{+}_{\mathbf{AC}} \perp.$$

Combining this with $D \to^{*}_{\mathbf{AC}} \langle P;\tau\rangle$ yields

$$D \to^{+}_{\mathbf{AC}} \perp.$$

As $C' = \perp$, we have $C' \sim \perp$ by the definition of the simulation relation. Thus we may take $D' = \perp$.

Next suppose

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

Proposition 3.10(2) yields an **AC** computation N', an **AC** store τ', and a label map η' such that

$$\langle N;\tau\rangle \rightarrow^{+}_{\mathbf{AC}} \langle N';\tau'\rangle,$$

and

$$\langle \mathcal{C}_0[M'];\theta'\rangle \overset{\eta'}{\sim}_{\sharp} \langle \mathcal{D}_0[N'];\tau'\rangle.$$

By the definition of $\rightarrow_{\mathbf{AC}}$, we obtain

$$\langle \mathcal{D}_0[N];\tau\rangle \rightarrow^{+}_{\mathbf{AC}} \langle \mathcal{D}_0[N'];\tau'\rangle.$$

Since $P \equiv \mathcal{D}_0[N]$, composing with $D \rightarrow^{*}_{\mathbf{AC}} \langle P;\tau\rangle$ gives

$$D \rightarrow^{+}_{\mathbf{AC}} \langle \mathcal{D}_0[N'];\tau'\rangle.$$

Also, because

$$C' = \langle \mathcal{C}_0[M'];\theta'\rangle,$$

we obtain

$$C' \sim \langle \mathcal{D}_0[N'];\tau'\rangle$$

by the definition of the simulation relation, with no target administrative step. □

A.2.4 Non-termination preservation

The next definition collects the shapes that an evaluation frame can consume.

Definition A.29 (head-shape matching). For a $\mathbf{DEL}_{\text{one}}$ computation M and an **AC** computation N, we write $M \rhd N$ if all of the following hold:

- (H1) if $N \equiv \mathbf{return}\, W$, then $M \equiv \mathbf{return}\, V$ for some $\mathbf{DEL}_{\text{one}}$ value V;
- (H2) if $N \equiv \lambda x.\ N_0$, then $M \equiv \lambda x.\ M_0$ for some $\mathbf{DEL}_{\text{one}}$ computation M_0;
- (H3) if $N \equiv \langle N_1, N_2\rangle$, then $M \equiv \langle M_1, M_2\rangle$ for some M_1 and M_2;
- (H4) if $N \equiv \mathcal{E}[\mathbf{yield}\, W]$ for a pure **AC** context $\mathcal{E}$, then $M \equiv \mathcal{H}[\mathbf{S_0} k.\ M_0]$ for a pure $\mathbf{DEL}_{\text{one}}$ context $\mathcal{H}$, a variable k, and a computation M_0.

Conditions (H1) and (H4) are the conclusions of Lemma A.22 and Lemma A.23; conditions (H2) and (H3) are their analogues for abstractions and lazy pairs.

We say a $\mathbf{DEL}_{\text{one}}$ or an **AC** configuration C is *stuck* if C is neither $\perp$ nor a return configuration, and no reduction rule is applicable to C. We also say C is *irreducible* if no reduction rule is applicable to C; thus a stuck configuration is irreducible, while a return configuration is irreducible but not stuck.

Lemma A.30. *Assume $\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle$ and that $\langle M;\theta\rangle$ is irreducible. Then there exist an* **AC** *computation N' and an* **AC** *store τ' such that*

$$\langle N;\tau\rangle \rightarrow^{*}_{\mathbf{AC}} \langle N';\tau'\rangle,$$

the configuration $\langle N';\tau'\rangle$ is irreducible, and $M \rhd N'$.

Proof. We prove the statement by induction on the derivation of

$$\langle M;\theta\rangle \overset{\eta}{\sim} \langle N;\tau\rangle.$$

Since $\langle M;\theta\rangle \overset{\eta}{\sim}_{\sharp} \langle N;\tau\rangle$ holds, $\mathsf{WF}_{\mathbf{DEL}_{\text{one}}}(\langle M;\theta\rangle)$ and $\mathsf{Inv}(\theta,\tau,\eta)$ hold.

We consider the clauses where the target term is exactly the runtime translation of the source term. In all of them the store is unchanged, so we take $\tau' = \tau$; and we take $N' \equiv N$ except for the throw term, where the target does take administrative steps.

Suppose first that

$$M \equiv \mathbf{case}\ V\ \mathbf{of}\ (x_1, x_2) \mapsto M_0.$$

Then

$$N \equiv \mathbf{case}\ \underline{V}_\eta\ \mathbf{of}\ (x_1, x_2) \mapsto \underline{M_0}_\eta.$$

Since $\langle M; \theta \rangle$ is irreducible, the value V is not a pair. By Lemma A.21, $\underline{V}_\eta$ is not a pair either. Therefore the target product-matching term is not reducible. It also has no distinguished subcomputation to reduce, so $\langle N; \tau \rangle$ is irreducible. As the outermost constructor of N is a product matching, none of (H1) to (H4) has its premise satisfied, so $M \rhd N$ holds vacuously. The variant-matching case and the force case are similar.

In the cases

$$M \equiv \mathbf{return}\, V, \qquad M \equiv \lambda x.\ M_0, \qquad M \equiv \langle M_1, M_2 \rangle,$$

N is the runtime translation of M, which has the same outermost constructor. These computations have no beta-reduction rule and contain no reducible subcomputation, so their translations are irreducible as well. For $M \rhd N$, the premise of exactly one of (H1), (H2) and (H3) is satisfied, and its conclusion holds because M has that same outermost constructor; (H4) cannot happen.

If

$$M \equiv \mathbf{S_0} k.\ M_0,$$

then, by the translation in Figure 10,

$$N \equiv \mathbf{yield}\left\{\lambda u.\ \mathbf{let}\ k = u!\ \mathbf{in}\ \underline{M_0}_\eta\right\}$$

for a fresh variable u. A yield-redex exists only inside an active labeled computation. This term is not under an active label, so the target configuration is irreducible. Only the premise of (H4) is satisfied, with $\mathcal{E} = [\ ]$, and its conclusion holds with $\mathcal{H} = [\ ]$.

Finally suppose $M \equiv \mathbf{throw}\, V\, W$ (and so $N \equiv \underline{\mathbf{throw}\, V\, W}_\eta$). Since $\langle M; \theta \rangle$ is irreducible and source well-formedness holds, V cannot be a continuation label. Indeed, if $V = l$, then well-formedness gives $l \in \mathrm{Dom}(\theta)$, and hence either $\theta(l) = \mathbf{nil}$, in which case the source (fail) rule applies, or $\theta(l) \neq \mathbf{nil}$, in which case the source (throw) rule applies. Both contradict irreducibility.

By Lemma A.21, $\underline{V}_\eta$ is not a coroutine label. In the reduction of N, the fixed administrative redexes for *get* are executed first; after that, the computation has the form

$$N' \equiv \mathbf{let}\ zc = \mathbf{resume}\, \underline{V}_\eta\ (\mathbf{inj}_{\mathrm{Get}}\, ())\ \mathbf{in}\ N_0$$

for the variant matching N_0 of Figure 10; these steps leave the store unchanged. Since $\underline{V}_\eta$ is not a coroutine label, no **AC** rule can be applied to the distinguished subcomputation, and the (let) rule cannot be applied either, so $\langle N'; \tau \rangle$ is irreducible. Thus, N reduces in finitely many steps to an irreducible target configuration. Its outermost constructor is a let and its distinguished subcomputation is a **resume** term, so, as above, none of (H1) to (H4) has its premise satisfied and $M \rhd N'$ holds vacuously.

Then, we show the inductive step by case analysis on the last rule used to derive $\langle M; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle$.

Let case. Suppose the last rule is the let clause:

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

Since $\langle \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2;\theta\rangle$ is irreducible, the subconfiguration $\langle M_1;\theta\rangle$ is irreducible and M_1 is not of the form $\mathbf{return}\,V$. By the induction hypothesis, there are N_1' and τ' such that

$$\langle N_1;\tau\rangle \to^*_{\mathbf{AC}} \langle N_1';\tau'\rangle,$$

the configuration $\langle N_1';\tau'\rangle$ is irreducible, and $M_1 \rhd N_1'$. By (H1), the computation N_1' is not of the form $\mathbf{return}\,W$. Hence

$$\left\langle \mathbf{let}\ x = N_1\ \mathbf{in}\ \underline{M_2}_\eta;\tau\right\rangle \to^*_{\mathbf{AC}} \left\langle \mathbf{let}\ x = N_1'\ \mathbf{in}\ \underline{M_2}_\eta;\tau'\right\rangle.$$

The target configuration is irreducible: the subcomputation N_1' is not reducible, and the (let) rule cannot be applied since N_1' is not a return computation.

It remains to check $\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 \rhd \mathbf{let}\ x = N_1'\ \mathbf{in}\ \underline{M_2}_\eta$. The premises of (H1), (H2) and (H3) are not satisfied, since the outermost constructor of the target is a let. For (H4), suppose the target is $\mathcal{E}[\mathbf{yield}\,W]$ for a pure $\mathbf{AC}$ context $\mathcal{E}$. Then $\mathcal{E}$ is not $[\ ]$, so $\mathcal{E} = (\mathbf{let}\ x = \mathcal{E}_1\ \mathbf{in}\ \underline{M_2}_\eta)$ for a pure context $\mathcal{E}_1$ with $N_1' \equiv \mathcal{E}_1[\mathbf{yield}\,W]$. By (H4) for $M_1 \rhd N_1'$, we get $M_1 \equiv \mathcal{H}_1[\mathbf{S_0}k.\ M_0]$ for a pure $\mathbf{DEL}_{\mathrm{one}}$ context $\mathcal{H}_1$, and therefore $\mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 \equiv \mathcal{H}[\mathbf{S_0}k.\ M_0]$ for the pure context $\mathcal{H} = (\mathbf{let}\ x = \mathcal{H}_1\ \mathbf{in}\ M_2)$, as required.

Application case. Suppose the last rule is the application clause, so

$$M \equiv M_1\,V, \qquad N \equiv N_1\,\underline{V}_\eta,$$

with

$$\langle M_1;\theta\rangle \overset{\eta}{\sim} \langle N_1;\tau\rangle.$$

Since $\langle M_1\,V;\theta\rangle$ is irreducible, the subconfiguration $\langle M_1;\theta\rangle$ is irreducible and M_1 is not an abstraction. By the induction hypothesis, $\langle N_1;\tau\rangle$ reduces to an irreducible target configuration $\langle N_1';\tau'\rangle$ with $M_1 \rhd N_1'$. By (H2), this target computation N_1' is not an abstraction either. Therefore

$$\left\langle N_1\,\underline{V}_\eta;\tau\right\rangle \to^*_{\mathbf{AC}} \left\langle N_1'\,\underline{V}_\eta;\tau'\right\rangle,$$

and the target configuration is irreducible. The check of $M_1\,V \rhd N_1'\,\underline{V}_\eta$ is as in the let case, with the application frame in place of the let frame.

Projection case. Suppose the last rule is the projection clause. Then

$$M \equiv \mathbf{prj}_i\,M_1, \qquad N \equiv \mathbf{prj}_i\,N_1,$$

with

$$\langle M_1;\theta\rangle \overset{\eta}{\sim} \langle N_1;\tau\rangle.$$

Since the source projection is irreducible, $\langle M_1;\theta\rangle$ is irreducible and M_1 is not a lazy pair. By the induction hypothesis, $\langle N_1;\tau\rangle$ reduces to an irreducible target configuration $\langle N_1';\tau'\rangle$ with $M_1 \rhd N_1'$, and by (H3) the computation N_1' is not a lazy pair either. Hence

$$\langle \mathbf{prj}_i\,N_1;\tau\rangle \to^*_{\mathbf{AC}} \langle \mathbf{prj}_i\,N_1';\tau'\rangle,$$

and the target configuration is irreducible. The check of $\mathbf{prj}_i\,M_1 \rhd \mathbf{prj}_i\,N_1'$ is again as in the let case.

Dollar case. Suppose the last rule is the dollar clause. Thus

$$M \equiv \langle M_1 \,|\, x.M_2 \rangle, \qquad N \equiv \mathrm{Act}_\eta(N_1, x, M_2, m),$$

and

$$\langle M_1; \theta \rangle \overset{\eta}{\sim} \langle N_1; \tau \rangle, \qquad \tau(m) = \mathbf{nil},$$

together with the side condition that every source label mapped to the coroutine m is invalid.

Since $\langle \langle M_1 \,|\, x.M_2 \rangle; \theta \rangle$ is irreducible, M_1 cannot be reduced, cannot be a return computation, and cannot be of the form $\mathcal{H}[\mathbf{S_0}k.\ M_0]$ for any pure context H. These are exactly the three ways in which a dollar term can reduce: by reducing its body, by the ret rule, or by the shift rule.

By the induction hypothesis, $\langle N_1; \tau \rangle$ reduces to an irreducible target configuration $\left\langle N_1'; \tau' \right\rangle$ with $M_1 \rhd N_1'$. By (H1), the computation N_1' is not a return computation; by (H4), it is not of the form $\mathcal{E}[\mathbf{yield}\, W]$ for a pure target context $\mathcal{E}$.

Therefore

$$\left\langle \mathrm{Act}_\eta(N_1, x, M_2, m); \tau \right\rangle \to^*_{\mathbf{AC}} \left\langle \mathrm{Act}_\eta(N_1', x, M_2, m); \tau' \right\rangle.$$

The target configuration is irreducible: its distinguished subcomputation N_1' cannot step, the (F) rule for the let frame cannot be applied since N_1' is not a return computation, and the (yield) rule also cannot be applied since N_1' is not a pure-context occurrence of **yield**. Finally, $M \rhd \mathrm{Act}_\eta(N_1', x, M_2, m)$ holds vacuously: the outermost constructor of $\mathrm{Act}_\eta(N_1', x, M_2, m)$ is a let, and its distinguished subcomputation is the labeled computation $m : \ldots$, which is not a pure-context occurrence of **yield**. □

Proposition A.31. *Assume $\langle M; \theta \rangle \overset{\eta}{\sim}_\sharp \langle N; \tau \rangle$. If $\langle M; \theta \rangle$ is stuck, then $\langle N; \tau \rangle$ reduces to a stuck configuration, i.e., there exist an* **AC** *computation N' and an* **AC** *store τ' such that $\langle N; \tau \rangle \to^*_{\mathbf{AC}} \langle N'; \tau' \rangle$ and $\langle N'; \tau' \rangle$ is stuck.*

Proof. Since $\langle M; \theta \rangle$ is stuck, it is irreducible and M is not of the form $\mathbf{return}\, V$. Lemma A.30 therefore gives an **AC** computation N' and an **AC** store τ' such that $\langle N; \tau \rangle \to^*_{\mathbf{AC}} \langle N'; \tau' \rangle$, the configuration $\langle N'; \tau' \rangle$ is irreducible, and $M \rhd N'$. By (H1), the computation N' is not of the form $\mathbf{return}\, W$. Hence $\langle N'; \tau' \rangle$ is neither $\perp$ nor a return configuration, and no reduction rule applies to it; that is, it is stuck. □

Corollary A.32. *If $C \sim D$ and C is stuck, then there exists an* **AC** *configuration D' such that*

$$D \to^*_{\mathbf{AC}} D'$$

and D' is stuck.

Proof. Since C is stuck, it is not $\perp$. Hence write $C = \langle M; \theta \rangle$. By the definition of $\sim$, there exist an **AC** configuration D_0 and a label map η such that

$$D \to^*_{\mathbf{AC}} D_0 \qquad \text{and} \qquad \langle M; \theta \rangle \overset{\eta}{\sim}_\sharp D_0.$$

Write $D_0 = \langle N; \tau \rangle$. Proposition A.31 gives N' and τ' such that

$$\langle N; \tau \rangle \to^*_{\mathbf{AC}} \langle N'; \tau' \rangle$$

and $\langle N'; \tau' \rangle$ is stuck. Composing this reduction with $D \to^*_{\mathbf{AC}} D_0$, we obtain the desired stuck target configuration. □

Lemma A.33. *If $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ diverges, then $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges.*

Proof. By Corollary A.25 and repeated use of Theorem A.28. Each source step induces at least one target step. □

Lemma A.34. *If* $\langle M; \emptyset \rangle \to^{+}_{\mathbf{D}} \bot$, *then* $\langle \underline{M}; \emptyset \rangle \to^{+}_{\mathbf{AC}} \bot$.

Proof. By Corollary A.25 and repeated use of Theorem A.28. □

A.2.5 Strong macro-expressibility

Theorem A.35 (preservation of semantics). *For every* $\mathbf{DEL}_{\mathrm{one}}$ *program* M, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ *is defined.*

Proof. We prove the two directions separately.

First suppose that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is defined. Then there exist a $\mathbf{DEL}_{\mathrm{one}}$-value V and a $\mathbf{DEL}_{\mathrm{one}}$-store θ such that

$$\langle M; \emptyset \rangle \to^{*}_{\mathbf{D}} \langle \mathbf{return}\, V; \theta \rangle.$$

By Corollary A.25,

$$\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle.$$

Repeatedly applying Theorem A.28 along the source reduction, we obtain an $\mathbf{AC}$-configuration D such that

$$\langle \underline{M}; \emptyset \rangle \to^{*}_{\mathbf{AC}} D \qquad \text{and} \qquad \langle \mathbf{return}\, V; \theta \rangle \sim D.$$

By the definition of $\sim$, either both configurations are $\bot$, which is impossible because the source configuration is $\langle \mathbf{return}\, V; \theta \rangle$, or there exist an $\mathbf{AC}$-computation N, an $\mathbf{AC}$-store τ, and a label map η such that

$$D \to^{*}_{\mathbf{AC}} \langle N; \tau \rangle \qquad \text{and} \qquad \langle \mathbf{return}\, V; \theta \rangle \overset{\eta}{\sim}_{\sharp} \langle N; \tau \rangle.$$

By the definition of $\sim_{\sharp}$, the last item contains

$$\langle \mathbf{return}\, V; \theta \rangle \overset{\eta}{\sim} \langle N; \tau \rangle.$$

By Lemma A.22, there exists an $\mathbf{AC}$-value W such that

$$N \equiv \mathbf{return}\, W.$$

Hence

$$\langle \underline{M}; \emptyset \rangle \to^{*}_{\mathbf{AC}} \langle \mathbf{return}\, W; \tau \rangle,$$

and therefore $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is defined.

We prove the converse direction by contraposition. Assume that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is undefined. Since $\to_{\mathbf{D}}$ is deterministic, exactly one of the following alternatives holds.

(1) The source evaluation diverges. Then $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ diverges by Lemma A.33.

(2) The source evaluation reaches the error state $\bot$. Thus

$$\langle M; \emptyset \rangle \to^{+}_{\mathbf{D}} \bot.$$

By Lemma A.34,

$$\langle \underline{M}; \emptyset \rangle \to^{+}_{\mathbf{AC}} \bot.$$

Since $\to_{\mathbf{AC}}$ is deterministic and $\bot$ is not a return configuration, the target cannot also terminate successfully.

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

Fig. 17. Translation from $\mathbf{EFF}_{\text{one}}$ to $\mathbf{DEL}_{\text{one}}$

(3) The source evaluation reaches a stuck configuration. Then there exist a $\mathbf{DEL}_{\text{one}}$-computation M' and a $\mathbf{DEL}_{\text{one}}$-store θ such that

$$\langle M; \emptyset \rangle \to_{\mathbf{D}}^{*} \langle M'; \theta \rangle,$$

the configuration $\langle M'; \theta \rangle$ is stuck; in particular, M' is not of the form $\mathbf{return}\,V$. By Corollary A.25 and repeated applications of Theorem A.28 along this source reduction sequence, there exists an $\mathbf{AC}$-configuration D such that

$$\left\langle \underline{M}; \emptyset \right\rangle \to_{\mathbf{AC}}^{*} D \qquad \text{and} \qquad \langle M'; \theta \rangle \sim D.$$

By Corollary A.32, there exists an $\mathbf{AC}$-configuration D' such that

$$D \to_{\mathbf{AC}}^{*} D'$$

and D' is stuck. Therefore

$$\left\langle \underline{M}; \emptyset \right\rangle \to_{\mathbf{AC}}^{*} D'.$$

Since $\to_{\mathbf{AC}}$ is deterministic and a stuck configuration is not a return configuration, the target cannot also reduce to a return configuration. This implies that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is undefined.

Therefore, in all cases, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is undefined. □

THEOREM A.36 (STRONG MACRO-TRANSLATION). *The translation $M \mapsto \underline{M}$ of Figure 10 is a strong macro-translation from $\mathbf{DEL}_{\text{one}}$ to $\mathbf{AC}$.*

PROOF. The translation maps every $\mathbf{DEL}_{\text{one}}$ program to an $\mathbf{AC}$ program and is homomorphic on the $\mathbf{MAM}$ constructors by definition. Every constructor peculiar to $\mathbf{DEL}_{\text{one}}$ is translated by a fixed syntactic abstraction, as shown in Figure 10. The preservation of semantics is shown by Theorem A.35. □

B Supplementary Proofs for Section 4

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

In this section, we prove the correctness of the macro-translation defined in Figure 17.

B.1.1 Well-formedness

Definition B.1. For an $\mathbf{EFF}_{\text{one}}$ computation M, let $\mathrm{LB}(M)$ be the set of continuation labels occurring in M. For an $\mathbf{EFF}_{\text{one}}$ store θ, define

$$\mathrm{LB}(\theta) \stackrel{\text{def}}{=} \bigcup\{\mathrm{LB}(\theta(l)) \mid l \in \mathrm{Dom}(\theta), \theta(l) \neq \mathbf{nil}\}.$$

For an $\mathbf{EFF}_{\text{one}}$ configuration $C = \langle M; \theta\rangle$, define $\mathrm{LB}(C) \stackrel{\text{def}}{=} \mathrm{LB}(M) \cup \mathrm{LB}(\theta)$.

Definition B.2 ($\mathbf{EFF}_{\text{one}}$ well-formedness). An $\mathbf{EFF}_{\text{one}}$ configuration $\langle M; \theta\rangle$ is *well-formed*, written $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(\langle M; \theta\rangle)$, if and only if

$$\mathrm{LB}(\langle M; \theta\rangle) \subseteq \mathrm{Dom}(\theta).$$

Proposition B.3 (preservation of $\mathbf{EFF}_{\text{one}}$ well-formedness). *If* $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(\langle M; \theta\rangle)$ *and* $\langle M; \theta\rangle \to_{\mathbf{E}} \langle M'; \theta'\rangle$, *then* $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(\langle M'; \theta'\rangle)$.

Proof. We prove this by case analysis on the $\mathbf{EFF}_{\text{one}}$ beta-reduction rule used in the source step. Since an evaluation context is unchanged by the reduction and $\mathrm{LB}(C[M]) = \mathrm{LB}(C) \cup \mathrm{LB}(M)$, it suffices to check that the reduct contains only labels defined in the updated store.

In the **MAM** cases and the case (ret), no continuation label is generated and the store is unchanged. Each of these rules reduces a redex to a term built from its subterms, so every continuation label occurring in the reduct already occurs in the redex, and the claim follows from the premise.

In the case (op), we have

$$\langle \mathbf{with}\ H\ \mathbf{handle}\ (\mathcal{H}[\mathsf{op}\, V]); \theta\rangle \to_{\mathbf{E}} \left\langle M_j[V/p_j, l/k_j]; \theta'\right\rangle,$$
$$\theta' \stackrel{\text{def}}{=} \theta[l := \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, x]]$$

for a fresh label $l \notin \mathrm{Dom}(\theta)$. The reduct contains only the labels occurring in the redex together with the fresh label l, all of which belong to $\mathrm{Dom}(\theta') = \mathrm{Dom}(\theta) \cup \{l\}$. The new store entry contains only the labels occurring in H and $\mathcal{H}$, which occur in the redex and hence belong to $\mathrm{Dom}(\theta)$. The other entries are unchanged.

In the case (throw), we have

$$\langle \mathbf{throw}\, l\, V; \theta\rangle \to_{\mathbf{E}} \langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, V]; \theta[l := \mathbf{nil}]\rangle,$$

where $\theta(l) = \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, x]$. The reduct contains only the labels occurring in $\theta(l)$ and in V; the former belong to $\mathrm{LB}(\theta) \subseteq \mathrm{Dom}(\theta)$ by the premise, and the latter to $\mathrm{LB}(M) \subseteq \mathrm{Dom}(\theta)$. Since $\mathrm{Dom}(\theta[l := \mathbf{nil}]) = \mathrm{Dom}(\theta)$ and $\mathrm{LB}(\theta[l := \mathbf{nil}]) \subseteq \mathrm{LB}(\theta)$, the claim follows.

In the case (fail), the configuration reduces to $\perp$, for which the claim is vacuously true. □

We use *label maps*: here a label map is a partial function $\eta : \mathscr{L}_{\mathbf{E}} \rightharpoonup \mathscr{L}_{\mathbf{D}} \times \mathscr{L}_{\mathbf{D}}$ that sends an $\mathbf{EFF}_{\text{one}}$ continuation label l to a pair $\eta(l) = (\eta(l)_1, \eta(l)_2)$ of $\mathbf{DEL}_{\text{one}}$ continuation labels. The label map η keeps track of the source-to-target correspondence between continuation labels. We then parameterize and extend the translation with η by

$$\underline{l}_\eta \stackrel{\text{def}}{=} \eta(l)_1,$$

provided $\eta(l)$ is defined. We call this extension a *runtime translation* and write it as $\underline{\cdot}_\eta$; we extend runtime translations to translations on contexts by mapping a hole to a hole.

Using label maps and the runtime translation, we define coherence conditions and invariant conditions.

Definition B.4 (coherence conditions). For an $\mathbf{EFF}_{\text{one}}$ store θ and a label map η, we define $\mathsf{Coh}(\theta, \eta)$ to hold if and only if the following conditions are satisfied.

(C1) $\mathrm{Dom}(\theta) = \mathrm{Dom}(\eta)$;

(C2) for all $l, l' \in \mathrm{Dom}(\theta)$ and $i, j \in \{1, 2\}$, if $\eta(l)_i = \eta(l')_j$, then $l = l'$ and $i = j$.

Definition B.5 (invariant conditions). For an $\mathbf{EFF}_{\text{one}}$ store θ, a $\mathbf{DEL}_{\text{one}}$ store τ, and a label map η, we define $\mathsf{Inv}(\theta, \tau, \eta)$ to hold if and only if the following conditions are satisfied for every $l \in \mathrm{Dom}(\theta)$.

(IC1) $\eta(l)_1 \in \mathrm{Dom}(\tau)$ and $\eta(l)_2 \in \mathrm{Dom}(\tau)$;

(IC2) if $\theta(l) = \mathbf{nil}$, then $\tau(\eta(l)_1) = \mathbf{nil}$;

(IC3) if $\theta(l) = \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, y]$ for an $\mathbf{EFF}_{\text{one}}$ handler H and an $\mathbf{EFF}_{\text{one}}$ pure context $\mathcal{H}$, then

$$\tau(\eta(l)_1) = \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ \mathbf{throw}\, \eta(l)_2\, y\ \left\{H^{\text{ops}}_{\eta}\right\} \middle|\, z.\mathbf{return}\, z \right\rangle,$$

$$\tau(\eta(l)_2) = \lambda y.\ \left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_{\eta} \middle| H^{\text{ret}}_{\eta} \right\rangle,$$

where H^{ops}_{η} (resp. H^{ret}_{η}) denotes the translation of the operational clauses (resp. return clause) of H with respect to η.

Definition B.6 (simulation relation). For an $\mathbf{EFF}_{\text{one}}$ configuration C and a $\mathbf{DEL}_{\text{one}}$ configuration D, we write $C \sim D$ if and only if either $C = D = \bot$, or there exist an $\mathbf{EFF}_{\text{one}}$ computation M, a $\mathbf{DEL}_{\text{one}}$ computation N, an $\mathbf{EFF}_{\text{one}}$ store θ, a $\mathbf{DEL}_{\text{one}}$ store τ, and a label map η such that

(1) $C = \langle M; \theta \rangle$ and $D = \langle N; \tau \rangle$;

(2) $N \equiv \underline{M}_{\eta}$;

(3) $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(C)$;

(4) $\mathsf{Coh}(\theta, \eta)$ and $\mathsf{Inv}(\theta, \tau, \eta)$.

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 are translated into those in τ.

Lemma B.7. *The following statements hold.*

(1) For any $\mathbf{EFF}_{\text{one}}$ *computation* M, $\langle M; \emptyset \rangle \sim \left\langle \underline{M}; \emptyset \right\rangle$ *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 the source configuration. □

Lemma B.8. *If an* $\mathbf{EFF}_{\text{one}}$ *computation* M *has the free variables* $x_1, \ldots x_n$, *then we have* $\underline{M[V_1/x_1, \ldots, V_n/x_n]}_{\eta} \equiv \underline{M}_{\eta}[\underline{V_1}_{\eta}/x_1, \ldots, \underline{V_n}_{\eta}/x_n]$ *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

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

□

Lemma B.9. *If $C \sim D$ and $C \to^{\beta}_{\mathbf{E}} C'$, then there exists a $\mathbf{DEL}_{\text{one}}$ 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 $C = \langle \mathbf{with}\ H\ \mathbf{handle}\ (\mathcal{H}[\mathsf{op}\, V]); \theta\rangle$ and

$$C \to_{\mathbf{E}} C' = \left\langle M_{\mathsf{op}}[V/p, l/k]; \theta'\right\rangle,$$

where $\theta' \stackrel{\text{def}}{=} \theta[l := \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, x]], H \equiv \{\mathbf{return}\ x \mapsto M_{\text{ret}},\ \ldots (\mathsf{op}\ p\ k \mapsto M_{\mathsf{op}}) \ldots\}$, and l is a fresh label. By the definition of $C \sim D$, there exist τ and η such that

$$D = \left\langle \left\langle \left\langle N \middle| H^{\text{ret}}_{\eta} \right\rangle \left\{H^{\text{ops}}_{\eta}\right\} \middle| z.\mathbf{return}\, z \right\rangle;\ \tau \right\rangle,$$

where

$$N \equiv \underline{\mathcal{H}}_{\eta} \begin{bmatrix} \mathbf{S_0}k.\ \lambda h.\ (\mathbf{let}\ y = (\mathbf{S_0}k'.\ h!\,\mathbf{inj}_{\mathsf{op}}\ (\underline{V}_{\eta}, k'))\ \mathbf{in} \\ \mathbf{throw}\ k\ y\ h) \end{bmatrix}.$$

The configuration D evaluates to

$$D' \stackrel{\text{def}}{=} \left\langle \underline{M_{\mathsf{op}}}_{\eta} \left[\underline{V}_{\eta}/p, m/k\right]; \tau' \right\rangle$$

where m and n are fresh, and

$$\tau' \stackrel{\text{def}}{=} \tau \begin{bmatrix} m := \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ \mathbf{throw}\, n\ y\ \left\{H^{\text{ops}}_{\eta}\right\} \middle| z.\mathbf{return}\, z \right\rangle, \\ n := \lambda y.\ \left\langle \underline{\mathcal{H}}_{\eta}[\mathbf{return}\ y] \middle| H^{\text{ret}}_{\eta} \right\rangle \end{bmatrix}.$$

Let η' be $\eta[l := (m, n)]$. By Lemma B.8, it follows that

$$\underline{M_{\mathsf{op}}}_{\eta}[\underline{V}_{\eta}/p, m/k] \equiv \underline{M_{\mathsf{op}}[V/p, l/k]}_{\eta'}.$$

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

$$C' \sim D',$$

witnessed by η'. It is straightforward to check that $\mathsf{Coh}(\theta', \eta')$ and $\mathsf{Inv}(\theta', \tau', \eta')$ hold, where the freshness of l, m, and n gives (C1) and (C2).

Case (throw):

Suppose $C = \langle \mathbf{throw}\, l\, V; \theta\rangle$ and $C' = \langle \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, V]; \theta'\rangle$, where $\theta(l) = \lambda x.\ \mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, x]$ and $\theta' \stackrel{\text{def}}{=} \theta[l := \mathbf{nil}]$. By the definition of $C \sim D$, there exist τ and η such that $D = \left\langle \mathbf{throw}\, \eta(l)_1\, \underline{V}_{\eta}; \tau \right\rangle$,

$$\tau(\eta(l)_1) = \lambda x.\ \left\langle \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ \mathbf{throw}\, \eta(l)_2\ y\ \left\{H^{\text{ops}}_{\eta}\right\} \middle| z.\mathbf{return}\, z \right\rangle,$$

and

$$\tau(\eta(l)_2) = \lambda y.\ \left\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_{\eta} \middle| H^{\text{ret}}_{\eta} \right\rangle.$$

Thus, D evaluates to

$$D' = \left\langle \left\langle \left(\left\langle \underline{\mathcal{H}[\mathbf{return}\ y]}_{\eta} \middle| H^{\text{ret}}_{\eta} \right\rangle \left[\underline{V}_{\eta}/y\right] \right) \left\{H^{\text{ops}}_{\eta}\right\} \middle| z.\mathbf{return}\, z \right\rangle; \tau' \right\rangle,$$

where $\tau' \overset{\text{def}}{=} \tau[\eta(l)_1 := \mathbf{nil}, \eta(l)_2 := \mathbf{nil}]$. Then, from Lemma B.8, we obtain

$$\left\langle \underline{\mathcal{H}[\mathbf{return}\, y]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle [\underline{V}_\eta / y] \equiv \left\langle \underline{\mathcal{H}[\mathbf{return}\, V]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle.$$

Therefore, we conclude that

$$C' \sim D',$$

witnessed by η, since

$$\underline{\mathbf{with}\ H\ \mathbf{handle}\ \mathcal{H}[\mathbf{return}\, V]}_\eta \equiv \left\langle \left\langle \underline{\mathcal{H}[\mathbf{return}\, V]}_\eta \middle| H^{\text{ret}}_\eta \right\rangle \{H^{\text{ops}}_\eta\} \middle| z.\mathbf{return}\, z \right\rangle.$$

Note that $\mathsf{Coh}(\theta', \eta)$ holds since $\mathrm{Dom}(\theta') = \mathrm{Dom}(\theta)$, and that $\mathsf{Inv}(\theta', \tau', \eta)$ also holds.

Case (fail):

Suppose $C = \langle \mathbf{throw}\, l\, V; \theta \rangle$, $\theta(l) = \mathbf{nil}$, and $C \to^{\beta}_{\mathbf{E}} \bot$. By the definition of $C \sim D$, there exist τ and η such that $D = \left\langle \mathbf{throw}\, \eta(l)_1\, \underline{V}_\eta; \tau \right\rangle$ and $\tau(\eta(l)_1) = \mathbf{nil}$. Hence, the invocation of $\eta(l)_1$ fails and D evaluates to $\bot$, which concludes the current case. □

Lemma B.10. *$\underline{\mathcal{K}[M]}_\eta \equiv \underline{\mathcal{K}}_\eta[\underline{M}_\eta]$ for an arbitrary evaluation context K and computation M.*

Proof. By straightforward induction on K. □

Lemma B.11. *Suppose that $\langle M; \theta \rangle \sim \langle N; \tau \rangle$ holds, and let η be a label map witnessing it, so that $N \equiv \underline{M}_\eta$. If M is decomposed into an evaluation context $\mathcal{C}$ and a redex M', then N is decomposed into the evaluation context $\underline{\mathcal{C}}_\eta$ and the 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 B.12 (simulation). *If $C \sim D$ and $C \to_{\mathbf{E}} C'$, then there exists a $\mathbf{DEL}_{\text{one}}$ 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 $C = \langle M; \theta \rangle$, and according to the semantics of $\mathbf{EFF}_{\text{one}}$, M can be decomposed into an evaluation context $\mathcal{C}$ and a redex M': $M \equiv \mathcal{C}[M']$. By the definition of $C \sim D$, there exist η and τ such that $D = \left\langle \underline{M}_\eta; \tau \right\rangle$ and both $\mathsf{Coh}(\theta, \eta)$ and $\mathsf{Inv}(\theta, \tau, \eta)$ hold. By Lemma B.11, $\underline{M}_\eta$ also can be decomposed into $\underline{\mathcal{C}}_\eta$ and a redex $\underline{M'}_\eta$, namely $\underline{M}_\eta \equiv \underline{\mathcal{C}}_\eta\left[\underline{M'}_\eta\right]$.

If $\langle M'; \theta \rangle \to^{\beta}_{\mathbf{E}} \bot$, then we have $C \to_{\mathbf{E}} \bot$. By Lemma B.9, $\left\langle \underline{M'}_\eta; \tau \right\rangle$ also evaluates to $\bot$. Hence, we conclude that $\left\langle \underline{\mathcal{C}}_\eta\left[\underline{M'}_\eta\right]; \tau \right\rangle \to^{+}_{\mathbf{D}} \bot$, that is, $D \to^{+}_{\mathbf{D}} \bot$.

Next, suppose $\langle M'; \theta \rangle \to^{\beta}_{\mathbf{E}} \langle M''; \theta' \rangle$. By Lemma B.9, we obtain $\left\langle \underline{M'}_\eta; \tau \right\rangle \to^{+}_{\mathbf{D}} \left\langle \underline{M''}_{\eta'}; \tau' \right\rangle$ and $\langle M''; \theta' \rangle \sim \left\langle \underline{M''}_{\eta'}; \tau' \right\rangle$ for some θ', τ', and η' with $\mathsf{Coh}(\theta', \eta')$ and $\mathsf{Inv}(\theta', \tau', \eta')$. Moreover, $\underline{\mathcal{C}}_\eta \equiv \underline{\mathcal{C}}_{\eta'}$ holds since η' is, by construction, an extension of η, and 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 B.10. □

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

Lemma B.13. *Suppose that $\mathrm{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ diverges, i.e., the reduction sequence starting from M is infinite. Then $\mathrm{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ also diverges.*

Proof. By applying Proposition B.12 repeatedly, we also obtain an infinite reduction sequence of $\underline{M}$. Since the reduction of $\mathbf{DEL}_{\text{one}}$ is deterministic, this implies that $\text{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ also diverges. ☐

Lemma B.14. *Suppose that* $\text{Eval}_{\mathbf{EFF}_{\text{one}}}(M)$ *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,* $\text{Eval}_{\mathbf{DEL}_{\text{one}}}(\underline{M})$ *also gets stuck.*

Proof. It suffices to show the following statement:

> Suppose that there exist M, N, θ, τ, and η such that $N \equiv \underline{M}_\eta$, that $\mathsf{WF}_{\mathbf{EFF}_{\text{one}}}(\langle M; \theta\rangle)$, and that $\mathsf{Coh}(\theta, \eta)$ and $\mathsf{Inv}(\theta, \tau, \eta)$ hold. If $\langle M; \theta\rangle$ is stuck, then $\langle N; \tau\rangle$ is also stuck.

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

Case 1:

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

Since $\langle M; \theta\rangle$ 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:

$$M \equiv \mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L_i}}\, x_i \mapsto M_i)_i\}$$

Since $\langle M; \theta\rangle$ is stuck, we know that $V \neq \mathbf{inj}_{\mathrm{L}}\, V'$ for any variable label L of $\mathbf{EFF}_{\text{one}}$ and $\mathbf{EFF}_{\text{one}}$ value V'. Then, by definition, $\underline{V}_\eta$ is also not a variant value. Thus the configuration $\left\langle \underline{M}_\eta; \tau \right\rangle$ is stuck, which concludes the case.

Case 3:

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

It follows that V is not a continuation label, since if $V = l$ for some l, then well-formedness implies $l \in \text{Dom}(\theta)$, which contradicts the assumption that $\langle \mathbf{throw}\, V\, W; \theta\rangle$ is stuck. Therefore, $\underline{V}_\eta$ is also not a continuation label, so $\left\langle \underline{M}_\eta; \tau \right\rangle$ is also stuck.

Case 4:

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

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 $\underline{M_1}_\eta$ is stuck and that $\underline{M_1}_\eta \neq \mathbf{return}\, W$ for any value W. Consequently, the configuration $\left\langle \mathbf{let}\ x = \underline{M_1}_\eta\ \mathbf{in}\ \underline{M_2}_\eta; \tau \right\rangle$ is also stuck, which completes the current case.

Case 5:

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

By the definition of the translation, we obtain

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

Since $\langle M;\theta\rangle$ is stuck, M' can 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 $\underline{M'}_\eta$ can 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 proof.

□

Lemma B.15. *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 Proposition B.12. □

Proposition B.16 (preservation of semantics). $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ *is defined.*

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 B.7 yields that $\langle M;\emptyset\rangle \sim \langle \underline{M};\emptyset\rangle$. By applying Proposition B.12 iteratively, there exist τ and η such that $\langle \underline{M};\emptyset\rangle \to^*_{\mathbf{D}} \langle \mathbf{return}\, \underline{V}_\eta;\tau\rangle$. This implies that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ is defined. We then show the “if” direction. We prove the contrapositive of the proposition: if $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ is undefined, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ is also undefined. There are three cases where $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(M)$ is undefined: the evaluation of M (1) diverges, (2) gets stuck, or (3) reaches $\bot$. In each case, Lemmas B.13, B.14, and B.15 imply that $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ diverges, gets stuck, or reaches $\bot$, respectively. Thus, $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\underline{M})$ is undefined, which completes the proof. □

B.2 From $\mathbf{DEL}_{\mathrm{one}}$ *to* $\mathbf{EFF}_{\mathrm{one}}$

As in Appendix B.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 B.17. 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 $C = D = \bot$, 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) $N \equiv \underline{M}_\eta$.
(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 B.18.

(1) *For any* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M*,* $\langle M; \emptyset\rangle \sim \left\langle \underline{M}; \emptyset\right\rangle$ *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. $\square$

Lemma B.19. *Suppose that a* $\mathbf{DEL}_{\mathrm{one}}$*-computation* M *has the free variables* $x_1, \ldots x_n$*, then we have* $\underline{M[V_1/x_1, \ldots, V_n/x_n]}_{\eta} \equiv \underline{M}_{\eta}[\underline{V_1}_{\eta}/x_1, \ldots, \underline{V_n}_{\eta}/x_n]$ *for any values* $V_1, \ldots V_n$ *and* η*.*

Proof. By straightforward induction on M. $\square$

Lemma B.20. *Suppose that* $C \sim D$ *and* $C \to^{\beta}_{\mathbf{D}} C'$*, 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 $C \to^{\beta}_{\mathbf{D}} C'$. For the sake of brevity, we consider a few non-trivial cases.

Case (shift):

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

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

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:

$$D' \stackrel{\text{def}}{=} \left\langle \underline{M}_{\eta}[m_H/k]; \tau'\right\rangle,$$

where $\tau' \stackrel{\text{def}}{=} \tau[m_H := \lambda y.\ \mathbf{with}\ H\ \mathbf{handle}\ \underline{\mathcal{H}}_{\eta}[\mathbf{return}\, y]]$ and m_H is a fresh label. Let η' be $\eta[l := m_H]$. By Lemma B.19, we obtain

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

Moreover, since m_H is taken 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 $C = \langle \mathbf{throw}\, l\, V; \theta\rangle$, $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\, y] \,|\, x.N\rangle$, and

$$C' = \langle\langle \mathcal{H}[\mathbf{return}\, V] \,|\, x.N\rangle; \theta[l := \mathbf{nil}]\rangle.$$

By the definition of $C \sim D$, we obtain

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

for some τ and η such that

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

where θ, τ, and η satisfy the IC.

D evaluates to

$$D' \stackrel{\text{def}}{=} \left\langle \textbf{with}\ \{\textbf{return}\ x \mapsto \underline{N},\ \texttt{shift0}\ p\ k \mapsto p!\ k\}\ \textbf{handle}\ \underline{\mathcal{H}}_{\eta}\left[\textbf{return}\ \underline{V}_{\eta}\right]; \tau[\eta(l) := \textbf{nil}] \right\rangle$$

$$= \left\langle \textbf{with}\ \{\textbf{return}\ x \mapsto \underline{N},\ \texttt{shift0}\ p\ k \mapsto p!\ k\}\ \textbf{handle}\ \underline{\mathcal{H}[\textbf{return}\ V]}_{\eta}; \tau[\eta(l) := \textbf{nil}] \right\rangle.$$

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

□

Lemma B.21. *$\underline{\mathcal{K}[M]}_{\eta} \equiv \underline{\mathcal{K}}_{\eta}[\underline{M}_{\eta}]$ for an arbitrary context K and computation M.*

Proof. By straightforward induction on K. □

Lemma B.22. *If $\langle M; \theta\rangle \sim \langle N; \tau\rangle$ and M can be decomposed into an evaluation context 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 $\underline{C}_{\eta}$ and a redex $\underline{M'}_{\eta}$.*

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

Proposition B.23 (simulation). *Suppose that $C \sim D$ and $C \longrightarrow_{\mathbf{D}} C'$, then there exists an $\mathbf{EFF}_{\text{one}}$ configuration D' such that $D \longrightarrow^{+}_{\mathbf{E}} D'$ and $C' \sim D'$.*

Proof. Since C is not in normal form, there exist M and θ such that $C = \langle M; \theta\rangle$ and according to the semantics of $\mathbf{DEL}_{\text{one}}$, M can be decomposed into an evaluation context C and a redex M'. By the definition of $C \sim D$, there exist η and τ such that $D = \left\langle \underline{M}_{\eta}; \tau \right\rangle$, and θ, τ, and η satisfy the IC. By Lemma B.22, $\underline{M}_{\eta}$ can also be decomposed into $\underline{C}_{\eta}$ and a redex $\underline{M'}_{\eta}$, namely $\underline{M}_{\eta} \equiv \underline{C}_{\eta}\left[\underline{M'}_{\eta}\right]$.

If $\langle M'; \theta\rangle \longrightarrow^{\beta}_{\mathbf{D}} \perp$, then we have $C \longrightarrow_{\mathbf{D}} \perp$. By Lemma B.20, $\left\langle \underline{M'}_{\eta}; \tau \right\rangle$ also evaluates to $\perp$. Hence, we conclude that $\left\langle \underline{C}_{\eta}\left[\underline{M'}_{\eta}\right]; \tau \right\rangle \longrightarrow^{+}_{\mathbf{E}} \perp$, that is, $D \longrightarrow^{+}_{\mathbf{E}} \perp$.

Next, suppose $\langle M'; \theta\rangle \longrightarrow^{\beta}_{\mathbf{D}} \langle M''; \theta'\rangle$. By Lemma B.20, we obtain $\left\langle \underline{M'}_{\eta}; \tau \right\rangle \longrightarrow^{+}_{\mathbf{E}} \left\langle \underline{M''}_{\eta'}; \tau' \right\rangle$ and $\langle M''; \theta'\rangle \sim \left\langle \underline{M''}_{\eta'}; \tau' \right\rangle$, where θ', τ', and η' satisfy the IC. Moreover, $\underline{C}_{\eta} \equiv \underline{C}_{\eta'}$ holds since the labels in $\mathrm{Dom}(\eta') \setminus \mathrm{Dom}(\eta)$ are fresh and do not appear in C. Therefore, we conclude that $\langle C[M'']; \theta'\rangle \sim \left\langle \underline{C[M'']}_{\eta'}; \tau' \right\rangle$ by Lemma B.21. □

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

Lemma B.24. *We say a $\mathbf{DEL}_{\text{one}}$-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 B.25. *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 Proposition B.23 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 B.26. *Suppose that* $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *gets stuck: there exists a term* M' *and a store* θ *such that* $\langle M; \emptyset\rangle \rightarrow^*_{\mathbf{D}} \langle M'; \theta\rangle$, $M' \neq \mathbf{return}\, V$ *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 $N \equiv \underline{M}_{\eta}$, $\langle M; \theta\rangle$ 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 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 $\langle M; \theta\rangle$ 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 the assumption that $\langle \mathbf{throw}\, V\, W; \theta\rangle$ is stuck. This implies that $\underline{V}_{\eta}$ is not an $\mathbf{EFF}_{\mathrm{one}}$ continuation label and so $\underline{\mathbf{throw}\, V\, W}_{\eta}$ 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 of the form $\mathcal{H}[\mathbf{S_0}k.\ L]$ or 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 of the form $\mathcal{P}[\mathsf{op}\, W]$ or 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 B.27. *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 Proposition B.23. □

COROLLARY B.28 (PRESERVATION OF SEMANTICS). $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ *is defined.*

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 B.18 yields that $\langle M;\emptyset\rangle \sim \langle \underline{M};\emptyset\rangle$. By applying Proposition B.23 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 $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is undefined, $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ is also undefined. There are three cases where $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M)$ is undefined: the evaluation of M (1) diverges, (2) gets stuck, or (3) reaches $\bot$. In each case, Lemmas B.25, B.26, and B.27 imply that $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ also diverges, gets stuck, or reaches $\bot$, respectively. Thus, $\mathrm{Eval}_{\mathbf{EFF}_{\mathrm{one}}}(\underline{M})$ is undefined, which completes the proof. □

C Supplementary proofs for Section 5.1

In this section, we give a proof of Theorem 5.1 and Theorem 5.3.

C.1 Proof of Theorem 5.1

C.1.1 Lemmas

Definition C.1. For a $\mathbf{DEL}_{\mathrm{one}}$ term E, let $\mathrm{Lab}(E)$ be the set of continuation labels occurring in E. For a store θ, let

$$\mathrm{Lab}(\theta) \overset{\mathrm{def}}{=} \mathrm{Dom}(\theta) \cup \bigcup_{l\in \mathrm{Dom}(\theta),\theta(l)\neq \mathbf{nil}} \mathrm{Lab}(\theta(l)).$$

For a configuration $\langle M;\theta\rangle$, let $\mathrm{Lab}(\langle M;\theta\rangle) = \mathrm{Lab}(M) \cup \mathrm{Lab}(\theta)$.

Definition C.2. For a $\mathbf{DEL}_{\mathrm{one}}$ term E and a store θ, define $\mathrm{Reach}(E,\theta)$ to be the smallest subset of $\mathrm{Dom}(\theta)$ such that:

- if $l \in \mathrm{Lab}(E) \cap \mathrm{Dom}(\theta)$, then $l \in \mathrm{Reach}(E,\theta)$;
- if $l \in \mathrm{Reach}(E,\theta)$ and $\theta(l) = K \neq \mathbf{nil}$, then every label in $\mathrm{Lab}(K) \cap \mathrm{Dom}(\theta)$ also belongs to $\mathrm{Reach}(E,\theta)$.

Definition C.3. For a set S of labels, write

$$\theta \succeq_S \theta'$$

if, for every $l \in S$, either $\theta'(l) = \theta(l) \neq \mathbf{nil}$ or $\theta'(l) = \mathbf{nil}$. In other words, restricted on S, θ' is obtained from θ by possibly invalidating some entries. Note that $\succeq_S$ is transitive given a fixed set S.

LEMMA C.4. *Suppose* $\langle M;\theta\rangle \to^*_{\mathbf{D}} \langle M';\theta'\rangle$ *and let* $S \overset{\mathrm{def}}{=} \mathrm{Reach}(M,\theta)$*. Then,*

(1) $\theta \succeq_S \theta'$;
(2) $\mathrm{Reach}(M, \theta') \subseteq S$.

Proof. By induction on the length of the reduction sequence. For the base case, there is nothing to prove. For the inductive step, assume

$$\langle M; \theta\rangle \to_{\mathbf{D}}^* \langle M_0; \theta_0\rangle \to_{\mathbf{D}} \langle M'; \theta'\rangle.$$

By the induction hypothesis, we obtain

- $\theta \succeq_S \theta_0$;
- $\mathrm{Reach}(M, \theta_0) \subseteq S$.

We conduct case analysis on the rule applied in $\langle M_0; \theta_0\rangle \to_{\mathbf{D}} \langle M'; \theta'\rangle$.

Cases (**MAM**) and (ret):

These rules do not change the store, so the consequences trivially hold.

Case (shift):

In this case, a fresh label l is taken and bound in the store: θ' equals $\theta_0[l \mapsto K]$ for some K. (1) trivially holds.[19] l does not occur in M, so l is not reachable from M. Thus $l \notin \mathrm{Reach}(M, \theta')$ and $\mathrm{Reach}(M, \theta') = \mathrm{Reach}(M, \theta_0) \subseteq S$.

Case (throw):

This rule invalidates a label l in $\mathrm{Dom}(\theta_0)$: $\theta' = \theta_0[l \mapsto \mathbf{nil}]$. Thus, (1) trivially holds. We shall prove $\mathrm{Reach}(M, \theta') \subseteq \mathrm{Reach}(M, \theta_0)$ by induction on the well-founded relation on $\mathrm{Reach}(M, \theta')$ induced from reachability. Let $m \in \mathrm{Reach}(M, \theta')$. Then, there are two possible cases:

1. $m \in \mathrm{Lab}(M) \cap \mathrm{Dom}(\theta')$, or
2. there exists a label $m' \in \mathrm{Reach}(M, \theta')$ such that $\theta'(m') \neq \mathbf{nil}$ and $m \in \mathrm{Lab}(\theta'(m')) \cap \mathrm{Dom}(\theta')$.

For case 1, m also belongs to $\mathrm{Dom}(\theta_0)$ since $\mathrm{Dom}(\theta') = \mathrm{Dom}(\theta_0)$. Thus, $m \in \mathrm{Reach}(M, \theta_0)$. For case 2, $\theta'(m') \neq \mathbf{nil}$ implies $m' \neq l$, so $\theta'(m') = \theta_0(m') \neq \mathbf{nil}$. Thus $m \in \mathrm{Lab}(\theta_0(m')) \cap \mathrm{Dom}(\theta_0)$, and with the induction hypothesis that $m' \in \mathrm{Reach}(M, \theta_0)$, we obtain $m \in \mathrm{Reach}(M, \theta_0)$.

Therefore, $\mathrm{Reach}(M, \theta') \subseteq \mathrm{Reach}(M, \theta_0) \subseteq S$ holds.

This completes the proof. □

Definition C.5. A *finite permutation* π of continuation labels is a bijection on $\mathscr{L}_{\mathbf{D}}$ whose support $\mathrm{Supp}(\pi) \stackrel{\mathrm{def}}{=} \{l \in \mathscr{L}_{\mathbf{D}} \mid \pi(l) \neq l\}$ is finite. It acts on values and computations by renaming labels using π. We write the action as $\pi \cdot E$ where E is a value or computation. For a store θ, we define the action $\pi \cdot \theta$ by

$$(\pi \cdot \theta)(l) \stackrel{\mathrm{def}}{=} \pi \cdot (\theta(\pi^{-1}(l))),$$

provided $\pi^{-1}(l) \in \mathrm{Dom}(\theta)$.

Lemma C.6. *Let θ and θ' be $\mathbf{DEL}_{\mathrm{one}}$ stores, S is a subset of* $\mathrm{Dom}(\theta) \cap \mathrm{Dom}(\theta')$*, and M be a computation. Assume there exist a $\mathbf{DEL}_{\mathrm{one}}$ value V and a store σ' such that*

$$\theta \succeq_S \theta', \quad \mathrm{Reach}(M, \theta') \subseteq S, \quad \textit{and} \quad \langle M; \theta'\rangle \to_{\mathbf{D}}^* \langle \mathbf{return}\, V; \sigma'\rangle.$$

[19] Note that $\succeq_S$ is transitive.

Then, there exist a store σ and a finite permutation π with $\pi|_S = \mathrm{id}$ such that

$$\langle M;\theta\rangle \to_{\mathbf{D}}^{*} \langle \mathbf{return}\,(\pi \cdot V);\sigma\rangle.$$

Proof. Fix the reduction sequence starting from $\langle M;\theta'\rangle$:

$$\langle M;\theta'\rangle = C_0 \to_{\mathbf{D}} C_1 \to_{\mathbf{D}} \cdots \to_{\mathbf{D}} \cdots \to_{\mathbf{D}} C_n = \langle \mathbf{return}\,V;\sigma'\rangle,$$

with $C_i = \langle M_i;\theta_i\rangle$. By induction on $i \le n$, we construct a reduction sequence

$$\langle M;\theta\rangle = \bar{C}_0 \to_{\mathbf{D}} \bar{C}_1 \to_{\mathbf{D}} \cdots \to_{\mathbf{D}} \bar{C}_i,$$

with $\bar{C}_i = \left\langle \bar{M}_i;\bar{\theta}_i\right\rangle$, finite subsets $F_i, \bar{F}_i$ of $\mathscr{L}_{\mathbf{D}}$, and bijections

$$\rho_i : S \cup F_i \to S \cup \bar{F}_i$$

such that $\rho_i|_S = \mathrm{id}$ and

(1) $\bar{M}_i = \rho_i \cdot M_i$;
(2) for every $l \in S \cup F_i$, if $\theta_i(l) \ne \mathbf{nil}$, then $\bar{\theta}_i(\rho_i(l)) = \rho_i \cdot \theta_i(l)$.[20]

For $i = 0$, take $F_0 = \bar{F}_0 = \emptyset$ and $\rho_0 = \mathrm{id}_S$. Then, $\bar{M}_0 = \rho_0 \cdot M = M$. If $l \in S$ and $\theta'(l) \ne \mathbf{nil}$, $\theta \succeq_S \theta'$ implies $\theta(l) = \theta'(l)$.

Now assume we have constructed a reduction sequence of length i, finite sets F_i and $\bar{F}_i$, and bijections ρ_i up to i. We conduct case analysis on the step $C_i \to_{\mathbf{D}} C_{i+1}$.

Cases (MAM) and (ret):

In these cases, the next step is determined by the shape of the current computation. Since $\bar{M}_i = \rho_i \cdot M_i$, the same rule is applied to it, with the store $\bar{\theta}_i$ unchanged. Thus, $\left\langle \bar{M}_{i+1};\bar{\theta}_{i+1}\right\rangle \overset{\mathrm{def}}{=} \left\langle \rho_i \cdot M_{i+1};\bar{\theta}_i\right\rangle$. It is straightforward to check that the side condition is preserved.

Case (shift):

Assume the reduction $C_i \to_{\mathbf{D}} C_{i+1}$ is

$$\langle\langle \mathcal{H}[\mathbf{S_0}k.\ M] \,|\, x.N\rangle;\theta_i\rangle \to_{\mathbf{D}} \langle M[l/k];\theta_i[l \mapsto \lambda y.\ \langle \mathcal{H}[\mathbf{return}\,y] \,|\, x.N\rangle]\rangle,$$

where l is a fresh label. Then, $\bar{C}_i$ has the same redex modulo the renaming by ρ_i, so the same rule is applied to it, creating a fresh label l'. Here, we define $F_{i+1} \overset{\mathrm{def}}{=} F_i \cup \{l\}$, $\bar{F}_{i+1} \overset{\mathrm{def}}{=} \bar{F}_i \cup \{l'\}$, and $\rho_{i+1} \overset{\mathrm{def}}{=} \rho_i[l \mapsto l']$. The content of l' in the updated store is $\rho_i \cdot \theta_i(l)$, which equals $\rho_{i+1} \cdot \theta_i(l)$ since l does not occur inside $\theta_i(l)$. So the side condition is preserved.

Case (throw):

Assume the reduction $C_i \to_{\mathbf{D}} C_{i+1}$ is

$$\langle \mathbf{throw}\,l\,V;\theta\rangle \to_{\mathbf{D}} \langle\langle \mathcal{H}[\mathbf{return}\,V] \,|\, x.N\rangle;\theta[l \mapsto \mathbf{nil}]\rangle,$$

where $\theta(l) = \lambda y.\ \langle \mathcal{H}[\mathbf{return}\,y] \,|\, x.N\rangle$. If l occurs inside M, then Lemma C.4 implies

$$l \in \mathrm{Reach}(M,\theta_i) \subseteq \mathrm{Reach}(M,\theta') \subseteq S.$$

Otherwise, we get $l \in F_i$ by construction. In either case, $\rho_i(l)$ is defined, and the side condition on the i-th step implies that

$$\bar{\theta}_i(\rho_i(l)) = \rho_i \cdot \theta_i(l).$$

[20]Note that ρ_i naturally extends to a finite permutation by $\rho_i(l) \overset{\mathrm{def}}{=} l$ if $l \notin S \cup F_i$.

Thus, $\bar{C}_i$ has the same redex modulo the renaming by ρ_i, and we obtain $\bar{M}_{i+1} = \rho_i \cdot M_{i+1}$ and $\bar{\theta}_{i+1} = \bar{\theta}_i[\rho(l) \mapsto \mathbf{nil}]$. It is straightforward to check that the side condition is preserved.

Case (fail):

This case is impossible.

Finally, we obtain $\bar{M}_n = \rho_n \cdot \mathbf{return}\, V = \mathbf{return}\,(\rho_n \cdot V)$ and $\rho_i|_S = \mathrm{id}$, which is the desired result. □

Lemma C.7. *If* $\langle \mathbf{let}\ x = M\ \mathbf{in}\ N; \theta\rangle \rightarrow^*_{\mathbf{D}} \langle \mathbf{return}\, V; \theta''\rangle$*, then there exist a value* V' *and a store* θ' *such that* $\langle M; \theta\rangle \rightarrow^*_{\mathbf{D}} \langle \mathbf{return}\, V'; \theta'\rangle$ *and* $\langle N[V'/x]; \theta'\rangle \rightarrow^*_{\mathbf{D}} \langle \mathbf{return}\, V; \theta''\rangle$*.*

Proof. By induction on the length of the reduction sequence. Since the frame $\mathbf{let}\ x = [\,]\ \mathbf{in}\ N$ contains no dollar frame, redexes for the (shift) rule can occur only within M. The frame is preserved until M becomes $\mathbf{return}\, V'$, as the only rule that can consume the frame is (F). □

C.1.2 Proof of Theorem 5.1

Let

$$\Omega \stackrel{\text{def}}{=} (\lambda x.\ x\,!\,x)\ \{\lambda x.\ x\,!\,x\},$$

and define

$$\Delta_{\mathrm{A,B}}(x, y) \stackrel{\text{def}}{=} \left(\begin{array}{l} \mathbf{case}\ (x, y)\ \mathbf{of}\ \{ \\ \quad (\mathbf{inj}_{\mathrm{A}}\ \_, \mathbf{inj}_{\mathrm{B}}\ \_) \mapsto \mathbf{return}\ () \\ \quad \_ \mapsto \Omega \\ \} \end{array}\right).$$

Consider the **REF** program

$$M_{\neq} \stackrel{\text{def}}{=} \left(\begin{array}{l} \mathbf{let}\ r = \mathbf{create}\, \mathbf{inj}_{\mathrm{A}}\ ()\ \mathbf{in} \\ \mathbf{let}\ i = \mathbf{get}\ r\ \mathbf{in} \\ \mathbf{let}\ \_ = \mathbf{set}\ r\ \mathbf{inj}_{\mathrm{B}}\ ()\ \mathbf{in} \\ \mathbf{let}\ j = \mathbf{get}\ r\ \mathbf{in} \\ \Delta_{\mathrm{A,B}}(i, j) \end{array}\right).$$

In **REF**, the first **get** returns $\mathbf{inj}_{\mathrm{A}}\ ()$ and the second **get** returns $\mathbf{inj}_{\mathrm{B}}\ ()$, and therefore $\mathrm{Eval}_{\mathbf{REF}}(M_{\neq})$ is defined.

Suppose that there exists a weak macro-translation $\underline{\cdot}$ from **REF** to $\mathbf{DEL}_{\mathrm{one}}$. Then, since $\underline{\cdot}$ acts homomorphically on **MAM** constructors, the translation $\underline{M_{\neq}}$ has the shape

$$\begin{array}{l} \mathbf{let}\ r = \underline{\mathbf{create}}\,(\mathbf{inj}_{\mathrm{A}}\ ())\ \mathbf{in} \\ \mathbf{let}\ i = \underline{\mathbf{get}}\, r\ \mathbf{in} \\ \mathbf{let}\ \_ = \underline{\mathbf{set}}\, r\ (\mathbf{inj}_{\mathrm{B}}\ ())\ \mathbf{in} \\ \mathbf{let}\ k = \underline{\mathbf{get}}\, r\ \mathbf{in} \\ \Delta_{\mathrm{A,B}}(i, k) \end{array}$$

and $\underline{M_{\neq}}$ contains no continuation label. Since $\mathrm{Eval}_{\mathbf{REF}}(M_{\neq})$ is defined, so is $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}\left(\underline{M_{\neq}}\right)$: there exists a store θ such that

$$\left\langle \underline{M_{\neq}}; \emptyset\right\rangle \rightarrow^*_{\mathbf{D}} \langle \mathbf{return}\,(); \theta\rangle.$$

Apply Lemma C.7 repeatedly to this reduction sequence. Then there exist values X, V_1, W, V_2 and stores $\theta_0, \theta_1, \theta_2, \theta_3$ such that

$$\left\langle \underline{\mathbf{create}}\,(\mathbf{inj}_{\mathrm{A}}\ ()); \emptyset\right\rangle \rightarrow^*_{\mathbf{D}} \langle \mathbf{return}\, X; \theta_0\rangle,$$

$$\left\langle \underline{\textbf{get}}\, X; \theta_0 \right\rangle \rightarrow^*_{\mathbf{D}} \langle \textbf{return}\, V_1; \theta_1 \rangle, \tag{12}$$
$$\left\langle \underline{\textbf{set}}\, X\, (\textbf{inj}_{\mathrm{B}}\, ()); \theta_1 \right\rangle \rightarrow^*_{\mathbf{D}} \langle \textbf{return}\, W; \theta_2 \rangle,$$
$$\left\langle \underline{\textbf{get}}\, X; \theta_2 \right\rangle \rightarrow^*_{\mathbf{D}} \langle \textbf{return}\, V_2; \theta_3 \rangle, \tag{13}$$

and moreover the evaluation of $\Delta_{\mathrm{A,B}}(V_1, V_2)$ terminates successfully. Thus, $V_1 \equiv \textbf{inj}_{\mathrm{A}}\, V_1'$ and $V_2 \equiv \textbf{inj}_{\mathrm{B}}\, V_2'$ for some values V_1' and V_2'.

By applying Lemma C.4, we obtain

$$\theta_0 \succeq_S \theta_1 \succeq_S \theta_2,$$

where $S \stackrel{\text{def}}{=} \mathrm{Reach}(\underline{\textbf{get}}\, X, \theta_0) = \mathrm{Reach}(X, \theta_0)$ (note that the fixed syntactic abstraction $\underline{\textbf{get}}$ does not contain any continuation labels), and

$$S \supseteq \mathrm{Reach}(X, \theta_2).$$

Thus, by applying Lemma C.6 to (12) and (13), we obtain

$$\left\langle \underline{\textbf{get}}\, X; \theta_0 \right\rangle \rightarrow^*_{\mathbf{D}} \left\langle \pi \cdot \textbf{return}\, V_2; \sigma_0' \right\rangle,$$

for some finite permutation π and a store σ_0'. The determinism of the reduction implies

$$V_1 = \pi \cdot V_2.$$

Since finite permutations acts homomorphically on constructors other than continuation labels, we obtain

$$\begin{aligned} V_1 &= \pi \cdot V_2 \\ &= \pi \cdot \textbf{inj}_{\mathrm{B}}\, V_2' \\ &= \textbf{inj}_{\mathrm{B}}\, (\pi \cdot V_2'), \end{aligned}$$

which is a contradiction since V_1 is of form $\textbf{inj}_{\mathrm{A}}\, V_1'$. Therefore, there is no weak macro-translation from **REF** to $\mathbf{DEL}_{\mathrm{one}}$.

C.2 Proof of Theorem 5.3

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

$$\underline{l}_\eta \stackrel{\text{def}}{=} \eta(l).$$

Definition C.8. 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) $N \equiv \underline{M}_\eta$
(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

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

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 C.9. *For any* **AC** *store* θ *and value* V,

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

PROOF. By straightforward calculation. □

Next, we establish a substitution property for the translation.

LEMMA C.10. *Suppose that a* **REF**-*computation* M *has free variables* $x_1, \ldots x_n$. *Then for any values* $V_1, \ldots V_n$ *and label map* 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 C.11. *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 brevity, we focus on the reduction rules specific to **REF**.

Case (create):

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

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

The configuration D evaluates to $D' = \left\langle \mathbf{return}\, l'; \tau\left[l' := \left\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ loop!\, \underline{V}_{\eta}\ y\right\}\right]\right\rangle$. Let η' be $\eta[l := l']$. By Lemma C.10, we have $\mathbf{return}\, l' \equiv \underline{\mathbf{return}\, l}_{\eta'}$. Therefore, we conclude that

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

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

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

and

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

The reduction of D proceeds as follows:

$$\begin{aligned} D &= \left\langle \mathbf{resume}\ (\eta(l))\ \left(\mathbf{inj}_{\mathrm{Set}}\, \underline{V}_{\eta}\right); \tau\right\rangle \\ &\to_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\left\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y\right\}!\ \left(\mathbf{inj}_{\mathrm{Set}}\, \underline{V}_{\eta}\right)\right); \tau[(\eta(l)) := \mathbf{nil}]\right\rangle \\ &\to^{+}_{\mathbf{AC}} \left\langle (\eta(l)) : \left(loop!\, \underline{\theta(l)}_{\eta}\ \left(\mathbf{inj}_{\mathrm{Set}}\, \underline{V}_{\eta}\right)\right); \tau[(\eta(l)) := \mathbf{nil}]\right\rangle \end{aligned}$$

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

$$\to^{+}_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\begin{array}{l} \mathbf{case}\ \left(\mathbf{inj}_{\mathrm{Set}}\, \underline{V}_{\eta}\right)\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ \_ \mapsto \mathbf{let}\ y = \mathbf{yield}\, \underline{\theta(l)}_{\eta}\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ loop!\, v\, y\} \end{array} \right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

$$\to_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ loop!\, \underline{V}_{\eta}\ y\right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

$$\to_{\mathbf{AC}} \left\langle \mathbf{return}\ (); \tau\left[(\eta(l)) := \left\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ loop!\, \underline{V}_{\eta}\ y\right\}\right] \right\rangle$$

Therefore, we obtain

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

since the updated stores and η satisfy the IC.

Case (get):

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

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

and

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

The reduction of D proceeds as follows:

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

$$\to_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\left\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y\right\}!\ \left(\mathbf{inj}_{\mathrm{Get}}\ ()\right)\right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

$$\to^{+}_{\mathbf{AC}} \left\langle (\eta(l)) : \left(loop!\, \underline{\theta(l)}_{\eta}\ \left(\mathbf{inj}_{\mathrm{Get}}\ ()\right)\right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

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

$$\to^{+}_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\begin{array}{l} \mathbf{case}\ \left(\mathbf{inj}_{\mathrm{Get}}\ ()\right)\ \mathbf{of}\{ \\ \quad \mathbf{inj}_{\mathrm{Get}}\ \_ \mapsto \mathbf{let}\ y = \mathbf{yield}\, \underline{\theta(l)}_{\eta}\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y \\ \quad \mathbf{inj}_{\mathrm{Set}}\ v \mapsto \mathbf{let}\ y = \mathbf{yield}\ ()\ \mathbf{in}\ loop!\, v\, y\} \end{array} \right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

$$\to_{\mathbf{AC}} \left\langle (\eta(l)) : \left(\mathbf{let}\ y = \underline{\theta(l)}_{\eta}\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y\right); \tau[(\eta(l)) := \mathbf{nil}] \right\rangle$$

$$\to_{\mathbf{AC}} \left\langle \mathbf{return}\, \underline{\theta(l)}_{\eta}; \tau\left[(\eta(l)) := \left\{\lambda x.\ \mathbf{let}\ y = \mathbf{return}\, x\ \mathbf{in}\ loop!\, \underline{\theta(l)}_{\eta}\ y\right\}\right] \right\rangle$$

Therefore, we obtain

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

since the updated stores and η satisfy the IC.

$\square$

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 C.12. *$\underline{\mathcal{K}[M]}_\eta \equiv \underline{\mathcal{K}}_\eta[\underline{M}_\eta]$ for an arbitrary context K and computation M.*

Proof. By straightforward induction on K. □

Lemma C.13. *If $\langle M;\theta\rangle \sim \langle N;\tau\rangle$ and M can be decomposed into an evaluation context C and a redex M', then there exists some η for which $N \equiv \underline{M}_\eta$ can be also decomposed into an evaluation context $\underline{C}_\eta$ and a redex $\underline{M'}_\eta$ such that $N \equiv \underline{C[M']}_\eta$.*

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

Proposition C.14 (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 $C = \langle M;\theta\rangle$, $D = \left\langle \underline{M}_\eta;\tau\right\rangle$, and θ, τ, and η satisfy the IC. Since C is not in normal form, M can be decomposed into an evaluation context C and a redex M'. By Lemma C.13, $\underline{M}_\eta$ can also be decomposed into $\underline{C}_\eta$ and a redex $\underline{M'}_\eta$, namely, $\underline{M}_\eta \equiv \underline{C}_\eta\left[\underline{M'}_\eta\right]$.

Now, suppose $\langle M';\theta\rangle \to^{\beta}_{\mathbf{R}} \langle M'';\theta'\rangle$. By Lemma C.11, there exist η' and τ' such that $\left\langle \underline{M'}_\eta;\tau\right\rangle \to^{+}_{\mathbf{AC}} \left\langle \underline{M''}_{\eta'};\tau'\right\rangle$, $\langle M'';\theta'\rangle \sim \left\langle \underline{M''}_{\eta'};\tau'\right\rangle$, and θ', τ', and η' satisfy the IC. Moreover, $\underline{C}_\eta \equiv \underline{C}_{\eta'}$ holds since the labels in $\mathrm{Dom}(\eta')\setminus\mathrm{Dom}(\eta)$ are fresh and do not appear in C. Hence, we conclude that $\langle C[M''];\theta'\rangle \sim \left\langle \underline{C[M'']}_{\eta'};\tau'\right\rangle$ by Lemma C.12. □

Lemma C.15. *For any* **REF***-computation M, $\langle M;\emptyset\rangle \sim \left\langle \underline{M};\emptyset\right\rangle$ holds.*

Proof. By induction on M. □

Lemma C.16. *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. This property is preserved under reduction.*

Proof. By straightforward case analysis on M. □

Lemma C.17. *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 C.14 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 C.18. *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 $N \equiv \underline{M}_\eta$, $\langle M;\theta\rangle$ 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:

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

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 $\left\langle \underline{\textbf{case } V \textbf{ of } (x_1, x_2) \mapsto M}_\eta ; \tau \right\rangle$ is stuck, and this concludes the current case.
Case 2:

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

It follows that $V \neq l$ for any continuation label l, since if $V = l$ for some l, then by the well-formedness of $\langle \textbf{throw}\, V\, W; \theta \rangle$, we know that $l \in \mathrm{Dom}(\theta)$. However, this contradicts that $\langle \textbf{throw}\, V\, W; \theta \rangle$ is stuck. This implies that $\underline{\textbf{set } V\ W}_\eta$ is also stuck, which concludes the current case.
Case 3:

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

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

□

We now prove Theorem 5.3.

Proof of Theorem 5.3. We prove that $\mathrm{Eval}_{\mathbf{REF}}(M)$ is defined if and only if $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is defined and show the "only if" direction first. Suppose that $\langle M; \emptyset \rangle \to^{+}_{\mathbf{R}} \langle \textbf{return}\, V; \theta \rangle$ for some θ. By Lemma C.15, we obtain $\langle M; \emptyset \rangle \sim \langle \underline{M}; \emptyset \rangle$. By repeatedly applying Proposition C.14, it follows that there exist η and τ such that

$$\langle \underline{M}; \emptyset \rangle \to^{+}_{\mathbf{AC}} \left\langle \textbf{return}\, \underline{V}_\eta ; \tau \right\rangle.$$

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

Next, we prove the contrapositive of the "if" direction. There are two cases where $\mathrm{Eval}_{\mathbf{REF}}(M)$ is undefined: the evaluation of M (1) diverges, or (2) gets stuck. In each case, Lemmas C.17 and C.18 imply that $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ also diverges or gets stuck, respectively. Thus, $\mathrm{Eval}_{\mathbf{AC}}(\underline{M})$ is undefined, which completes the proof.

□

D Supplementary proofs for Section 5.2

In this section, we give the proof of Theorem 5.5.

Lemma D.1. *If a* **REF** *configuration* $\langle M; \theta \rangle$ *diverges,*[21] *then for every* **REF** *evaluation context* $\mathcal{E}$, $\langle \mathcal{E}[M]; \theta \rangle$ *also diverges.*

[21]We say that a configuration C diverges when the maximal reduction sequence starting from C is infinite.

Proof. By induction on the structure of $\mathcal{E}$. The base case is trivial. Let $\mathcal{F}$ be the outermost frame of $\mathcal{E}$ and $\mathcal{E}'$ be the inner evaluation context. By the induction hypothesis, there exists the maximal reduction sequence

$$\langle \mathcal{E}'[M]; \theta \rangle = C_0 \to_{\mathbf{R}} C_1 \to_{\mathbf{R}} C_2 \to_{\mathbf{R}} \cdots .$$

By the definition of the reduction, each step

$$C_i = \langle M_i; \theta_i \rangle \to_{\mathbf{R}} C_{i+1} = \langle M_{i+1}; \theta_{i+1} \rangle$$

induces the step

$$\langle \mathcal{F}[M_i]; \theta_i \rangle \to_{\mathbf{R}} \langle \mathcal{F}[M_{i+1}]; \theta_{i+1} \rangle.$$

Thus, we obtain an infinite reduction sequence

$$\langle \mathcal{E}[M]; \theta \rangle = \langle \mathcal{F}[\mathcal{E}'[M]]; \theta \rangle \to_{\mathbf{R}} \langle \mathcal{F}[M_1]; \theta_1 \rangle \to_{\mathbf{R}} \cdots .$$

Since the reduction is deterministic, $\langle \mathcal{E}[M]; \theta \rangle$ diverges. □

Lemma D.2. *In* **REF***, for any evaluation context $\mathcal{E}$, computation M, and store θ, the evaluation of $\langle \mathcal{E}[\mathbf{let}\ \_ = M\ \mathbf{in}\ \Omega]; \theta \rangle$ does not terminate successfully. In particular, the evaluation gets stuck at the configuration of the form $\langle \mathcal{E}[\mathbf{let}\ \_ = M_0\ \mathbf{in}\ \Omega]; \theta_0 \rangle$ for some computation M_0 and store θ_0, or diverges.*

Proof. Let $\mathcal{L} \stackrel{\text{def}}{=} \mathbf{let}\ \_ = [\ ]\ \mathbf{in}\ \Omega$. Suppose the evaluation of the given configuration terminates successfully. Then, there exist a value V and a store θ' such that

$$\langle \mathcal{E}[\mathcal{L}[M]]; \theta \rangle \to_{\mathbf{R}}^{+} \langle \mathbf{return}\, V; \theta' \rangle.$$

Here, we use the following lemma, which can be easily checked.

Lemma D.3. *Given an arbitrary evaluation context C and a store τ, and suppose*

$$\langle C[\mathcal{L}[M]]; \tau \rangle \to_{\mathbf{R}} \langle P; \tau' \rangle$$

is a **REF** *reduction. Then one of the following two cases holds.*

(1) The step is performed inside M.
(2) The let-frame $\mathcal{L}$ is eliminated: there exists a value W such that

$$M \equiv \mathbf{return}\, W,$$

and

$$\langle C[\mathcal{L}[\mathbf{return}\, W]]; \tau \rangle \to_{\mathbf{R}} \langle C[\Omega]; \tau \rangle.$$

By this lemma, as long as the let-frame $\mathcal{L}$ has not been eliminated, each reduction step can only occur inside M. Since the final configuration no longer contains the let-frame, there must be a step where it is eliminated. Hence, for some evaluation context $\mathcal{E}_0$, value W, and store θ_0, the sequence is of the form

$$\langle \mathcal{E}[\mathcal{L}[M]]; \theta \rangle \to_{\mathbf{R}}^{*} \langle \mathcal{E}_0[\mathcal{L}[\mathbf{return}\, W]]; \theta_0 \rangle \to_{\mathbf{R}} \langle \mathcal{E}_0[\Omega]; \theta_0 \rangle.$$

However, Ω diverges, and by Lemma D.1, $\langle \mathcal{E}_0[\Omega]; \theta_0 \rangle$ also diverges. Since the reduction is deterministic, this implies that the unique maximal reduction sequence starting from $\langle \mathcal{E}[\mathcal{L}[M]]; \theta \rangle$ is infinite. This contradicts the assumption that reducing $\langle \mathcal{E}[\mathcal{L}[M]]; \theta \rangle$ yields a finite reduction sequence to a return configuration.

Therefore, the evaluation cannot reach a return configuration. In particular, the evaluation either diverges, or reaches a stuck configuration of the form $\langle \mathcal{E}[\mathbf{let}\ \_ = M_0\ \mathbf{in}\ \Omega]; \theta_0\rangle$ for some computation M_0 and store θ_0. ◻

Proposition D.4. *Let* $\mathcal{L} \stackrel{\text{def}}{=} \mathbf{let}\ \_ = [\ ]\ \mathbf{in}\ \Omega$. *For any* **REF** *computations* M *and* N, *the thunks* $\{\mathcal{L}[M]\}$ *and* $\{\mathcal{L}[N]\}$ *are observationally equivalent. That is, for every context* C *that expects a value,* $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[M]\}])$ *is defined if and only if* $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[N]\}])$ *is defined.*

We show this proposition by proving a lock-step simulation. The idea is that $\{\mathcal{L}[M]\}$ and $\{\mathcal{L}[N]\}$ are *values*, and thus they are copied around by substitution during a reduction; the only step that can reveal the difference between them is the one that forces them, and such a step leads to a configuration whose evaluation never terminates successfully by Lemma D.2.

Definition D.5. We define the relation $\sim$ on **REF** values and on **REF** computations as the least pair of relations closed under the following two rules:

(S1) $\{\mathcal{L}[M]\} \sim \{\mathcal{L}[N]\}$ for any **REF** computations M and N;

(S2) for every n-ary constructor F of **REF** and terms $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$ with $P_i \sim Q_i$ for each i,

$$F(P_1, \ldots, P_n) \sim F(Q_1, \ldots, Q_n).$$

We extend $\sim$ to (general) contexts by regarding the hole as a nullary constructor, to stores by letting $\theta \sim \tau$ hold if and only if $\mathrm{Dom}(\theta) = \mathrm{Dom}(\tau)$ and $\theta(l) \sim \tau(l)$ for every $l \in \mathrm{Dom}(\theta)$, and to configurations by letting $\langle M; \theta\rangle \sim \langle N; \tau\rangle$ hold if and only if $M \sim N$ and $\theta \sim \tau$.

Note that $\sim$ is reflexive, since the nullary constructors—variables, the unit value, and reference cells—are related to themselves by (S2), and that $\sim$ is symmetric, since (S1) is imposed for every pair of computations.

Lemma D.6. *The following statements hold.*

(1) If $M \sim N$, *then* M *and* N *have the same outermost constructor, and their corresponding immediate subterms are related by* $\sim$. *The same holds for* $V \sim W$ *whenever* V *is not a thunk; in particular, a value related to a reference cell* l *is* l *itself.*

(2) If $M \sim N$ *and* $V \sim W$, *then* $M[V/x] \sim N[W/x]$, *and likewise for values.*

(3) If $C \sim C'$ *and* $P \sim Q$, *where* P *and* Q *are terms of the sort that the holes expect, then* $C[P] \sim C'[Q]$.

Proof. (1) Since Rule (S1) relates only thunks, any other related pair must be derived by (S2). Thus (1) holds.

(2) By induction on the derivation of $M \sim N$. The case of (S2) is immediate from the induction hypothesis. In the case of (S1), since Ω contains no free variables,

$$\{\mathcal{L}[M_0]\}[V/x] \equiv \{\mathcal{L}[M_0[V/x]]\} \quad \text{and} \quad \{\mathcal{L}[N_0]\}[W/x] \equiv \{\mathcal{L}[N_0[W/x]]\},$$

and these two are again related by (S1).

(3) By induction on the derivation of $C \sim C'$. The case of (S2) is immediate from the induction hypothesis. The base case is either the one where both contexts are the hole, which is immediate, or the one where (S1) is applied. In the latter case, we have $C \equiv \{\mathcal{L}[C_0]\}$ and $C' \equiv \left\{\mathcal{L}\left[C_0'\right]\right\}$ for some contexts C_0 and C_0'; then $C[P]$ and $C'[Q]$ are again related by (S1). ◻

Lemma D.7. *Suppose $M \sim N$ and $M \equiv \mathcal{E}[R]$, where $\mathcal{E}$ is an evaluation context and R is a redex. Then $N \equiv \mathcal{E}'[R']$ for some evaluation context $\mathcal{E}'$ and redex R' such that $\mathcal{E} \sim \mathcal{E}'$, $R \sim R'$, and R and R' are redexes of the same rule.*

Proof. By induction on $\mathcal{E}$. In the base case, $M \equiv R$, and we take $\mathcal{E}' \stackrel{\text{def}}{=} [\,]$ and $R' \stackrel{\text{def}}{=} N$. That R' is a redex of the same rule as R follows from Lemma D.6(1): for instance, if $R \equiv V!$ with $V \equiv \{M_1\}$, then $R' \equiv W!$ with $V \sim W$, and W is again a thunk since a value related to a thunk is a thunk by either rule; if $R \equiv \mathbf{case}\ V\ \mathbf{of}\ \{(\mathbf{inj}_{\mathrm{L}_i}\ x_i \mapsto M_i)_i\}$ with $V \equiv \mathbf{inj}_{\mathrm{L}_k}\ V_0$, then R' is a variant matching against $\mathbf{inj}_{\mathrm{L}_k}\ W_0$ and whose branches are indexed by the same labels, so that the same branch is selected; and if $R \equiv \mathbf{get}\ l$, then $R' \equiv \mathbf{get}\ l$.

In the induction step, $\mathcal{E} \equiv \mathcal{F}[\mathcal{E}''[\,]]$ for a frame $\mathcal{F}$, and Lemma D.6(1) gives $N \equiv \mathcal{F}'[N'']$ with $\mathcal{F} \sim \mathcal{F}'$ and $\mathcal{E}''[R] \sim N''$, to which we apply the induction hypothesis. Thus, there exist some evaluation context $\mathcal{E}'$ and redex R' such that $N'' \equiv \mathcal{E}'[R']$, $\mathcal{E}'' \sim \mathcal{E}'$, $R \sim R'$, and R and R' are redexes of the same rule. It is immediate to check $\mathcal{F}[\mathcal{E}''[\,]] \sim \mathcal{F}'[\mathcal{E}'[\,]]$. Since the decomposition of a computation into an evaluation context and a redex is unique, $\mathcal{F}[\mathcal{E}'[R']]$ is the unique decomposition of N. ◻

Lemma D.8 (simulation). *Suppose $\langle M;\theta\rangle \sim \langle N;\tau\rangle$ and $\langle M;\theta\rangle \to_{\mathbf{R}} \langle M';\theta'\rangle$. Then one of the following two statements holds.*

(1) There exist a computation N' and a store τ' such that $\langle N;\tau\rangle \to_{\mathbf{R}} \langle N';\tau'\rangle$ and $\langle M';\theta'\rangle \sim \langle N';\tau'\rangle$.

(2) $M' \equiv \mathcal{E}[\mathcal{L}[M_0]]$ for some evaluation context $\mathcal{E}$ and computation M_0. In this case, the evaluation of $\langle M';\theta'\rangle$ does not terminate successfully by Lemma D.2.

Proof. Let $M \equiv \mathcal{E}[R]$ be the decomposition of M into an evaluation context and a redex. By Lemma D.7, $N \equiv \mathcal{E}'[R']$ with $\mathcal{E} \sim \mathcal{E}'$ and $R \sim R'$, where R and R' are redexes of the same rule. Since $\mathcal{E} \sim \mathcal{E}'$, when we prove (1), it suffices by Lemma D.6(3) to relate the two computations plugging the hole in order to relate the resulting configurations. We proceed by case analysis on this rule, showing the representative cases only.

- Case (U). Here, we have $R \equiv \{M_1\}!$ and $R' \equiv \{N_1\}!$. If $\{M_1\} \sim \{N_1\}$ is derived by (S2), then $M_1 \sim N_1$, and hence $\langle N;\tau\rangle \to_{\mathbf{R}} \langle \mathcal{E}'[N_1];\tau\rangle$ with $\mathcal{E}[M_1] \sim \mathcal{E}'[N_1]$; thus (1) holds. Otherwise, it is derived by (S1), so $M_1 \equiv \mathcal{L}[M_0]$ for some computation M_0, and therefore $M' \equiv \mathcal{E}[\mathcal{L}[M_0]]$; thus (2) holds. This is the only case where (2) arises.
- Case (F). Here, we have $R \equiv \mathbf{let}\ x = \mathbf{return}\,V\ \mathbf{in}\ M_1$ and $R' \equiv \mathbf{let}\ x = \mathbf{return}\,W\ \mathbf{in}\ N_1$, with $V \sim W$ and $M_1 \sim N_1$. By Lemma D.6(2), we have $M_1[V/x] \sim N_1[W/x]$, and hence $\mathcal{E}[M_1[V/x]] \sim \mathcal{E}'[N_1[W/x]]$. Since the stores are unchanged, (1) holds. The cases $(\times)$, $(+)$, $(\to)$, and $(\&)$ are analogous.
- Case (create). Here, we have $R \equiv \mathbf{create}\,V$ and $R' \equiv \mathbf{create}\,W$ with $V \sim W$. Since $\mathrm{Dom}(\theta) = \mathrm{Dom}(\tau)$, the same fresh label l is chosen in both reductions, and $\theta[l := V] \sim \tau[l := W]$ holds.
- Case (get). Here, we have $R \equiv \mathbf{get}\ l$ and $R' \equiv \mathbf{get}\ l$ by Lemma D.6(1). Since $\mathrm{Dom}(\theta) = \mathrm{Dom}(\tau)$, the side condition of the rule is satisfied on both sides, and $\theta(l) \sim \tau(l)$ yields $\mathcal{E}[\mathbf{return}\,\theta(l)] \sim \mathcal{E}'[\mathbf{return}\,\tau(l)]$. The case (set) is analogous.

Thus, we are done. ◻

Proof of Proposition D.4. Here C ranges over the contexts such that both $C[\{\mathcal{L}[M]\}]$ and $C[\{\mathcal{L}[N]\}]$ are **REF** programs. Since $\sim$ is symmetric, it suffices to show the "only if" direction.

We first show the following statement by induction on n: if $\langle M_1; \theta_1\rangle \sim \langle N_1; \tau_1\rangle$ and $\langle M_1; \theta_1\rangle \to^n_{\mathbf{R}} \langle \mathbf{return}\, V; \theta\rangle$,[22] then $\langle N_1; \tau_1\rangle \to^n_{\mathbf{R}} \langle \mathbf{return}\, W; \tau\rangle$ for some value W and store τ. If $n = 0$, then $M_1 \equiv \mathbf{return}\, V$, and by Lemma D.6(1), we have $N_1 \equiv \mathbf{return}\, W$ for some value W. If $n > 0$, let $\langle M_1; \theta_1\rangle \to_{\mathbf{R}} \left\langle M_1'; \theta_1'\right\rangle$ be the first step of the given reduction sequence, and we then apply Lemma D.8. Its case (2) cannot occur, since the evaluation of $\left\langle M_1'; \theta_1'\right\rangle$ does terminate successfully by the assumption $\left\langle M_1'; \theta_1'\right\rangle \to^{n-1}_{\mathbf{R}} \langle \mathbf{return}\, V; \theta\rangle$. Hence its case (1) applies, and the induction hypothesis yields the desired reduction sequence.

Now suppose that $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[M]\}])$ is defined. By (S1), the reflexivity of $\sim$, and Lemma D.6(3), we have

$$\langle C[\{\mathcal{L}[M]\}]; \emptyset\rangle \sim \langle C[\{\mathcal{L}[N]\}]; \emptyset\rangle,$$

and therefore the above statement implies that $\mathrm{Eval}_{\mathbf{REF}}(C[\{\mathcal{L}[N]\}])$ is defined. □

Proof of Theorem 5.5. Consider the following two $\mathbf{DEL}_{\mathrm{one}}$ computations:

$$M_i \stackrel{\text{def}}{=} \left\langle \left\{\mathcal{L}\left[\mathbf{S_0} k.\ \mathbf{return}\, (\mathbf{inj}_{\mathrm{L_i}}\, ())\right]\right\}! \middle| x.\mathbf{return}\, x \right\rangle \quad (i = 1, 2),$$

where $\mathcal{L} \stackrel{\text{def}}{=} (\mathbf{let}\ \_ = [\ ]\ \mathbf{in}\ \Omega)$.

Lemma D.9. $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}\left(\left\langle \left\{\mathbf{let}\ \_ = \mathbf{S_0} k.\ \mathbf{return}\, (\mathbf{inj}_{\mathrm{L_i}}\, ())\ \mathbf{in}\ \Omega\right\}! \middle| x.\mathbf{return}\, x\right\rangle\right) = \mathbf{inj}_{\mathrm{L_i}}\, ()$.

Proof. By routine calculation. □

In $\mathbf{DEL}_{\mathrm{one}}$, these results can be distinguished by the following context $\mathcal{D}_i$:

$$\mathcal{D}_i \stackrel{\text{def}}{=} \left(\begin{array}{l} \mathbf{let}\ r = [\ ]\ \mathbf{in} \\ \mathbf{case}\ r\ \mathbf{of}\ \{ \\ \quad \mathbf{inj}_{\mathrm{L_i}}\ \_ \mapsto \mathbf{return}\, () \\ \quad \_ \mapsto \Omega \\ \} \end{array}\right).$$

Note that $\mathcal{D}_i$ consists only of **MAM** constructs; thus any weak macro-translation preserves it homomorphically.

Suppose that there exists a weak macro-translation from $\mathbf{DEL}_{\mathrm{one}}$ to **REF**. Then, the translation $\underline{M_i}$ can be written as

$$\mathcal{A}_{\mathrm{dollar},x}(\left\{\mathcal{L}\left[\mathcal{A}_{\mathrm{shift0},k}(\mathbf{return}\, (\mathbf{inj}_{\mathrm{L_i}}\, ()))\right]\right\}!, \mathbf{return}\, x),$$

where $\mathcal{A}_{\mathrm{dollar},x}$ and $\mathcal{A}_{\mathrm{shift0},k}$ are fixed general contexts corresponding to the translation of a source dollar which binds x in its additional part and a source shift0 which binds k in its body, respectively. Let C denote the context $\mathcal{A}_{\mathrm{dollar},x}([\ ]!, \mathbf{return}\, x)$ and N_i be the computation $\mathcal{L}\left[\mathcal{A}_{\mathrm{shift0},k}(\mathbf{return}\, (\mathbf{inj}_{\mathrm{L_i}}\, ()))\right]$; then, $\underline{M_i}$ can be written as $C[\{N_i\}]$. By Lemma D.2, the evaluation of N_i under any evaluation context and any store does not terminate successfully.

Lemma D.10. $\mathrm{Eval}_{\mathbf{REF}}\left(\underline{M_i}\right)$ *is defined, and it is of the form* $\mathbf{inj}_{\mathrm{L_i}}\, V_i$ *for some* **REF** *value* V_i.

[22] $C \to^n D$ means that C is reduced to D in exactly n-steps.

Proof. By Lemma D.9, both $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(M_i)$ and $\mathrm{Eval}_{\mathbf{DEL}_{\mathrm{one}}}(\mathcal{D}_i[M_i])$ are defined. Since $\underline{\cdot}$ is a weak macro-translation, both $\mathrm{Eval}_{\mathbf{REF}}\left(\underline{M_i}\right)$ and $\mathrm{Eval}_{\mathbf{REF}}\left(\underline{\mathcal{D}_i[M_i]}\right)$ are defined. Since $\mathcal{D}_i$ is a **MAM** context, and $\underline{\cdot}$ is homomorphic on **MAM** constructs,

$$\underline{\mathcal{D}_i[M_i]} \equiv \mathcal{D}_i[\underline{M_i}].$$

Thus $\mathrm{Eval}_{\mathbf{REF}}\left(\mathcal{D}_i[\underline{M_i}]\right)$ is defined, which implies that $\mathrm{Eval}_{\mathbf{REF}}\left(\underline{M_i}\right)$ is of the form $\mathbf{inj}_{\mathrm{L_i}}\ V_i$ for some **REF** value V_i. □

Thus, the evaluation of $C[\{N_1\}]$ and $C[\{N_2\}]$ yields $\mathbf{inj}_{\mathrm{L}_1}\ V_1$ and $\mathbf{inj}_{\mathrm{L}_2}\ V_2$, respectively, so $\{N_1\}$ and $\{N_2\}$ are observationally inequivalent. However, this contradicts Proposition D.4. Therefore, there is no weak macro-translation from $\mathbf{DEL}_{\mathrm{one}}$ to **REF**. □