A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

Dariusz Biernacki
University of Wrocław, Poland
dabi@cs.uni.wroc.pl

Sergueï Lenglet
Université de Lorraine, Nancy, France
serguei.lenglet@univ-lorraine.fr

Piotr Polesiuk
University of Wrocław, Poland
ppolesiuk@cs.uni.wroc.pl

Abstract

We present a complete coinductive syntactic theory for an untyped calculus of algebraic operations and handlers, a relatively recent concept that augments a programming language with unprecedented flexibility to define, combine and interpret computational effects. Our theory takes the form of a normal-form bisimilarity and its soundness w.r.t. contextual equivalence hinges on using so-called context variables to test evaluation contexts comprising normal forms other than values. The theory is formulated in purely syntactic elementary terms and its completeness demonstrates the discriminating power of handlers. It crucially takes advantage of the clean separation of effect handling code from effect raising construct, a distinctive feature of algebraic effects, not present in other closely related control structures such as delimited-control operators.

2012 ACM Subject Classification Theory of computation → Control primitives

Keywords and phrases algebraic effect, handler, behavioral equivalence, bisimilarity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.7

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments on the presentation of this work.

1 Introduction

Algebraic effects with handlers [22, 3] have become a popular technique of programming with computational effects such as exceptions, mutable state or nondeterminism. Their strength lies in their modularity, as it is possible to easily combine several effects thanks to the separation between syntax and semantics. Indeed, effects themselves are just syntactic constructs which do not carry any meaning; their semantics is given by the handlers, which come into play when an interpretation of an effect is needed for the computation to go through.

As an informal example, borrowed from [8], consider the reader effect ask, which returns a hidden value when triggered. An effect is used as a labeled operation, e.g., as in $\mathsf{do}_{\mathsf{ask}}\,() + \mathsf{do}_{\mathsf{ask}}\,() + 2$, and its meaning is given by a handler, as in

$$\mathsf{handle}\ \mathsf{do}_{\mathsf{ask}}\,() + \mathsf{do}_{\mathsf{ask}}\,() + 2\ \{\mathsf{ask}\colon x,k \to k\ 5;\ \mathsf{ret}\ y \to y\}$$

The handler specifies how it interprets the ask effect by the expression $x,k \to k\ 5$, where x stands for the value the effect operation is applied to (which is not used in this example), and k for its *continuation* or *resumption*, i.e., the rest of the computation, which includes the handler itself. Here, the handler simply passes 5 to the continuation, so that $\mathsf{do}_{\mathsf{ask}}\,() + \mathsf{do}_{\mathsf{ask}}\,() + 2$

eventually reduces to 12. Once the expression inside the handler is a value, it is passed to the *return clause* $\mathsf{ret}\ y \to y$, which in our case simply returns the result. Any expression can be used in an effect handler, including one making use of the continuation several times or not at all; for example, in

$$\mathsf{handle}\ \mathsf{do}_{\mathsf{ask}}\ () + \mathsf{do}_{\mathsf{ask}}\ () + 2\ \{\mathsf{ask}\colon x,k \to 13; \mathsf{ret}\ y \to y\}$$

the handler throws away the continuation when called the first time and returns 13, which is then the final result of the computation. Multiple effects can be used in an expression, which are then interpreted by a single handler, or by successive handlers enclosing the expression. The order of the handlers then specifies the semantics of all the effects combined.

While handlers make combining multiple effects programmer friendly, reasoning about the behavior of programs with effects and handlers appears to be inherently challenging, mainly due to the non-local transfer of control involved in effect handling. When it comes to the issue of program equivalence, the standard notion considered in calculi modeling programming languages, typically based on λ-calculi, is *contextual equivalence* [20], which requires program phrases to behave the same when plugged in any context. The quantification over all contexts makes this relation hard to use in practice, so one usually looks for more tractable characterizations of contextual equivalence, either in the form of logical relations [24] or coinductively defined bisimilarities [1, 17, 27].

In the presence of algebraic effects and handlers, the situation is even more interesting, because we have to take into account the possibility that the testing context may interpret any non-handled effects the two programs being tested might use. There exist some works on formal techniques for reasoning about program equivalence in calculi with algebraic effects, but they either do not include handlers in the language [16, 15, 14] or are directed by a type structure of the calculus [8] (we discuss related work in detail in Section 4). None of them, however, focuses on the control structure of a full calculus of algebraic effects and handlers (where effects are interpreted dynamically, unlike, e.g., in [14]) and in isolation from other concepts such as types. Algebraic effects are intimately related to delimited-control operators [12, 21], for which bisimulation theories have been studied extensively [4], yet they differ in a very essential way, as we argue in this work.

In this paper, we show that it is possible to characterize contextual equivalence in an untyped calculus with algebraic effects and handlers with one of the simplest notions of equivalence, namely *normal-form* (or *open*) bisimilarity [25, 17]. In a normal-form bisimilarity proof one compares open terms by reducing them to normal forms, which are then decomposed into bisimilar subterms. In a language with algebraic effects, we have to consider extra normal forms – programs with effects that have not been handled. More importantly, we have to observe how a context may handle an effect and its continuation. To this end, we introduce an extended calculus where contexts can be abstractly represented with *context variables*, a concept we used in our previous work on normal-form bisimulations for abortive continuations [7]. Such variables can be observed and discriminated upon by the bisimilarity that is defined for the extended calculus. Extending the calculus is a critical step in obtaining sound and complete bisimilarity, but it should be seen just as a tool for studying the plain calculus. When restricted to the plain calculus, the bisimilarity relates exactly those terms that are equivalent w.r.t. the contextual equivalence in the plain calculus.

In many calculi, the decomposition of normal forms as done in normal-form bisimilarity is usually too fine-grained and distinguishes programs that are in fact contextually equivalent [17]. The result of this paper shows that handlers contain sufficient discriminating power for normal-form bisimilarity to be complete w.r.t. contextual equivalence. It contrasts with other continuation-manipulating constructs such as (multi-prompted) delimited-control operators, for which finding a complete normal-form bisimilarity remains an open issue [4].

$$
\begin{array}{rcll}
\mathrm{Lbl} \ni l & & & \text{(effect labels)} \\
\mathrm{Var} \ni f,k,x,y,z & & & \text{(variables)} \\
\mathrm{Val} \ni u,v,w & ::= & x \mid \lambda x.e & \text{(values)} \\
\mathrm{Exp} \ni e & ::= & v \mid e_0\, e_1 \mid \mathsf{do}_l\, e \mid \mathsf{handle}\ e\,\{H;r\} & \text{(expressions)} \\
H & ::= & l_1 : h_1; \ldots ; l_n : h_n & \\
h & ::= & x,k \to e & \text{(effect handlers)} \\
r & ::= & \mathsf{ret}\ x \to e & \text{(return clause)}
\end{array}
$$

Figure 1 Syntax of λ_{eff}.

The rest of this paper is organized as follows. In Section 2, we present the syntax, semantics, and contextual equivalence of the plain calculus λ_{eff}, the minimal calculus with effects and handlers we consider for our study. In Section 3, we define the normal-form bisimilarity for the extended calculus and prove its soundness and completeness. We also define *up-to techniques*, proof techniques meant to simplify equivalence proofs, and we illustrate how the bisimilarity and these techniques can be used on examples. Additionally, we pinpoint the difference between algebraic effects and delimited-control operators and how it affects the definition of a normal-form bisimulation. In Section 4, we discuss related work, and we conclude in Section 5. The appendix contains the soundness and completeness proof sketches.

2 The Calculus λ_{eff}

Syntax. The calculus λ_{eff}, whose syntax is given in Figure 1, extends the λ-calculus with labeled effects $\mathsf{do}_l\, e$ and handlers $\mathsf{handle}\ e\,\{H;r\}$, where H is a list of effect handlers $l_i : x_i,k_i \to e_i$ and r is a return clause $\mathsf{ret}\ x \to e'$. The order of the list is irrelevant, but we assume the labels $l_1 \ldots l_n$ to be pairwise distinct. In a handler $x_i,k_i \to e_i$, the variable x_i represents the argument of the effect, while k_i stands for its continuation (or *resumption*). We write $\mathsf{lbl}(e)$ for the set of effect labels l that label do expressions in e. The choice of having a handler interpret several effects at once makes writing examples easier, but does not affect the behavioral theory: the definitions of the equivalences are the same if the handler takes care of one effect only.

An effect handler $x_i,k_i \to e_i$ binds x_i and k_i in e_i, and a λ-abstraction $\lambda x.e$ or a return clause $\mathsf{ret}\ x \to e$ bind x in e. We use the standard notions of free variables ($\mathsf{fv}(e)$ is the set of free variables in e), closed and open expressions, and we work modulo α-conversion of the bound variables. A variable is called fresh if it does not occur in any of the entities under consideration.

We assume the standard call-by-value Church encoding of natural numbers, booleans (true, false, $\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$), unit ($()$), and the sequence expression $(e_1; e_2)$ that we use in examples and in the proof of completeness.

Reduction semantics. We fix a call-by-value, left-to-right reduction strategy for λ_{eff} by defining the syntax of evaluation contexts as follows.

$$\mathrm{ECtx} \ni E \quad ::= \quad \Box \mid E\, e \mid v\, E \mid \mathsf{do}_l\, E \mid \mathsf{handle}\ E\,\{H;r\}$$

We write $E[e]$ for the plugging of the expression e into the context E, and $e\{v/x\}$ for the usual capture-avoiding substitution of x by v in e. Given a context E, we define the set of effects it handles, written $\mathrm{hl}(E)$, as follows.

$$\begin{aligned}
\mathrm{hl}(\Box) &\triangleq \emptyset \\
\mathrm{hl}(E\,e) &\triangleq \mathrm{hl}(E) \\
\mathrm{hl}(v\,E) &\triangleq \mathrm{hl}(E) \\
\mathrm{hl}(\mathsf{do}_l\,E) &\triangleq \mathrm{hl}(E) \\
\mathrm{hl}(\mathsf{handle}\,E\,\{l_1\colon h_1;\ldots;l_n\colon h_n;r\}) &\triangleq \mathrm{hl}(E)\cup\{l_1,\ldots,l_n\}
\end{aligned}$$

When writing expressions, we sometimes decorate a context with a label it does not handle, i.e., writing $E^{\bar{l}}$ if $l \notin \mathrm{hl}(E)$. Typically, we write $E^{\bar{l}}[\mathsf{do}_l\,v]$ for an expression where the effect l cannot be handled by E.

The reduction semantics of λ_{eff} is given by the following rules.

$$\begin{array}{rcll}
(\lambda x.e)\,v & \mapsto & e\{v/x\} & \\
\mathsf{handle}\,v\,\{H;\mathsf{ret}\,x\to e\} & \mapsto & e\{v/x\} & \\
E[\mathsf{do}_l\,v] & \mapsto & e\{v/x\}\{\lambda z.E[z]/k\} & \text{if } E=\mathsf{handle}\,E'^{\bar{l}}\,\{H;r\} \\
 & & & \text{and } l\colon x,k\to e\in H \\
 & & & \text{and } z \text{ is fresh} \\
E[e] & \to & E[e'] & \text{if } e\mapsto e'
\end{array}$$

We write $\to^*$ for the reflexive and transitive closure of $\to$. In the third rule, we see that the effect $\mathsf{do}_l\,v$ is interpreted by the first enclosing handler, as $E=\mathsf{handle}\,E'^{\bar{l}}\,\{H;r\}$ and E' does not handle l. The handler has access not only to the argument v of the effect, but also to its continuation, represented as a function $\lambda z.E[z]$. Note that the handler itself is part of the captured continuation, meaning that it can handle further effects when the continuation is resumed.[1] If a handler obtains a value (second rule), there are no more effects to handle and the value is passed to the return clause. The semantics is deterministic, as it can be shown that an expression is either a normal form or can be uniquely decomposed into a redex and an evaluation context.

▶ **Example 1.** Let us consider the example from the introduction:

$$e \triangleq \mathsf{handle}\,\mathsf{do}_{\mathsf{ask}}\,() + \mathsf{do}_{\mathsf{ask}}\,() + 2\,\{\mathsf{ask}\colon x,k\to k\,5;\mathsf{ret}\,y\to y\}$$

If $E \triangleq \mathsf{handle}\,\Box\,\{\mathsf{ask}\colon x,k\to k\,5;\mathsf{ret}\,y\to y\}$ and z is a fresh variable, then e reduces as follows:

$$\begin{aligned}
e &\to (\lambda z.E[z+\mathsf{do}_{\mathsf{ask}}\,()+2])\,5 \\
&\to \mathsf{handle}\,5+\mathsf{do}_{\mathsf{ask}}\,()+2\,\{\mathsf{ask}\colon x,k\to k\,5;\mathsf{ret}\,y\to y\} \\
&\to (\lambda z.E[5+z+2])\,5 \\
&\to \mathsf{handle}\,5+5+2\,\{\mathsf{ask}\colon x,k\to k\,5;\mathsf{ret}\,y\to y\} \\
&\to^* \mathsf{handle}\,12\,\{\mathsf{ask}\colon x,k\to k\,5;\mathsf{ret}\,y\to y\} \\
&\to 12
\end{aligned}$$

[1] Such handlers are known as *deep* handlers as opposed to *shallow* handlers also considered in the literature [21].

Normal forms and contextual equivalence. When considering open expressions, normal forms can be of the following kinds.

▶ **Lemma 2.** *An open expression e is a normal form iff e is a value, or $e = E[x\,v]$ for some E, x, and v, or $e = E^{\bar{l}}[\mathsf{do}_l\,v]$ for some E, l, and v.*

Values and expressions $E[x\,v]$ (referred to as *open-stuck terms*) are usual normal forms which can already be found in the plain λ-calculus. The expression $E^{\bar{l}}[\mathsf{do}_l\,v]$ cannot reduce further, as E cannot handle the effect l; we refer to such normal forms as *control-stuck terms*. Closed normal forms are either λ-abstractions or control-stuck terms.

Contextual equivalence equates expressions behaving the same in all contexts. In the presence of multiple closed normal forms as in λ_{eff}, several definitions of contextual equivalence are possible, depending on whether we observe termination of evaluation in general, or to specific, meaningful normal forms – usually values. It turns out that such a choice does not matter in λ_{eff}, as the definitions coincide; we explain why after presenting the definition we use in this paper. We let C range over arbitrary contexts, i.e., expressions with a hole $\Box$. We write $e \Downarrow_{\mathrm{v}}$ if there is a value v, such that $e \to^* v$, and $e \Uparrow$ if e reduces infinitely, e.g., $\Omega \Uparrow$, where $\Omega = (\lambda x.x\,x)\,(\lambda x.x\,x)$.

▶ **Definition 3.** *Two expressions e_1 and e_2 are contextually equivalent, written $e_1 \equiv e_2$, if for all contexts C, such that $C[e_1]$ and $C[e_2]$ are closed, we have $C[e_1] \Downarrow_{\mathrm{v}}$ iff $C[e_2] \Downarrow_{\mathrm{v}}$.*

It can be shown that this definition introduces the same notion of contextual equivalence as the one in which we observe simply termination of evaluation, instead of evaluation to a value. The reason is that for any control-stuck term $e_1 = E_1^{\bar{l}}[\mathsf{do}_l\,v_1]$, taking

$$C = \mathsf{handle}\ \Box\ \{l\colon x,k \to \Omega;\ \mathsf{ret}\ x \to x\}$$

we have $C[e_1] \Uparrow$, whereas $C[v_2] \Downarrow_{\mathrm{v}}$ for any value v_2, and taking

$$C' = \mathsf{handle}\ \Box\ \{l\colon x,k \to x;\ \mathsf{ret}\ x \to x\}$$

we have $C'[e_1] \Downarrow_{\mathrm{v}}$, whereas $C'[e_2] \Uparrow$ for any e_2 such that $e_2 \Uparrow$. Thus, we can always build a context that preserves non-termination and evaluation to a value, but that at the same time coerces a control-stuck term to either a non-terminating expression (C) or to a value (C'). The two contextual equivalences therefore coincide, a situation which differs from other context-manipulating constructs such as delimited-control operators [4].

3 Normal-Form Bisimilarity

We first informally introduce our notion of normal-form bisimilarity, before giving its definition and discussing its soundness and completeness. We also explain why, in spite of the relationship between handlers and multi-prompted delimited continuations, it is more difficult to define a complete normal-form bisimilarity for the latter than for the former.

3.1 Informal Presentation

Normal-form bisimulation reduces expressions to normal forms and decomposes them into related subterms; for example, an open-stuck term $E_1[x\,v_1]$ is related to e_2 if e_2 reduces to a similar term such that the contexts and values are pairwise related. Compared to the plain λ-calculus [17, 7], we have to consider an extra normal form – control-stuck terms – but also take into account the fact that contexts may handle effects.

Dealing with control-stuck terms follows the same logic as for open-stuck terms: $E_1{}^{\bar{l}}[\mathsf{do}_l\, v_1]$ is related to e_2 if e_2 reduces to a control-stuck term with related values and contexts. Comparing contexts requires more care, as it depends how they are used. A context $E_1{}^{\bar{l}}$ surrounding a control-stuck term can only be captured and then plugged with a value, so it is enough to test them with a fresh variable representing that value. Such contexts represent resumptions (delimited continuations, really) that are bound to the continuation variable k in effect handlers and used to obtain suitable interpretation of the effect.

In contrast, in an open-stuck term $E_1[x\, v_1]$, the application may reduce to an effect which could be handled by E_1. Testing such contexts with only a fresh variable is not enough as it would relate $\Box$ and $\mathsf{handle}\ \Box\, \{H; \mathsf{ret}\ x \to x\}$, two contexts which behave differently as soon as they are plugged with an effect handled by H. We need to observe which handlers are surrounding the context holes, but without requiring the sequence of handled effects to be exactly the same. Indeed, successive "identity handlers" $h \triangleq x,k \to k\, x$ should be related if they handle the same effects, even in a different order: the context $\mathsf{handle}\ \mathsf{handle}\ \Box\, \{l_2 : h; \mathsf{ret}\ x \to x\}\, \{l_1 : h; \mathsf{ret}\ x \to x\}$ is expected to be equivalent to the context $\mathsf{handle}\ \mathsf{handle}\ \Box\, \{l_1 : h; \mathsf{ret}\ x \to x\}\, \{l_2 : h; \mathsf{ret}\ x \to x\}$.

A simple way to compare the handlers behaviors is to plug the contexts with a control-stuck term $\mathsf{do}_l\, x$ for a fresh x and for any l (handled by the contexts). However, such a testing term is not strong enough, as it would relate a handler which throws away the continuation to one that does not, e.g., $E_1 = \mathsf{handle}\ \Box\, \{l : x,k \to x; \mathsf{ret}\ x \to x\}$ and $E_2 = \mathsf{handle}\ \Box\, \{l : x,k \to k\, x; \mathsf{ret}\ x \to x\}$. We need to account for the fact that control-stuck terms may be surrounded with a context without introducing a quantification over these contexts which would go against the principles behind normal-form bisimulation. We do so by extending the syntax of the calculus with *context variables*, a construct we introduced in previous works to track the whereabouts of contexts captured by control operators [7, 4]. In a control-stuck term $\alpha^{\bar{l}}[\mathsf{do}_l\, x]$, the context variable $\alpha^{\bar{l}}$ stands for a context which does not handle l, and its presence allows to distinguish between the two contexts E_1 and E_2.

Adding context variables to λ_{eff} generates new normal forms of the shape $E[\alpha^{\bar{l}}[v]]$ and $E[\alpha^{\bar{l}}[E'^{\overline{l'}}[\mathsf{do}_{l'}\, v]]]$ (with $l \neq l'$), where the computation is stuck because we do not know which context $\alpha^{\bar{l}}$ stands for. The bisimulation deals with these normal forms in a very regular way, simply asking to reduce to a normal form of the same shape with related contexts and values. In the end, the definition we obtain (Definition 5) follows the usual pattern of normal-form bisimulation – the only subtlety being in how to compare contexts – and yet the resulting bisimilarity is sound and complete w.r.t. the contextual equivalence of the extended calculus. More importantly, the restriction of the bisimilarity to plain calculus terms yields the contextual equivalence for the plain calculus.

3.2 Extended Calculus

As explained in the previous section, we extend the syntax of λ_{eff} with context variables in order to observe how contexts are captured when effects are triggered. We assume a set CVar of context variables, ranged over by α and β. Similar to evaluation contexts, we decorate these variables with an effect it does not handle: the variable $\alpha^{\bar{l}}$ is a context variable standing for a context which does not handle l. In particular, when considering a control-stuck term, the context variable is always decorated with an effect label. Moreover, we write $\alpha^{\bar{l}} \neq \beta^{\overline{l'}}$ if $l \neq l'$ or $\alpha \neq \beta$.

We extend the syntax of expressions and evaluation contexts as follows.

$$e ::= \ldots \mid \alpha^{\bar{l}}[e] \qquad\qquad E ::= \ldots \mid \alpha^{\bar{l}}[E]$$

We write $\mathsf{cv}(e)$ for the set of context variables occurring in e. We adapt the definition of hl so that $\mathrm{hl}(\alpha^{\bar{l}}[E]) \triangleq (\mathrm{Lbl} \setminus \{l\}) \cup \mathrm{hl}(E)$, as $\alpha^{\bar{l}}$ stands for a context not handling l but which may potentially handle any other label. While the reduction rules themselves are the same, the semantics of the extended calculus is still affected by the change in the grammar of evaluation contexts. In particular, it admits more normal forms than the plain λ_{eff}.

▶ **Lemma 4.** *An open expression e is a normal form in the extended calculus iff e is a value, or $e = E[x\ v]$ for some E, x, and v, or $e = E^{\bar{l}}[\mathsf{do}_l\ v]$ for some E, l, and v, or $e = E[\alpha^{\bar{l}}[v]]$ for some E, $\alpha^{\bar{l}}$, and v, or $e = E_1[\alpha^{\bar{l}}[E_2^{\ \overline{l'}}[\mathsf{do}_{l'}\ v]]]$ for some E_1, E_2, v, $\alpha^{\bar{l}}$ and l' such that $l \neq l'$.*

We refer to normal forms of the shape $E[\alpha^{\bar{l}}[v]]$ as *context-stuck terms* and those of the shape $E_1[\alpha^{\bar{l}}[E_2^{\ \overline{l'}}[\mathsf{do}_{l'}\ v]]]$ as *control/context-stuck terms*. The latter differ from control-stuck terms of the form $E^{\bar{l}}[\mathsf{do}_l\ v]$, because $\alpha^{\bar{l}}$ may be replaced by a context handling l', so even if E_1 does not handle l' we cannot consider $E_1[\alpha^{\bar{l}}[E_2^{\ \overline{l'}}]]$ as a context not handling l'.

A context variable cannot be bound, therefore an open term may contain context variables or free expression variables. In contrast, an expression or context is closed if it does not have any context variable or free expression variable.

Given an expression e, a context variable $\alpha^{\bar{l}}$ and a context $E^{\bar{l}}$, we define the *context substitution* $e\{E^{\bar{l}}/\alpha^{\bar{l}}\}$ so that $(\alpha^{\bar{l}}[e])\{E^{\bar{l}}/\alpha^{\bar{l}}\} \triangleq E^{\bar{l}}[e\{E^{\bar{l}}/\alpha^{\bar{l}}\}]$, and the substitution is recursively propagated to the sub-expressions in the other cases.

3.3 Definition

We define the bisimulation for the extended calculus using the notion of *diacritical progress* we developed in a previous work [2, 6], which distinguishes between *active* and *passive* clauses. Roughly, passive clauses are between simulation states which should be considered equal, while active clauses are between states where actual progress is taking place. This distinction does not change the notions of bisimulation or bisimilarity, but it simplifies the soundness proof of the bisimilarity. It also allows for the definition of powerful *up-to techniques*, functions on relations meant to simplify bisimilarity proofs. For normal-form bisimilarity, our framework enables up-to techniques which respect η-expansion [7], a necessary condition to reach completeness.

Given a relation $\mathcal{R}$ on expressions, we extend it to values and evaluation contexts in the following way.

$$\frac{v_1\ x\ \mathcal{R}\ v_2\ x \qquad x \text{ fresh}}{v_1\ \mathcal{R}^{\mathsf{v}}\ v_2} \qquad\qquad \frac{E_1[x]\ \mathcal{R}\ E_2[x] \qquad x \text{ fresh}}{E_1\ \mathcal{R}^{\mathsf{r}}\ E_2}$$

$$\frac{E_1[x]\ \mathcal{R}\ E_2[x] \qquad \forall l \in \mathrm{hl}(E_1) \cup \mathrm{hl}(E_2).E_1[\alpha^{\bar{l}}[\mathsf{do}_l\ x]]\ \mathcal{R}\ E_2[\alpha^{\bar{l}}[\mathsf{do}_l\ x]] \qquad x, \alpha^{\bar{l}} \text{ fresh}}{E_1\ \mathcal{R}^{\mathsf{c}}\ E_2}$$

The $\cdot^{\mathsf{v}}$ extension compares values by simply applying them to a fresh variable; such a test, compliant with η-expansion [7], is valid because λ-abstractions are the only values of our language. As explained in Section 3.1, we consider two extensions for evaluation contexts, as it depends how these are used: $\cdot^{\mathsf{r}}$ is used when we know the contexts are plugged only with values (resumptions), while $\cdot^{\mathsf{c}}$ assumes that they can be filled with any expression, including an effectful one. As a result, $\cdot^{\mathsf{c}}$ compares how the contexts deal with the effects they may handle (the ones in $\mathrm{hl}(E_1) \cup \mathrm{hl}(E_2)$), by testing them with an expression $\alpha^{\bar{l}}[\mathsf{do}_l\ x]$ built using a fresh context variable $\alpha^{\bar{l}}$ which can be observed during the bisimulation game.

We define progress, bisimulation and bisimilarity using these extensions.

▶ **Definition 5.** *A relation $\mathcal{R}$ progresses to $\mathcal{S}$, $\mathcal{T}$ written $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{S} \subseteq \mathcal{T}$, and $e_1\ \mathcal{R}\ e_2$ implies:*

- *if $e_1 \to e_1'$, then there exists e_2' such that $e_2 \to^* e_2'$ and $e_1'\ \mathcal{T}\ e_2'$;*
- *if $e_1 = v_1$, then there exists v_2 such that $e_2 \to^* v_2$ and $v_1\ \mathcal{S}^{\mathsf{v}}\ v_2$;*
- *if $e_1 = E_1[x\ v_1]$, then there exist E_2 and v_2 such that $e_2 \to^* E_2[x\ v_2]$, $E_1\ \mathcal{T}^{\mathsf{c}}\ E_2$, and $v_1\ \mathcal{T}^{\mathsf{v}}\ v_2$;*
- *if $e_1 = {E_1}^{\overline{l}}[\mathsf{do}_l\ v_1]$, then there exist E_2 and v_2 such that $e_2 \to^* {E_2}^{\overline{l}}[\mathsf{do}_l\ v_2]$, ${E_1}^{\overline{l}}\ \mathcal{T}^{\mathsf{r}}\ {E_2}^{\overline{l}}$, and $v_1\ \mathcal{T}^{\mathsf{v}}\ v_2$;*
- *if $e_1 = E_1[\alpha^{\overline{l}}[v_1]]$, then there exist E_2 and v_2 such that $e_2 \to^* E_2[\alpha^{\overline{l}}[v_2]]$, $E_1\ \mathcal{S}^{\mathsf{c}}\ E_2$, and $v_1\ \mathcal{S}^{\mathsf{v}}\ v_2$;*
- *if $e_1 = E_1[\alpha^{\overline{l}}[{E_1'}^{\overline{l'}}[\mathsf{do}_{l'}\ v_1]]]$ with $l \neq l'$, then there exist E_2, ${E_2'}^{\overline{l'}}$, and v_2 such that $e_2 \to^* E_2[\alpha^{\overline{l}}[{E_2'}^{\overline{l'}}[\mathsf{do}_{l'}\ v_2]]]$, $E_1\ \mathcal{T}^{\mathsf{c}}\ E_2$, ${E_1'}^{\overline{l'}}\ \mathcal{T}^{\mathsf{r}}\ {E_2'}^{\overline{l'}}$, and $v_1\ \mathcal{T}^{\mathsf{v}}\ v_2$;*
- *the symmetric of the above conditions on e_2.*

A normal-form bisimulation is a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{R}, \mathcal{R}$, and normal-form bisimilarity $\approx$ is the union of all normal-form bisimulations.

As pointed out before, the clauses dealing with normal forms are very similar, simply requiring e_2 to reduce to a normal form of the same kind, and then decomposing these normal forms into pairwise related subterms. We just have to be careful in using $\cdot^{\mathsf{r}}$ only for the contexts used as resumptions.

We progress towards $\mathcal{S}$ in the value and context-stuck term clauses and $\mathcal{T}$ in the others; the former are passive while the latter are active. Our framework prevents some up-to techniques from being applied after a passive transition. For values, we want to forbid the application of bisimulation up to context as it would be unsound: we could deduce that $v_1\ x$ and $v_2\ x$ are equivalent for all v_1 and v_2 just by building a candidate relation containing v_1 and v_2. Similarly, for context-stuck terms, we prevent the application of bisimulation up to substitution of context variables, as we could also relate any v_1 and v_2 from a candidate containing $\alpha^{\overline{l}}[v_1]$ and $\alpha^{\overline{l}}[v_2]$ by replacing the context variable with $\square\ x$.

▶ **Example 6.** We consider the handler of Example 1 for the reader effect, where we generalize the hidden value 5 to a given variable z:

$$E_1 \triangleq \mathsf{handle}\ \square\ \{\mathsf{ask}\colon x,k \to k\ z; \mathsf{ret}\ x \to x\}$$

Alternatively, the reader effect can be interpreted by the following handler obtained from the standard handler for mutable state:

$$E_2 \triangleq (\mathsf{handle}\ \square\ \{\mathsf{ask}\colon x,k \to \lambda y.k\ y\ y; \mathsf{ret}\ x \to \lambda y.x\})\ z$$

The context E_2 applies the handler to the current value of the state and let the handling code of the operation(s) access it through a λ-abstraction. (We would obtain a standard handler for mutable state by adding the clause $\mathsf{set}\colon x,k \to \lambda y.k\ y\ x$ handling the operation set which sets the value of the state.)

We show that these two handlers for the reader effect are equivalent by establishing the equivalence between the contexts $E_1 \approx^{\mathsf{c}} E_2$.

Proof. Relating $E_1[x]$ and $E_2[x]$ for a fresh x is easy, as $E_1[x] \to x$ and $E_2[x] \to (\lambda y.x)\ z \to x$. Testing with $\alpha^{\bar{l}}[\mathsf{do}_l\ x]$ and defining $E_2' \triangleq \mathsf{handle}\ \square\ \{l\colon x,k \to \lambda y.k\ y\ y; \mathsf{ret}\ x \to \lambda y.x\}$, we get

$$E_1[\alpha^{\bar{l}}[\mathsf{do}_l\ x]] \to^2 E_1[\alpha^{\bar{l}}[z]]$$
$$E_2[\alpha^{\bar{l}}[\mathsf{do}_l\ x]] \to (\lambda y.(\lambda y'.E_2'[\alpha^{\bar{l}}[y']])\ y\ y)\ z \to^2 E_2'[\alpha^{\bar{l}}[z]]z = E_2[\alpha^{\bar{l}}[z]]$$

We obtain two context-stuck terms, for which we need to relate identical variables and the contexts E_1 and E_2 we want to equate in the first place. In the end, we can easily build a bisimulation $\mathcal{R}$ such that $E_1\ \mathcal{R}^{\mathsf{c}}\ E_2$. ◀

3.4 Soundness and Up-to Techniques

In our framework [6] as in the works we extend [18, 23], proving that the bisimilarity is *compatible* – preserved by contexts – amounts to showing that a form of bisimulation up to context is valid, as explained after Lemma 10. We slightly reformulate our most recent work [6] to make it simpler but expressive enough it can be applied to λ_{eff}.

In what follows, we use s, f, g to range over *monotone* functions on relations, i.e., functions such that $\mathcal{R} \subseteq \mathcal{S}$ implies $\mathsf{f}(\mathcal{R}) \subseteq \mathsf{f}(\mathcal{S})$ for any $\mathcal{R}$, $\mathcal{S}$. We extend $\cup$ to functions so that for all $\mathcal{R}$, $(\mathsf{f} \cup \mathsf{g})(\mathcal{R}) = \mathsf{f}(\mathcal{R}) \cup \mathsf{g}(\mathcal{R})$. We define an ordering $\sqsubseteq$ on functions so that $\mathsf{f} \sqsubseteq \mathsf{g}$ if for all $\mathcal{R}$, $\mathsf{f}(\mathcal{R}) \subseteq \mathsf{g}(\mathcal{R})$, which is itself extended pointwise to pairs of functions.

As pointed out before, because of the distinction between passive and active clauses, not all up-to techniques can be applied in all clauses. In fact, we decompose an up-to technique into a pair of functions (s, f), where s can be used in passive clauses while f cannot.

▶ **Definition 7.** *A pair of monotone functions* (s, f) *is an up-to technique if for all* $\mathcal{R}$, $\mathcal{R} \rightarrowtail \mathsf{s}(\mathcal{R}), \mathsf{f}(\mathcal{R})$ *implies* $\mathcal{R} \subseteq \approx$.

In an up-to technique (s, f), s is said *strong* while f is said *weak*. Instead of proving directly that a pair is an up-to technique, we consider a sufficient criterion based on *respectfulness*[2] and the largest respectful pair, called the *diacritical companion* (u, w): if a pair (s, f) is below the companion, then it is an up-to technique.

The diacritical companion is defined using notions of *evolution* on monotone functions which can be seen as the higher-order counterpart of progress on relations. We decompose diacritical progress $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$ into *passive* progress $\mathcal{R} \overset{\mathsf{p}}{\rightarrowtail} \mathcal{S}$ and *active* progress $\mathcal{R} \overset{\mathsf{a}}{\rightarrowtail} \mathcal{T}$ to define different kinds of evolution.

▶ **Definition 8.** *Let* f, g *be monotone functions.*

- f passively evolves *to* g, *written* $\mathsf{f} \overset{\mathsf{p}}{\rightsquigarrow} \mathsf{g}$, *if for all* $\mathcal{R}$, $\mathcal{S}$, $\mathcal{R} \overset{\mathsf{p}}{\rightarrowtail} \mathcal{S}$ *implies* $\mathsf{f}(\mathcal{R}) \overset{\mathsf{p}}{\rightarrowtail} \mathsf{g}(\mathcal{S})$;
- f actively evolves *to* g, *written* $\mathsf{f} \overset{\mathsf{a}}{\rightsquigarrow} \mathsf{g}$, *if for all* $\mathcal{R}$, $\mathcal{S}$, $\mathcal{R} \overset{\mathsf{a}}{\rightarrowtail} \mathcal{S}$ *implies* $\mathsf{f}(\mathcal{R}) \overset{\mathsf{a}}{\rightarrowtail} \mathsf{g}(\mathcal{S})$;
- f restrictively evolves *to* g, *written* $\mathsf{f} \overset{\mathsf{p|a}}{\rightsquigarrow} \mathsf{g}$, *if for all* $\mathcal{R}$, $\mathcal{S}$, $\mathcal{R} \overset{\mathsf{p}}{\rightarrowtail} \mathcal{R} \overset{\mathsf{a}}{\rightarrowtail} \mathcal{S}$ *implies* $\mathsf{f}(\mathcal{R}) \overset{\mathsf{a}}{\rightarrowtail} \mathsf{g}(\mathcal{S})$.

Passive and active evolutions express the idea that f becomes g in respectively passive and active clauses. Restricted evolution allows a relation $\mathcal{R}$ to do some administrative step (passive progress) before doing some active progress, as long as we stay in $\mathcal{R}$. For λ_{eff}, it means that we can reduce a term to a value before doing some active progress with it.

[2] Our previous work [6] is built on the notion of *compatibility*, but the notion of progress we use in this paper makes Definition 9 correspond to respectfulness instead. See [26, 23, 6] for a discussion on the difference between the two notions.

▶ **Definition 9.** *A pair of monotone functions* (s,f) diacritically evolves *to* $(\mathsf{s}',\mathsf{f}'),(\mathsf{s}'',\mathsf{f}'')$, *written* $(\mathsf{s},\mathsf{f}) \rightsquigarrow (\mathsf{s}',\mathsf{f}'),(\mathsf{s}'',\mathsf{f}'')$ *if*

$$\mathsf{s} \overset{\mathsf{p}}{\rightsquigarrow} \mathsf{s}' \qquad \mathsf{f} \overset{\mathsf{p}}{\rightsquigarrow} \mathsf{f}' \qquad \mathsf{s} \overset{\mathsf{a}}{\rightsquigarrow} \mathsf{f}'' \qquad \mathsf{f} \overset{\mathsf{p}|\mathsf{a}}{\rightsquigarrow} \mathsf{f}''$$

A pair (s,f) *is* respectful *if* $(\mathsf{s},\mathsf{f}) \rightsquigarrow (\mathsf{s},\mathsf{f}),(\mathsf{s},\mathsf{f})$. *The* diacritical companion (u,w) *is the largest respectful pair.*

In words, the bisimulations of diacritical evolution are exactly respectful pairs, and its bisimilarity is the diacritical companion. Among other properties, we can show that any pair below the companion (including the companion itself) is an up-to technique.

▶ **Lemma 10.** *The following hold:*

- *if* $(\mathsf{s},\mathsf{f}) \sqsubseteq (\mathsf{u},\mathsf{w})$, *then* (s,f) *is an up-to technique;*
- $\mathsf{u} \sqsubseteq \mathsf{w}$*;*
- $\mathsf{w}(\approx) = \approx$.

The second inequality implies that any strong function can also be used as a weak one, justifying why such a function is said "strong", as it can be applied without restriction in any clause. The last equality states that the weak companion preserves bisimilarity, so for any $\mathsf{f} \sqsubseteq \mathsf{w}$, we also have $\mathsf{f}(\approx) \subseteq \approx$. If f is a contextual closure function (if $e_1 \mathcal{R} e_2$ then $C[e_1]\ \mathsf{f}(\mathcal{R})\ C[e_2]$), showing that it is below w is enough to deduce that $\approx$ is compatible.

The remaining question is how to prove that a given pair (s,f) is below the companion. In this paper, we use a degenerate but sufficient version of a theorem in our previous work [6, Theorem 4.12]. Let id be the identity on relations. We define $\mathsf{S}(\mathsf{s})$ inductively as the smallest function verifying:

- for all $\mathsf{g} \in \{\mathsf{id},\mathsf{s},\mathsf{u}\}$, $\mathsf{g} \sqsubseteq \mathsf{S}(\mathsf{s})$;
- for all $\mathsf{g} \in \{\mathsf{id},\mathsf{s},\mathsf{u}\}$, $\mathsf{g} \circ \mathsf{S}(\mathsf{s}) \sqsubseteq \mathsf{S}(\mathsf{s})$, $\mathsf{S}(\mathsf{s}) \circ \mathsf{g} \sqsubseteq \mathsf{S}(\mathsf{s})$, $\mathsf{g} \cup \mathsf{S}(\mathsf{s}) \sqsubseteq \mathsf{S}(\mathsf{s})$, and $\mathsf{S}(\mathsf{s}) \cup \mathsf{g} \sqsubseteq \mathsf{S}(\mathsf{s})$;

and $\mathsf{W}(\mathsf{s},\mathsf{f})$ inductively as the smallest function verifying:

- for all $\mathsf{g} \in \{\mathsf{id},\mathsf{s},\mathsf{f},\mathsf{w}\}$, $\mathsf{g} \sqsubseteq \mathsf{W}(s)$;
- for all $\mathsf{g} \in \{\mathsf{id},\mathsf{s},\mathsf{f},\mathsf{w}\}$, $\mathsf{g} \circ \mathsf{W}(\mathsf{s},\mathsf{f}) \sqsubseteq \mathsf{W}(\mathsf{s},\mathsf{f})$, $\mathsf{W}(\mathsf{s},\mathsf{f}) \circ \mathsf{g} \sqsubseteq \mathsf{W}(\mathsf{s},\mathsf{f})$, $\mathsf{g} \cup \mathsf{W}(\mathsf{s},\mathsf{f}) \sqsubseteq \mathsf{W}(\mathsf{s},\mathsf{f})$, and $\mathsf{W}(\mathsf{s},\mathsf{f}) \cup \mathsf{g} \sqsubseteq \mathsf{W}(\mathsf{s},\mathsf{f})$;

The function $\mathsf{S}(\mathsf{s})$ is the smallest function built from s, id, and u stable by composition and union, while $\mathsf{W}(\mathsf{s},\mathsf{f})$ is the smallest function built from s, f, id, and w stable by composition and union. Including u and w in their definition means that any function already proved respectively strong or weak is below respectively $\mathsf{S}(\mathsf{s})$ or $\mathsf{W}(\mathsf{s},\mathsf{f})$.

▶ **Theorem 11.** *Let* (s,f) *be monotone functions. If*

$$\mathsf{s} \overset{\mathsf{p}}{\rightsquigarrow} \mathsf{S}(\mathsf{s}) \qquad \mathsf{f} \overset{\mathsf{p}}{\rightsquigarrow} \mathsf{S}(\mathsf{s}) \circ \mathsf{f} \circ \mathsf{S}(\mathsf{s}) \qquad \mathsf{s} \overset{\mathsf{a}}{\rightsquigarrow} \mathsf{W}(\mathsf{s},\mathsf{f}) \qquad \mathsf{f} \overset{\mathsf{p}|\mathsf{a}}{\rightsquigarrow} \mathsf{W}(\mathsf{s},\mathsf{f})$$

then $(\mathsf{s},\mathsf{f}) \sqsubseteq (\mathsf{u},\mathsf{w})$ *and* (s,f) *is an up-to technique.*

The idea of the theorem is to see how s and f evolve and prove that the results of their evolutions is below what is on the right of the arrows. Any combination of weak functions can be obtained after an active or restricted evolution, but only strong functions can be used after a passive one, except that f can be used once. This constraint on f makes the soundness proofs of the most interesting up-to techniques of λ_{eff} more difficult (cf. Appendix A).

We define the up-to functions we consider for λ_{eff} in Figure 2. The first four are usual and can be found in many variants of the λ-calculus [7, 4]. The function red is the usual bisimulation up to reduction, where expressions can be related after some reduction steps, while refl equates any expression with itself. The function subst allows to replace a variable in related expressions with related values. Finally, lam is compatibility w.r.t. λ-abstraction.

Resumption position predicate.

$$\frac{}{\mathsf{resum}(\alpha^{\bar{l}}, x)} \qquad \frac{\mathsf{resum}(\alpha^{\bar{l}}, e)}{\mathsf{resum}(\alpha^{\bar{l}}, \lambda x.e)} \qquad \frac{\mathsf{resum}(\alpha^{\bar{l}}, e_1) \qquad \mathsf{resum}(\alpha^{\bar{l}}, e_2)}{\mathsf{resum}(\alpha^{\bar{l}}, e_1\ e_2)} \qquad \frac{\mathsf{resum}(\alpha^{\bar{l}}, e)}{\mathsf{resum}(\alpha^{\bar{l}}, \mathsf{do}_{l'}\ e)}$$

$$\frac{\mathsf{resum}(\alpha^{\bar{l}}, e) \qquad \forall l_i\colon x_i, k_i \to e_i \in H, \mathsf{resum}(\alpha^{\bar{l}}, e_i) \qquad \mathsf{resum}(\alpha^{\bar{l}}, e')}{\mathsf{resum}(\alpha^{\bar{l}}, \mathsf{handle}\ e\ \{H; \mathsf{ret}\ x \to e'\})} \qquad \frac{\mathsf{resum}(\alpha^{\bar{l}}, v)}{\mathsf{resum}(\alpha^{\bar{l}}, \alpha^{\bar{l}}[v])}$$

$$\frac{\mathsf{resum}(\alpha^{\bar{l}}, v)}{\mathsf{resum}(\alpha^{\bar{l}}, \alpha^{\bar{l}}[\mathsf{do}_l\ v])} \qquad \frac{\mathsf{resum}(\alpha^{\bar{l}}, e) \qquad \beta^{\overline{l'}} \neq \alpha^{\bar{l}}}{\mathsf{resum}(\alpha^{\bar{l}}, \beta^{\overline{l'}}[e])}$$

Up-to techniques.

$$\frac{e_1 \to^* e_1' \qquad e_2 \to^* e_2' \qquad e_1'\ \mathcal{R}\ e_2'}{e_1\ \mathsf{red}(\mathcal{R})\ e_2} \qquad \frac{}{e\ \mathsf{refl}(\mathcal{R})\ e}$$

$$\frac{e_1\ \mathcal{R}\ e_2 \qquad v_1\ \mathcal{R}^{\mathsf{v}}\ v_2}{e_1\{v_1/x\}\ \mathsf{subst}(\mathcal{R})\ e_2\{v_2/x\}} \qquad \frac{e_1\ \mathcal{R}\ e_2}{\lambda x.e_1\ \mathsf{lam}(\mathcal{R})\ \lambda x.e_2}$$

$$\frac{e_1\ \mathcal{R}\ e_2}{\alpha^{\bar{l}}[e_1]\ \mathsf{cvar}(\mathcal{R})\ \alpha^{\bar{l}}[e_2]} \qquad \frac{e_1\ \mathcal{R}\ e_2 \qquad {E_1}^{\bar{l}}\ \mathcal{R}^{\mathsf{c}}\ {E_2}^{\bar{l}}}{e_1\{{E_1}^{\bar{l}}/\alpha^{\bar{l}}\}\ \mathsf{csubst}(\mathcal{R})\ e_2\{{E_2}^{\bar{l}}/\alpha^{\bar{l}}\}}$$

$$\frac{e_1\ \mathcal{R}\ e_2 \qquad {E_1}^{\bar{l}}\ \mathcal{R}^{\mathsf{r}}\ {E_2}^{\bar{l}} \qquad \mathsf{resum}(\alpha^{\bar{l}}, e_1) \qquad \mathsf{resum}(\alpha^{\bar{l}}, e_2)}{e_1\{{E_1}^{\bar{l}}/\alpha^{\bar{l}}\}\ \mathsf{rsubst}(\mathcal{R})\ e_2\{{E_2}^{\bar{l}}/\alpha^{\bar{l}}\}}$$

Figure 2 Up-to functions for λ_{eff}.

The remaining functions are more specific to λ_{eff}. The function cvar plugs related terms into any context variable. This variable can then be replaced with contexts using either csubst or rsubst, depending whether the contexts behave as resumptions or not. In the latter case, the contexts should be related with $\cdot^{\mathsf{r}}$, and the context variable should be in *resumption position*, a condition we check with the predicate resum, defined in Figure 2. Roughly, $\mathsf{resum}(\alpha^{\bar{l}}, e)$ means that $\alpha^{\bar{l}}$ is about to be captured – i.e., plugged with an effect $\mathsf{do}_l\ v$ – or has already been captured, and is therefore plugged with a value.

The functions cvar, csubst, and rsubst can be used to define a more conventional bisimulation up to evaluation context, similar to the one of the plain λ-calculus [7].

▶ **Lemma 12.** *If* $e_1\ \mathcal{R}\ e_2$ *and* $E_1\ \mathcal{R}^{\mathsf{c}}\ E_2$*, then* $E_1[e_1]\ \mathsf{csubst}(\mathsf{cvar}(\mathcal{R}) \cup \mathsf{id})\ E_2[e_2]$.

We simply plug e_1 and e_2 into a fresh context variable which is then replaced with E_1 and E_2.

The functions we define are strong, except for csubst and rsubst.

▶ **Theorem 13.** *For all* $\mathsf{s} \in \{\mathsf{refl}, \mathsf{id}, \mathsf{red}, \mathsf{subst}, \mathsf{lam}, \mathsf{cvar}\}$*, we have* $\mathsf{s} \sqsubseteq \mathsf{u}$*. For all* $\mathsf{f} \in \{\mathsf{csubst}, \mathsf{rsubst}\}$*, we have* $\mathsf{f} \sqsubseteq \mathsf{w}$.

The proofs for the strong techniques are simple or as in the plain λ-calculus [7]; we sketch the proof for csubst and rsubst in the appendix. It is not surprising that these two functions are weak, as they essentially behave as bisimulation up to context, which is also weak in the plain λ-calculus. As explained in Section 3.3, they cannot be used in the passive clauses, i.e., when relating values or context-stuck terms.

Because cvar and csubst are up-to techniques, the bisimulation up to evaluation context is also sound, from which we deduce that $\approx$ is compatible w.r.t. evaluation contexts using Lemma 10. Thanks to lam, we know it is also preserved by λ-abstraction, so we can show the bisimilarity is compatible, from which we deduce it is a valid proof technique for the contextual equivalence of the plain calculus.

▶ **Corollary 14.** *Let e_1 and e_2 be expressions of the plain calculus. If $e_1 \approx e_2$, then $e_1 \equiv e_2$.*

Indeed, if $e_1 \approx e_2$, then for all contexts C, $C[e_1] \approx C[e_2]$ because $\approx$ is compatible. If $C[e_1] \Downarrow_{\mathrm{v}}$, then $C[e_2] \Downarrow_{\mathrm{v}}$ simply by definition of the bisimilarity.

The up-to techniques we define are useful beyond simply proving soundness of the bisimilarity; they can simplify the equivalence proof of two given terms, as illustrated by the following examples.

▶ **Example 15.** Dal Lago and Gavazzo [14] propose an example where two fixed-point combinators are signaling each β-reduction with a tick effect; we modify it so that the two expressions are equivalent with handlers (but the tick effect is now arbitrary). Let

$$e_1 \triangleq \lambda y.\mathsf{do}_{\mathsf{tick}}\,(\Delta_y\,\Delta_y) \qquad \Delta_y \triangleq \lambda x.(\mathsf{do}_{\mathsf{tick}}\,y)\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(x\,x\,z)$$
$$e_2 \triangleq \Theta\,\Theta \qquad \Theta \triangleq \lambda x.\lambda y.\mathsf{do}_{\mathsf{tick}}\,((\mathsf{do}_{\mathsf{tick}}\,y)\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(x\,x\,y\,z))$$

We prove these expressions are bisimilar up to, by building a candidate relation $\mathcal{R}$ incrementally, starting from e_1 and e_2.

Proof. The term e_1 is a value, and $e_2 \to \lambda y.\mathsf{do}_{\mathsf{tick}}\,((\mathsf{do}_{\mathsf{tick}}\,y)\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Theta\,\Theta\,y\,z))$, so we need to relate the bodies of the λ-abstractions. We have a reduction $\mathsf{do}_{\mathsf{tick}}\,(\Delta_y\,\Delta_y) \to \mathsf{do}_{\mathsf{tick}}\,((\mathsf{do}_{\mathsf{tick}}\,y)\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Delta_y\,\Delta_y\,z))$; the resulting term is control-stuck, which we relate to $\mathsf{do}_{\mathsf{tick}}\,((\mathsf{do}_{\mathsf{tick}}\,y)\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Theta\,\Theta\,y\,z))$ which is also control-stuck. The arguments of the effect are the same, and we need to relate the two contexts $\mathsf{do}_{\mathsf{tick}}\,(\Box\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Delta_y\,\Delta_y\,z))$ and $\mathsf{do}_{\mathsf{tick}}\,(\Box\,\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Theta\,\Theta\,y\,z))$.

Plugging them with a fresh variable, we obtain two open-stuck terms, meaning that we need to relate the two identical contexts $\mathsf{do}_{\mathsf{tick}}\,\Box$ and the values $\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Delta_y\,\Delta_y\,z)$ and $\lambda z.\mathsf{do}_{\mathsf{tick}}\,(\Theta\,\Theta\,y\,z)$. These last two values are related up to lambda and evaluation context if $\mathcal{R}$ contains $\Delta_y\,\Delta_y$ and $\Theta\,\Theta\,y$, and the bisimulation proof for these two expressions is the same as for e_1 and e_2. In the end, taking $\mathcal{R} \triangleq \{(e_1, e_2), (\Delta_y\,\Delta_y, \Theta\,\Theta\,y)\}$, we can show that $\mathcal{R}$ is a bisimulation up to refl, red, lam, and up to context, i.e., up to cvar and csubst. Note that we are allowed to use the latter weak technique when comparing open-stuck terms, as it is an active clause. ◀

▶ **Example 16.** We write E_R for the reader effect of Example 6, and consider the following handler to express backtracking.

$$E_{BT} \triangleq \mathsf{handle}\ \Box\ \{\mathsf{fail}\colon x,k \to ();\mathsf{flip}\colon x,k \to (\lambda z.k\ \mathsf{false})\ (k\ \mathsf{true});\mathsf{ret}\ x \to x\}$$
$$E_R \triangleq \mathsf{handle}\ \Box\ \{\mathsf{ask}\colon x,k \to k\ z;\mathsf{ret}\ x \to x\}$$

We prove that the two effects commute by showing that $E_{BT}[E_R] \approx^{\mathsf{c}} E_R[E_{BT}]$.

Sketch. We show that the relation $\mathcal{R}$ given by the following rules is a bisimulation up-to.

$$\frac{}{E_{BT}[E_R[v]]\ \mathcal{R}\ E_R[E_{BT}[v]]} \qquad \frac{}{E_{BT}[E_R[\alpha^{\bar{l}}[\mathsf{do}_l\, x]]]\ \mathcal{R}\ E_R[E_{BT}[\alpha^{\bar{l}}[\mathsf{do}_l\, x]]]}$$

$$\frac{e_1\ \mathsf{red}(\mathcal{R})\ E_R[e_2] \qquad z \notin \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2)}{(\lambda z.e_1)\ (E_{BT}[E_R[\alpha^{\bar{l}}[\mathsf{do}_l\, x]]])\ \mathcal{R}\ E_R[(\lambda z.e_2)\ (E_{BT}[\alpha^{\bar{l}}[\mathsf{do}_l\, x]]])}$$

The pair of the first rule is straightforward to check as each expression evaluates to v. For the second rule, the interesting cases are when l is an effect handled by E_{BT} or E_R. If $l = \mathsf{fail}$, the two expressions evaluate to $()$. If $l = \mathsf{ask}$, they evaluate to respectively $E_{BT}[E_R[\alpha^{\bar{l}}[z]]]$ and $E_R[E_{BT}[\alpha^{\bar{l}}[z]]]$, which are context-stuck terms and for which we can easily check the bisimulation requirements.

If $l = \mathsf{flip}$, then the expressions of the second rule reduce to respectively

$(\lambda z.((\lambda y.E_{BT}[E_R[\alpha^{\bar{l}}[y]]])\ \mathsf{false}))\ E_{BT}[E_R[\alpha^{\bar{l}}[\mathsf{true}]]]$, and

$E_R[(\lambda z.((\lambda y.E_{BT}[\alpha^{\bar{l}}[y]])\ \mathsf{false}))\ E_{BT}[\alpha^{\bar{l}}[\mathsf{true}]]]$.

To compare these context-stuck terms, we plug the two contexts with a fresh variable and a fresh control-stuck terms. When plugged with a fresh variable, we obtain $E_{BT}[E_R[\alpha^{\bar{l}}[\mathsf{false}]]]$ and $E_R[E_{BT}[\alpha^{\bar{l}}[\mathsf{false}]]]$, for which we can again easily check the bisimulation clause. With control-stuck terms, we obtain expressions related by the third rule defining $\mathcal{R}$. Checking bisimulation for the third rule is done by a similar case analysis on l and concludes the proof. ◀

3.5 Completeness

In this section we show that for any two expressions e_1 and e_2 in the plain calculus, if $e_1 \equiv e_2$, then $e_1 \approx e_2$. To this end, we first observe that if $e_1 \equiv e_2$, then $e_1 \equiv_{\mathrm{E}} e_2$, where $\equiv_{\mathrm{E}}$ is a relation on expressions in the extended calculus, defined as follows.

▶ **Definition 17.** *We write $e_1 \equiv_{\mathrm{E}} e_2$ if for all evaluation contexts E (from the extended calculus), and substitutions σ (i.e., finite mappings from variables to values and from context variables to contexts), such that $E[e_1]\sigma$ and $E[e_2]\sigma$ are closed expressions in the plain calculus, we have $E[e_1]\sigma \Downarrow_{\mathrm{v}}$ iff $E[e_2]\sigma \Downarrow_{\mathrm{v}}$.*

▶ **Lemma 18.** *If $e_1 \equiv e_2$, then $e_1 \equiv_{\mathrm{E}} e_2$.*

Proof. Assume that $e_1 \equiv e_2$ and take any evaluation context E and closing substitution σ, such that $E[e_1]\sigma \Downarrow_{\mathrm{v}}$. Then, it must be the case that $E[e_2]\sigma \Downarrow_{\mathrm{v}}$ as well, since otherwise e_1 and e_2 would be distinguished by the following context:

$$C = (\lambda x_1 \ldots \lambda x_n.E\sigma)\ v_1 \ldots v_n$$

assuming $\mathsf{dom}(\sigma) = \{x_1, \ldots x_n, \alpha_1, \ldots, \alpha_m\}$ and $\sigma(x_i) = v_i$ for $1 \le i \le n$. ◀

The main lemma of this section establishes that $\equiv_{\mathrm{E}}$ is a bisimulation, which, by Lemma 18, implies completeness of $\approx$ w.r.t. $\equiv$.

▶ **Lemma 19.** *$\equiv_{\mathrm{E}}$ is a bisimulation.*

Proof. The proof consists in a case-by-case verification of the conditions stated in Definition 5 for the candidate relation $\equiv_{\mathrm{E}}$. Here we present one of the most representative cases that, in our opinion, illustrates best the power of the calculus and the techniques used in the remaining cases.

Case: $e_1 = E_1[\bar{\alpha^l}[v_1]]$ and $e_1 \equiv_{\mathrm{E}} e_2$. We need to show that there exist E_2 and v_2 such that: (1) $e_2 \to^* E_2[\bar{\alpha^l}[v_2]]$, (2) $v_1 \equiv_{\mathrm{E}}^{\mathsf{v}} v_2$, and (3) $E_1 \equiv_{\mathrm{E}}^{\mathsf{c}} E_2$.

To prove (1), we take a fresh label l', and we define a substitution σ as follows:

$$\begin{array}{rcll} \sigma(x) & = & \lambda y.\Omega & \text{for } x \in \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\ \sigma(\beta^{\overline{l''}}) & = & \mathsf{handle}\ \square\ \{H_{l''}; \mathsf{ret}\ x \to \Omega\} & \text{for } \beta^{\overline{l''}} \in \mathsf{cv}(e_1) \cup \mathsf{cv}(e_2) \text{ and } \beta^{\overline{l''}} \neq \alpha^{\bar{l}} \\ \sigma(\alpha^{\bar{l}}) & = & \mathsf{do}_{l'}\ \square & \end{array}$$

where $H_{l''} = l_1 \colon x, k \to \Omega; \ldots; l_n \colon x, k \to \Omega$ and $\{l_1, \ldots, l_n\} = \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2) - \{l''\}$, and we consider a context $E = \mathsf{handle}\ \square\ \{l' \colon x, k \to x; \mathsf{ret}\ x \to \Omega\}$. It is easy to see that $E[e_1]\sigma \Downarrow_{\mathrm{v}}$ and that if e_2 evaluates to a normal form which is not $E_2[\bar{\alpha^l}[v_2]]$ for some E_2 and v_2, then either $E[e_2]\sigma \Uparrow$ or $E[e_2]\sigma$ reduces to a control-stuck term (the latter case occurs when e_2 itself reduces to a control-stuck term $E_2[\mathsf{do}_{l''}\ v_2]$).

To prove (2), we take a fresh variable z, a context E and a closing substitution σ, and we assume that $E[v_1\ z]\sigma \Downarrow_{\mathrm{v}}$. To see that $E[v_2\ z]\sigma \Downarrow_{\mathrm{v}}$ as well, we construct a substitution σ' and a context E' such that $E'[e_i]\sigma' \Downarrow_{\mathrm{v}}$ iff $E[v_i\ z]\sigma \Downarrow_{\mathrm{v}}$ for $i = 1, 2$. To this end we take fresh labels l', get and put (the latter two to encode a binary state as an algebraic effect), and we define σ' to be equal to σ everywhere, except for $\alpha^{\bar{l}}$:[3]

$$\sigma'(\alpha^{\bar{l}}) = \sigma(\alpha^{\bar{l}})[(\lambda x.\mathsf{if}\ \mathsf{do}_{\mathsf{get}}\ ()\ \mathsf{then}\ (\mathsf{do}_{\mathsf{put}}\ \mathsf{false};\ \mathsf{do}_{l'}\ x)\ \mathsf{else}\ x)\ \square]$$

along with

$$E'_b = (\mathsf{handle}\ E''\ \{\mathsf{get} \colon x, k \to \lambda y.k\ y\ y;\ \mathsf{put} \colon x, k \to \lambda y.k\ ()\ x;\ \mathsf{ret}\ x \to \lambda y.x\})\ b$$
$$E'' = \mathsf{handle}\ \square\ \{l' \colon x, k \to E[x\ z];\ \mathsf{ret}\ x \to x\}.$$

where $b \in \{\mathsf{true}, \mathsf{false}\}$. Let us notice that

$$E'_{\mathsf{true}}[e_i]\sigma' \to^* E'_{\mathsf{false}}[E_i[\mathsf{do}_{l'}\ v_i]]\sigma' \to^* E'_{\mathsf{false}}[E[v_i\ z]]\sigma'$$

The idea is to use $\bar{\alpha^l}$, the single synchronization point of e_1 and e_2 available, in such a way that the first time $\bar{\alpha^l}$ is used, $E'_{\mathsf{true}}[e_i]\sigma'$ reduces to an expression behaving like $E[v_i\ z]\sigma$. To ensure this, we make sure that any subsequent uses of $\bar{\alpha^l}$ (it could occur in v_i or E) actually mean $\sigma(\bar{\alpha^l})$. But when the state is set to false, the λ-abstraction in $\sigma'(\bar{\alpha^l})$ behaves like the identity, and filling the hole of $\sigma'(\bar{\alpha^l})$ with a value v simply passes v to $\sigma(\bar{\alpha^l})$. Filling it with a control-stuck term $E'^{\overline{l'}}[\mathsf{do}_{l'}\ v]$ allows $\sigma(\bar{\alpha^l})$ to eventually handle the effect, capturing a context equivalent to $(\lambda z.z)\ E'^{\overline{l'}}$. In the end, $E'_{\mathsf{false}}[E[v_i\ z]]\sigma'$ behaves like $E[v_i\ z]\sigma$, up to a few additional reduction steps.

To prove (3), we have to show: (a) $E_1[z] \equiv_{\mathrm{E}} E_2[z]$ for a fresh variable z, and (b) $E_1[\overline{\alpha^{l''}}[\mathsf{do}_{l''}\ z]] \equiv_{\mathrm{E}} E_2[\overline{\alpha^{l''}}[\mathsf{do}_{l''}\ z]]$ for any l'' and fresh $\overline{\alpha^{l''}}$ and z. Assuming we compare expressions using E and σ in both cases, we proceed as in (2), except that in (a) we take

$$E'' = \mathsf{handle}\ \square\ \{l' \colon x, k \to E[k\ z];\ \mathsf{ret}\ x \to x\}$$

and in (b) we take

$$E'' = \mathsf{handle}\ \square\ \{l' \colon x, k \to E[k\ (\overline{\alpha^{l''}}[\mathsf{do}_{l''}\ z])];\ \mathsf{ret}\ x \to x\}.$$

The remaining cases are proved similarly and can be found in Appendix B. ◀

▶ **Corollary 20.** *For any expressions e_1 and e_2 in the plain calculus, if $e_1 \equiv e_2$, then $e_1 \approx e_2$.*

[3] Strictly speaking, σ' additionally takes into account the free variables and context variables that occur in e_1 or e_2, but that have been reduced away and are not present in the resulting normal forms. The values and contexts σ' assigns to such variables are irrelevant.

3.6 Comparison with Multi-Prompted Delimited Continuations

Algebraic effects and handlers studied in the untyped setting, as in this work, diverge from their categorical origins [22], and can be considered a new form of delimited control [10, 11]. As a matter of fact, there exist mutual encodings of algebraic effects and (deep) handlers over a single operation and the control operator shift0 [28], both in an untyped [12] and polymorphically typed settings [21]. These encodings are not fully abstract and therefore they do not guarantee that a behavioral theory, such as the one presented in this work, would carry over to the corresponding calculus of delimited continuations. Given that we allow for multi-labeled algebraic operations, the corresponding calculus in our case would be a generalization of shift0 to its multi-prompted version $\mathsf{shift0}_l$ where the main reduction rule is:

$$\mathsf{prompt}_l E^{\bar{l}}[\mathsf{shift0}_l k.e] \quad \mapsto \quad e\{\lambda z.\mathsf{prompt}_l E^{\bar{l}}[z]/k\}$$

We can observe that in contrast to the calculus of algebraic effects, the party responsible for handling the effect is the same as the one that actually does the effect – it is not the prompt that handles it, but the expression e. The reversal of the roles makes algebraic effects considerably more programmer-friendly, but it also simplifies the theory, compared to the one for classical delimited-control operators. In particular, the techniques we propose in this work appear not to be sufficient for constructing a normal-form bisimulation theory for multi-prompted shift0.

The main obstacle is encountered when we relate evaluation contexts, say E_1 and E_2. The requirement that $E_1[z]$ and $E_2[z]$ (for a fresh z) be related is uncontroversial. However, how should we test E_1 and E_2 for control effects? We need a notion of an abstract control-stuck term and we do not know how to represent it in this calculus. We could introduce a syntactic category of control-stuck-term variables for this purpose, but this would lead nowhere – plugging E_1 and E_2 with such a variable would immediately result in control-stuck terms – there simply is no code that could test the contexts.

One could try to decompose the contexts E_1 and E_2 into some corresponding sub-contexts and relate those, following the approach that works for single-prompted control operators shift and reset for which there exists a sound normal-form bisimilarity [4]. Whether this could lead to a complete theory is not clear and requires further study. As for single-prompted control operators, be it shift or shift0, reaching completeness seems a tall order – notice that the completeness proof of Section 3.5 hinges on the existence of fresh effect labels (prompts).

4 Related Work

Up to now, most works studying the behavioral theory of a calculus with generic algebraic effects were not considering handlers, but interpretations of effects instead, usually in a monad. In such a setting, the behavior of an effect is therefore given for all programs once and for all by the interpretation. In contrast, with handlers, the behavior of an effect may change between programs or during the execution of a program as it depends on how it is handled. The calculus we consider is therefore more expressive than those of the works we list below, with a more discriminative contextual equivalence. It explains why we can reach completeness with a syntactic equivalence such as normal-form bisimilarity while previous works do not achieve completeness with more elaborate equivalences such as applicative bisimilarity. As a matter of fact, the completeness proof presented in this paper relies on an encoding of state and resembles the completeness proof we developed for higher-order state in a previous work [5]. The definition of the normal-form bisimilarity for state, unlike the one presented in this work, did not require any extensions of the calculus. However, its

structure is considerably more involved since in the absence of control operators, to reach completeness, we had to explicitly handle deferred diverging terms and impose a stack-like discipline on the way evaluation contexts are tested.

Some recent works interpret effects in a monad and use *relators* which express how interpreted terms should be compared in the monad. Relators allow to develop the behavioral theory of a calculus with effects in a very abstract setting: e.g., one can get for free that the bisimilarity is a congruence provided that a relator exists for the interpretation monad. Relators have been studied for applicative bisimilarity in call-by-value [15] or call-by-name [16], and for normal-form bisimilarity in call-by-value [14]. As pointed out by the authors in [16], *"there is however little hope to prove a generic full-abstraction result [w.r.t. contextual equivalence] in such a setting, although for certain notions of an effect, full abstraction is already known to hold."* However, completeness can be obtained in some cases, as in an untyped call-by-name calculus with deterministic effects [16].

The other path to completeness in typed languages is through logic or logical relations. Johann et al. [13] propose a contextual equivalence and a logical relation characterizing it in a call-by-name calculus with effects. Their framework deals with different effects in a uniform way but with some limitations, as for instance nondeterminism, local store, or the combination of effects cannot be accounted for. Simpson and Voorneveld [29] present a modal logic for a call-by-value calculus which coincides with Dal Lago et al.'s applicative bisimilarity [15], but not with contextual equivalence, as demonstrated later [19]. Matache and Staton improve on these results by defining a logic for a calculus in continuation-passing style that coincides with both applicative bisimilarity and contextual equivalence [19]. Finally, Biernacki et al. [8] define a step-indexed logical relation for a call-by-value calculus with effects and handlers; to the best of our knowledge, it is the only previous work with handlers.

5 Conclusion

We present a sound and complete normal-form bisimilarity for a calculus with effects and handlers. The crucial point is to accurately observe how evaluation contexts may handle effects. First, we distinguish between resumptions, which are plugged only with values, from regular contexts, which may be plugged with any expressions, including effectful ones. We then test the latter contexts using control-stuck terms where the continuation is represented by a context variable, which allows to track how the captured continuation is handled. Extending the calculus with context variables introduces new normal forms which are compared by the bisimilarity in a very simple and regular way. The fact that such a simple notion of normal-form bisimilarity is complete shows the discriminating power of handlers. A consequence is that the examples of equivalent programs we provide are quite simple, as more complex effectful expressions are easily distinguished by handlers.

There are several directions for future work. As pointed out in Section 3.6, it remains an open question how to define complete normal-form bisimulations in the calculus of multi-prompted delimited-control operators corresponding to deep handlers studied in this work. Then, it would be worthwhile to investigate whether the results presented in this paper carry over to shallow handlers. Finally, there exist a number of type-and-effect systems for algebraic effects of varying complexity [8, 9, 21], and one can wonder how features such as effect polymorphism along with effect coercions would influence the theory of this paper.

References

1 Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, The University of Texas Year of Programming Series, chapter 4, pages 65–116. Addison-Wesley, 1990.

2 Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. *Logical Methods in Computer Science*, 13(3), 2017.

3 Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015. doi:10.1016/j.jlamp.2014.02.001.

4 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Bisimulations for delimited-control operators. *Logical Methods in Computer Science*, 15(2), 2019.

5 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2019. doi:10.1007/978-3-030-17127-8_6.

6 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Diacritical companions. In Barbara König, editor, *Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics*, volume 347 of *Electronic Notes in Theoretical Computer Science*, pages 25–43, London, England, 2019.

7 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Proving soundness of extensional normal-form bisimilarities. *Logical Methods in Computer Science*, 15(1), 2019.

8 Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Handle with care: relational interpretation of algebraic effects and handlers. *PACMPL*, 2(POPL):8:1–8:30, 2018.

9 Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Abstracting algebraic effects. *PACMPL*, 3(POPL):6:1–6:28, 2019. doi:10.1145/3290319.

10 Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

11 Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 180–190, San Diego, California, January 1988. ACM Press.

12 Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: effect handlers, monadic reflection, delimited control. *PACMPL*, 1(ICFP):13:1–13:29, 2017.

13 Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. In Jean-Pierre Jouannaud, editor, *Proceedings of the 25th IEEE Symposium on Logic in Computer Science (LICS 2010)*, pages 209–218, Edinburgh, UK, July 2010. IEEE Computer Society Press.

14 Ugo Dal Lago and Francesco Gavazzo. Effectful normal form bisimulation. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 263–292, Prague, Czech Republic, April 2019. Springer.

15 Ugo Dal Lago, Francesco Gavazzo, and Paul Blain Levy. Effectful applicative bisimilarity: Monads, relators, and Howe's method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE Computer Society.

16 Ugo Dal Lago, Francesco Gavazzo, and Ryo Tanaka. Effectful applicative similarity for call-by-name lambda calculi. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro, editors, *Joint Proceedings of the 18th Italian Conference on Theoretical Computer*

Science and the 32nd Italian Conference on Computational Logic co-located with the 2017 IEEE International Workshop on Measurements and Networking (2017 IEEE M&N), volume 1949 of *CEUR Workshop Proceedings*, pages 87–98, Naples, Italy, September 2017. CEUR-WS.org.

17 Søren B. Lassen. Eager normal form bisimulation. In Prakash Panangaden, editor, *Proceedings of the Twentieth Annual IEEE Symposium on Logic in Computer Science*, pages 345–354, Chicago, IL, June 2005. IEEE Computer Society Press.

18 Jean-Marie Madiot. *Higher-order languages: dualities and bisimulation enhancements.* PhD thesis, Université de Lyon and Università di Bologna, 2015.

19 Cristina Matache and Sam Staton. A sound and complete logic for algebraic effects. In Mikolaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 382–399, Prague, Czech Republic, April 2019. Springer.

20 James H. Morris. *Lambda Calculus Models of Programming Languages.* PhD thesis, Massachusets Institute of Technology, 1968.

21 Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Typed equivalence of effect handlers and delimited control. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPIcs*, pages 30:1–30:16, Dortmund, Germany, June 2019. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

22 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:23)2013.

23 Damien Pous. Coinduction all the way up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 307–316, New York, NY, USA, July 2016. ACM.

24 John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Paris, France, 1983. IFIP.

25 Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. In Andre Scedrov, editor, *LICS'92*, pages 102–109, Santa Cruz, California, June 1992. IEEE Computer Society.

26 Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, October 1998.

27 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33(1):1–69, January 2011.

28 Chung-chieh Shan. A static simulation of dynamic delimited control. *Higher-Order and Symbolic Computation*, 20(4):371–401, 2007.

29 Alex Simpson and Niels F. W. Voorneveld. Behavioural equivalence via modalities for algebraic effects. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 300–326, Thessaloniki, Greece, April 2018. Springer.

A Soundness Proof Sketch

We only discuss the case of csubst and rsubst, as the others are proved as in the plain λ-calculus [7]. In particular, we use the fact that

▶ **Lemma 21.** $\mathsf{subst} \sqsubseteq \mathsf{u}$

We want to prove that csubst and rsubst are weak, but to circumvent the constraint that they cannot be composed twice in a passive clause, we combine csubst and rsubst in a single ssubst doing simultaneous substitutions.

$$\frac{e_1\ \mathcal{R}\ e_2 \qquad \sigma_1\ \mathcal{R}^\sigma\ \sigma_2}{e_1\sigma_1\ \mathsf{ssubst}(\mathcal{R})\ e_2\sigma_2}$$

We let σ ranges over simultaneous substitution, meaning that if $\sigma(\alpha^{\bar{l}}) = E^{\bar{l}}$, then $(\alpha^{\bar{l}}[e])\sigma \triangleq E^{\bar{l}}[e\sigma]$; we do not apply σ to $E^{\bar{l}}$. We define $\mathcal{R}^\sigma$ pairwise such that we have either $\sigma_1(\alpha^{\bar{l}})\ \mathcal{R}^{\mathsf{c}}\ \sigma_2(\alpha^{\bar{l}})$, or $\sigma_1(\alpha^{\bar{l}})\ \mathcal{R}^{\mathsf{r}}\ \sigma_2(\alpha^{\bar{l}})$ with $\mathsf{resum}(\alpha^{\bar{l}}, e_1)$ and $\mathsf{resum}(\alpha^{\bar{l}}, e_2)$.

▶ **Lemma 22.** ssubst $\sqsubseteq$ w

Proof. Let $\mathcal{R} \rightarrowtail \mathcal{R}, \mathcal{S}$, $e_1\sigma_1\ \mathsf{subst}(\mathcal{R})\ e_2\sigma_2$ with $e_1\ \mathcal{R}\ e_2$ and $\sigma_1\ \mathcal{R}^{\mathsf{c}}\ \sigma_2$. We proceed by case analysis on the behavior of e_1. The cases where e_1 reduces, is a value, or is an open-stuck term are simple.

Suppose $e_1 = {E_1'}^{\bar{l}}[\mathsf{do}_l\, v_1]$, then there exist ${E_2'}^{\bar{l}}$ and v_2 such that $e_2 \to^* {E_2'}^{\bar{l}}[\mathsf{do}_l\, v_2]$, ${E_1'}^{\bar{l}}\ \mathcal{S}^{\mathsf{r}}\ {E_2'}^{\bar{l}}$ and $v_1\ \mathcal{S}^{\mathsf{v}}\ v_2$. Any context variable surrounding the hole of ${E_1'}^{\bar{l}}$ can only be of the form $\alpha_i^{\bar{l}}$, meaning that ${E_1'}^{\bar{l}}\sigma_1$ still does not handle l, and the resulting terms are control-stuck. We progress to ssubst, so we can conclude.

Suppose $e_1 = E_1[\alpha^{\bar{l}}[v_1]]$ with $\alpha^{\bar{l}} \in \mathsf{dom}(\sigma_1)$ (the case where the variable is not in the domain is easily handled). There exist E_2 and v_2 such that $e_2 \to^* E_2[\alpha^{\bar{l}}[v_2]]$, $E_1\ \mathcal{R}^{\mathsf{c}}\ E_2$, and $v_1\ \mathcal{R}^{\mathsf{v}}\ v_2$. From $\sigma_1(\alpha^{\bar{l}})\ \mathcal{R}^{\mathsf{c}}\ \sigma_2(\alpha^{\bar{l}})$, we get in particular $\sigma_1(\alpha^{\bar{l}})[x]\ \mathcal{R}\ \sigma_2(\alpha^{\bar{l}})[x]$ for a fresh x, therefore $\sigma_1(\alpha^{\bar{l}})[v_1]\ \mathsf{subst}(\mathcal{R})\ \sigma_2(\alpha^{\bar{l}})[v_2]$. We have two special cases to consider, $\sigma_1(\alpha^{\bar{l}}) = \beta^{\bar{l}}[\Box]$ and $\sigma_1(\alpha^{\bar{l}}) = \Box$; in the other cases, $\sigma_1(\alpha^{\bar{l}})[v_1]$ is doing something active and we can conclude using Lemma 21.

If $\sigma_1(\alpha^{\bar{l}}) = \Box$, we have $x\ \mathcal{R}\ \sigma_2(\alpha^{\bar{l}})[x]$, from which we deduce that there exist w such that $\sigma_2(\alpha^{\bar{l}})[x] \to^* w$ and $x\ \mathcal{R}^{\mathsf{v}}\ w$. As a result, $e_1\sigma_1 = E_1'\{\!/\sigma\}1[v_1\sigma_1]$, and $e_2\sigma_2 \to^* E_2'\sigma_2[w\{v_2/x\}\sigma_2]$. Since we have $E_1'[v_1]\ \mathsf{subst}(\mathsf{subst}(\mathcal{R}))\ E_2'[w\{v_2/x\}]$, we can conclude again with Lemma 21.

If $\sigma_1(\alpha^{\bar{l}}) = \beta^{\bar{l}}[\Box]$, then from $\beta^{\bar{l}}[x]\ \mathcal{R}\ \sigma_2(\alpha^{\bar{l}})[x]$, there exist E_2' and w such that $\sigma_2(\alpha^{\bar{l}})[x] \to^* E_2'[\beta^{\bar{l}}[w]]$, $\Box\ \mathcal{R}^{\mathsf{c}}\ E_2'$, and $x\ \mathcal{R}^{\mathsf{v}}\ w$. Therefore we have $e_2\sigma_2 \to^* E_2\sigma_2[\sigma_2(\alpha^{\bar{l}})[v_2\sigma_2]] \to^* E_2\sigma_2[E_2'[\beta^{\bar{l}}[w\{v_2\sigma_2/x\}]]]$, yielding a context-stuck term that is to be related to $E_1\sigma_1[\beta^{\bar{l}}[v_1\sigma_1]]$. We are fine w.r.t. the values, as we have $v_1\sigma_1\ \mathsf{ssubst}(\mathsf{subst}(\mathcal{R}))\ w\{v_2\sigma_2/x\}$. For the contexts, we first relate $E_1\sigma_1[y]$ and $E_2\sigma_2[E_2'[y]]$ for a fresh y. Because $\Box\ \mathcal{R}^{\mathsf{c}}\ E_2'$, there exists w' such that $E_2'[y] \to^* w'$ and $y\ \mathcal{R}^{\mathsf{v}}\ w'$. As a result, we have $E_2\sigma_2[E_2'[y]] \to^* E_2\sigma_2[w']$, and therefore $E_1\sigma_1[y]\ \mathsf{red}(\mathsf{ssubst}(\mathsf{subst}(\mathcal{R})))\ E_2\sigma_2[E_2'[y]]$, which is what we need. Then we must relate $E_1\sigma_1[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]$ and $E_2\sigma_2[E_2'[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]]$ for any l' and fresh $\gamma^{\overline{l'}}$ and y. Because $\Box\ \mathcal{R}^{\mathsf{c}}\ E_2'$, there exist ${E_2''}^{\overline{l'}}$ and w' such that $E_2'[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]] \to^* {E_2''}^{\overline{l'}}[\mathsf{do}_{l'}\, w']$, $\gamma^{\overline{l'}}[\Box]\ \mathcal{S}^{\mathsf{r}}\ {E_2''}^{\overline{l'}}$, and $y\ \mathcal{S}^{\mathsf{v}}\ w'$. From $E_1\ \mathcal{R}^{\mathsf{c}}\ E_2$, we get $E_1[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]\ \mathcal{R}\ E_2[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]$, so if $E_1[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]] \to e_1'$ for some e_1' (the case where l' is not handled is not interesting), then there exists e_2' such that $E_2[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]] \to^* e_2'$ and $e_1'\ \mathcal{S}\ e_2'$. Therefore, $E_2\sigma_2[E_2'[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]] \to^* E_2\sigma_2[{E_2''}^{\overline{l'}}[\mathsf{do}_{l'}\, w']] \to^* e_2'\sigma_2'\{w'/y\}$ where $\sigma_2'(\gamma^{\overline{l'}}) = {E_2''}^{\overline{l'}}$ and is equal to σ_2 otherwise. Because $E_1\sigma_1[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]] \to e_1\sigma_1 = e_1\sigma_1'\{y/y\}$ where $\sigma_1'(\gamma^{\overline{l'}}) = \gamma^{\overline{l'}}[\Box]$ and is equal to σ_1 otherwise, we deduce $E_1\sigma_1[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]\ \mathsf{red}(\mathsf{subst}(\mathsf{ssubst}(\mathcal{S})))\ E_2\sigma_2[E_2'[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, y]]]$ which is enough to conclude.

Suppose $e_1 = E_1[\alpha^{\bar{l}}[{E_1'}^{\overline{l'}}[\mathsf{do}_{l'}\, v_1]]]$ with $l \neq l'$ and $\alpha^{\bar{l}} \in \mathsf{dom}(\sigma_1)$; then there exist E_2, ${E_2'}^{\overline{l'}}$, and v_2 such that $e_2 \to^* E_2[\alpha^{\bar{l}}[{E_2'}^{\overline{l'}}[\mathsf{do}_{l'}\, v_2]]]$, $E_1\ \mathcal{T}^{\mathsf{c}}\ E_2$, ${E_1'}^{\overline{l'}}\ \mathcal{T}^{\mathsf{r}}\ {E_2'}^{\overline{l'}}$, and $v_1\ \mathcal{T}^{\mathsf{v}}\ v_2$. From $\sigma_1(\alpha^{\bar{l}})\ \mathcal{R}^{\mathsf{c}}\ \sigma_2(\alpha^{\bar{l}})$, we get $\sigma_1(\alpha^{\bar{l}})[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, x]]\ \mathcal{R}\ \sigma_2(\alpha^{\bar{l}})[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\, x]]$ for fresh $\gamma^{\overline{l'}}$ and x. If

$\sigma_1(\alpha^{\bar{l}})[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\ x]] \to e_1'$ for some e_1', then there exists e_2' such that $\sigma_2(\alpha^{\bar{l}})[\gamma^{\overline{l'}}[\mathsf{do}_{l'}\ x]] \to^* e_2'$ and $e_1'\ \mathcal{S}\ e_2'$. Then $e_1\sigma_1 \to E_1[e_1'\{v_1/x\}\{{E_1'}^{\overline{l'}}/\gamma^{\overline{l'}}\}]\sigma_1$ and $e_2\sigma_2 \to^* E_2[e_2'\{v_2/x\}\{{E_2'}^{\overline{l'}}/\gamma^{\overline{l'}}\}]\sigma_2$, and the resulting expressions are in $\mathsf{ssubst}(\mathsf{ssubst}(\mathsf{cvar}(\mathsf{ssubst}(\mathsf{subst}(\mathcal{S})))))$, which is fine, because we are in an active clause. ◀

B Completeness Proof Sketch

The proof proceeds as described in Section 3.5: given $e_1 \equiv_{\mathrm{E}} e_2$, we check that for each behavior of e_1, e_2 is able to match. If e_1 is a normal form, we verify that (1) e_2 evaluates to a normal form of the same kind, and the normal forms can be decomposed into related sub-parts. For each case, we give the substitution σ and the context E enforcing (1). Checking that related sub-parts are contextually equivalent relies in most cases on an encoding of a mutable state using handlers, as in Section 3.5. In all the subcases below, we assume the labels get and put to be fresh, and given a boolean b and a context E'', we define

$$E_b' = (\mathsf{handle}\ E''\ \{\mathsf{get}\colon z,k \to \lambda y.k\ y\ y;\ \mathsf{put}\colon z,k \to \lambda y.k\ ()\ z;\ \mathsf{ret}\ z \to \lambda y.z\})\ b$$

We define E'' in each subcase where the encoding is needed.

Case: $e_1 \to e_1'$. Because the reduction is deterministic, we still have $e_1' \equiv_{\mathrm{E}} e_2$.

Case: $e_1 = v_1$. To check (1), take σ as follows:

$$\begin{array}{rcll} \sigma(x) & = & \lambda y.\Omega & \text{for } x \in \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\ \sigma(\alpha^{\bar{l}}) & = & \mathsf{handle}\ \square\ \{H_l; \mathsf{ret}\ x \to \Omega\} & \text{for } \alpha^{\bar{l}} \in \mathsf{cv}(e_1) \cup \mathsf{cv}(e_2) \end{array}$$

where $H_l = l_1\colon x,k \to \Omega; \ldots; l_n\colon x,k \to \Omega$ with $\{l_1, \ldots, l_n\} = \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2) \setminus \{l\}$, and $E = \square$. Hence, there exists v_2 such that $e_2 \to^* v_2$; we check that $v_1 \equiv_{\mathrm{E}}^{\mathsf{v}} v_2$.

Let x be a fresh variable, E a context, and σ a closing substitution such that $E[v_1\ x]\sigma \Downarrow_{\mathrm{v}}$. Then $E[e_2\ x]\sigma \to^* E[v_2\ x]\sigma$ and since $e_1 \equiv_{\mathrm{E}} e_2$, we also have $E[v_2\ x]\sigma \Downarrow_{\mathrm{v}}$.

Case: $e_1 = E_1[x\ v_1]$. To check (1), take σ as follows:

$$\begin{array}{rcll} \sigma(z) & = & \lambda y.\Omega & \text{for } z \in \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \setminus \{x\} \\ \sigma(x) & = & \lambda y.\mathsf{do}_{l'}\ \lambda z.z & \\ \sigma(\alpha^{\bar{l}}) & = & \mathsf{handle}\ \square\ \{H_l; \mathsf{ret}\ x \to \Omega\} & \text{for } \alpha^{\bar{l}} \in \mathsf{cv}(e_1) \cup \mathsf{cv}(e_2) \end{array}$$

where $l' \notin \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2)$, $H_l = l_1\colon x,k \to \Omega; \ldots; l_n\colon x,k \to \Omega$ with $\{l_1, \ldots, l_n\} = \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2) \setminus \{l\}$, and $E = \mathsf{handle}\ \square\ \{l'\colon y,k \to y; \mathsf{ret}\ x \to \Omega\}$. Hence, there exists $E_2[x\ v_2]$ such that $e_2 \to^* E_2[x\ v_2]$; we check that (2) $v_1 \equiv_{\mathrm{E}}^{\mathsf{v}} v_2$ and (3) $E_1 \equiv_{\mathrm{E}}^{\mathsf{c}} E_2$.

For (2), let y be a fresh variable and consider the *testing arguments* E and σ such that σ is a closing substitution and $E[v_1\ y]\sigma \Downarrow_{\mathrm{v}}$. Let l' be a fresh label, and define σ' to be equal to σ everywhere, except for x:

$$\sigma'(x) = \lambda z.\mathsf{if}\ \mathsf{do}_{\mathsf{get}}\ ()\ \mathsf{then}\ (\mathsf{do}_{\mathsf{put}}\ \mathsf{false};\ \mathsf{do}_{l'}\ z)\ \mathsf{else}\ \sigma(x)$$

and consider

$$E'' = \mathsf{handle}\ \square\ \{l'\colon z,k \to E[z\ y];\ \mathsf{ret}\ z \to z\}.$$

Then E'_{true} and σ' are the *discriminating arguments*, i.e., $E[v_1\ y]\sigma \Downarrow_{\mathrm{v}}$ iff $E'_{\mathsf{true}}[e_1]\sigma' \Downarrow_{\mathrm{v}}$ iff $E'_{\mathsf{true}}[e_2]\sigma' \Downarrow_{\mathrm{v}}$ iff $E[v_2\ y]\sigma \Downarrow_{\mathrm{v}}$.

Proving (3) requires (a) $E_1[y] \equiv_{\mathrm{E}} E_2[y]$ for a fresh y, and (b) $E_1[\alpha^{\overline{l''}}[\mathsf{do}_{l''}\, y]] \equiv_{\mathrm{E}} E_2[\alpha^{\overline{l}}[\mathsf{do}_l\, y]]$ for any l and fresh $\alpha^{\overline{l}}$ and y. Assuming the same testing arguments E and σ, both cases are proved as in (2), except that in (a) we take

$$E'' = \mathsf{handle}\ \square\ \{l' \colon z, k \to E[k\ y];\ \mathsf{ret}\ z \to z\}$$

and in (b) we take

$$E'' = \mathsf{handle}\ \square\ \{l' \colon z, k \to E[k\ (\alpha^{\overline{l}}[\mathsf{do}_l\, y])];\ \mathsf{ret}\ z \to z\}.$$

Case: $e_1 = E_1^{\,\overline{l}}[\mathsf{do}_l\, v_1]$. To check (1), take σ as follows:

$$\begin{array}{rcll} \sigma(x) & = & \lambda y.\Omega & \text{for } x \in \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\ \sigma(\alpha^{\overline{l'}}) & = & \mathsf{handle}\ \square\ \{H_{l'};\ \mathsf{ret}\ x \to \Omega\} & \text{for } \alpha^{\overline{l'}} \in \mathsf{cv}(e_1) \cup \mathsf{cv}(e_2) \end{array}$$

where $H_{l'} = l_1 \colon x, k \to \Omega; \ldots; l_n \colon x, k \to \Omega$ with $\{l_1, \ldots, l_n\} = \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2) \setminus \{l'\}$, and $E = \mathsf{handle}\ \square\ \{l \colon y, k \to y;\ \mathsf{ret}\ x \to \Omega\}$. Hence, there exists $E_2^{\,\overline{l}}[\mathsf{do}_l\, v_2]$ such that $e_2 \to^* E_2^{\,\overline{l}}[\mathsf{do}_l\, v_2]$; we check that (2) $v_1 \equiv^{\mathsf{v}}_{\mathrm{E}} v_2$ and (3) $E_1^{\,\overline{l}} \equiv^{\mathsf{r}}_{\mathrm{E}} E_2^{\,\overline{l}}$.

Assuming we use a fresh variable x and E, σ as testing arguments, we conclude in the former case by considering $E' = \mathsf{handle}\ \square\ \{l \colon z, k \to E[z\ x];\ \mathsf{ret}\ z \to z\}$ and σ as discriminating arguments.

We prove (3) assuming x fresh and E, σ as testing arguments. Let l', l'' be fresh labels; we define

$$E'' = \mathsf{handle}\ E'''\ \{l' \colon z, k \to E_{l''}[z\ x];\ \mathsf{ret}\ z \to z\}.$$

where

$$E''' = \mathsf{handle}\ \square\ \{l \colon z, k \to \mathsf{if}\ \mathsf{do}_{\mathsf{get}}\ ()\ \mathsf{then}\ (\mathsf{do}_{\mathsf{put}}\ \mathsf{false};\ \mathsf{do}_{l'}\ k)\ \mathsf{else}\ k\ (\mathsf{do}_{l''}\ z);\ \mathsf{ret}\ z \to z\}.$$

and $E_{l''}$ is E where all the occurrences of l are replaced by l''. When l is handled first, we create the discriminating term; subsequent handlings are perfomed by E through l''. Renaming l into a fresh l'' in E is necessary to bypass the handler for l in E'''. The discriminating arguments are E'_{true} and σ.

Case: $e_1 = E_1[\alpha^{\overline{l}}[v_1]]$. Described in details in Section 3.5.

Case: $e_1 = E_1[\alpha^{\overline{l'}}[E_1'^{\,\overline{l}}[\mathsf{do}_l\, v_1]]]$. To check (1), take σ as follows:

$$\begin{array}{rcll} \sigma(x) & = & \lambda y.\Omega & \text{for } x \in \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\ \sigma(\beta^{\overline{l''}}) & = & \mathsf{handle}\ \square\ \{H_{l''};\ \mathsf{ret}\ x \to \Omega\} & \text{for } \beta^{\overline{l''}} \in \mathsf{cv}(e_1) \cup \mathsf{cv}(e_2) \text{ and } \beta^{\overline{l''}} \neq \alpha^{\overline{l'}} \\ \sigma(\alpha^{\overline{l'}}) & = & \mathsf{handle}\ \square\ \{l \colon x, k \to \mathsf{do}_{l'''}\ x;\ \mathsf{ret}\ x \to \Omega\} & \end{array}$$

where $l''' \notin \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2)$, $H_{l''} = l_1 \colon x, k \to \Omega; \ldots; l_n \colon x, k \to \Omega$ with $\{l_1, \ldots, l_n\} = \mathsf{lbl}(e_1) \cup \mathsf{lbl}(e_2) \setminus \{l''\}$, and $E = \mathsf{handle}\ \square\ \{l''' \colon x, k \to x;\ \mathsf{ret}\ x \to \Omega\}$. Hence, there exists $E_2[\alpha^{\overline{l'}}[E_2'^{\,\overline{l}}[\mathsf{do}_l\, v_2]]]$ such that $e_2 \to^* E_2[\alpha^{\overline{l'}}[E_2'^{\,\overline{l}}[\mathsf{do}_l\, v_2]]]$; we check that (2) $v_1 \equiv^{\mathsf{v}}_{\mathrm{E}} v_2$, (3) $E_1'^{\,\overline{l}} \equiv^{\mathsf{r}}_{\mathrm{E}} E_2'^{\,\overline{l}}$, and (4) $E_1 \equiv^{\mathsf{c}}_{\mathrm{E}} E_2$. In each case, we assume x and l'' to be fresh and the testing arguments to be E and σ.

The discriminating arguments for (2) are σ', defined to be equal to σ everywhere, except for $\alpha^{\overline{l'}}$:

$$\sigma'(\alpha^{\overline{l'}}) = \sigma(\alpha^{\overline{l'}})[\mathsf{handle}\ \square\ \{l\colon z,k \to \mathsf{if}\ \mathsf{do}_{\mathsf{get}}\ ()\ \mathsf{then}\ (\mathsf{do}_{\mathsf{put}}\ \mathsf{false};\ \mathsf{do}_{l''}\ z)\ \mathsf{else}\ k\ (\mathsf{do}_{l}\ x);\ \mathsf{ret}\ z \to z\}],$$

and E'_{true} assuming

$$E'' = \mathsf{handle}\ \square\ \{l''\colon z,k \to E[z\ x];\ \mathsf{ret}\ z \to z\}.$$

For (3), we prove $E_1'^{\bar{l}}[x] \equiv_{\mathrm{E}} E_2'^{\bar{l}}[x]$ as in (2), except that we take an extra fresh l''' and define

$$\sigma'(\alpha^{\overline{l'}}) = \sigma(\alpha^{\overline{l'}})[\mathsf{handle}\ \square\ \{l\colon z,k \to \mathsf{if}\ \mathsf{do}_{\mathsf{get}}\ ()\ \mathsf{then}\ (\mathsf{do}_{\mathsf{put}}\ \mathsf{false};\ \mathsf{do}_{l''}\ k)\ \mathsf{else}\ k\ (\mathsf{do}_{l'''}\ x);\ \mathsf{ret}\ z \to z\}]$$

and

$$E'' = \mathsf{handle}\ \square\ \{l''\colon z,k \to E_{l'''}[z\ x];\ \mathsf{ret}\ z \to z\}$$

where $E_{l'''}$ is the context E where the occurrences of l are replaced with l'''.

Proving (4) requires (a) $E_1[x] \equiv_{\mathrm{E}} E_2[x]$ and (b) $E_1[\alpha^{\overline{l'''}}[\mathsf{do}_{l'''}\ z]] \equiv_{\mathrm{E}} E_2[\alpha^{\overline{l'''}}[\mathsf{do}_{l'''}\ x]]$ for any l''' and fresh $\alpha^{\overline{l'''}}$. Assuming the same testing arguments, both cases are proved as in (2), except that in (a) we take

$$E'' = \mathsf{handle}\ \square\ \{l''\colon z,k \to E[k\ x];\ \mathsf{ret}\ z \to z\}$$

and in (b) we take

$$E'' = \mathsf{handle}\ \square\ \{l''\colon z,k \to E[k\ (\alpha^{\overline{l'''}}[\mathsf{do}_{l'''}\ x])];\ \mathsf{ret}\ z \to z\}.$$