

# A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

Dariusz Biernacki, Sergueï Lenglet, Piotr Polesiuk

► **To cite this version:**

Dariusz Biernacki, Sergueï Lenglet, Piotr Polesiuk. A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers. Formal Structures for Computation and Deduction, Jun 2020, Paris, France. 10.4230/LIPIcs.FSCD.2020.7 . hal-02559253

**HAL Id: hal-02559253**

**<https://hal.inria.fr/hal-02559253>**

Submitted on 30 Apr 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 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, 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.2020.3

## 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 `doask () + doask () + 2`, and its meaning is given by a handler, as in

$$\text{handle } \text{do}_{\text{ask}} () + \text{do}_{\text{ask}} () + 2 \{ \text{ask: } x, k \rightarrow k \ 5; \text{ret } y \rightarrow y \}$$

The handler specifies how it interprets the `ask` effect by the expression  $x, k \rightarrow k \ 5$ , 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 `doask () + doask () + 2` eventually reduces to 12. Once the expression inside the handler is a value, it is passed to the *return clause* `ret y → y`, which in our case simply returns the result. Any expression can



© Dariusz Biernacki and Sergueï Lenglet and Piotr Polesiuk;  
licensed under Creative Commons License CC-BY

5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020).

Editor: Zena M. Ariola; Article No. 3; pp. 3:1–3:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

43 be used in an effect handler, including one making use of the continuation several times or  
 44 not at all; for example, in

45  $\text{handle } \text{do}_{\text{ask}} () + \text{do}_{\text{ask}} () + 2 \{ \text{ask}: x, k \rightarrow 13; \text{ret } y \rightarrow y \}$

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

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

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

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

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

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

■ **Figure 1** Syntax of  $\lambda_{\text{eff}}$

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

## 100 2 The Calculus $\lambda_{\text{eff}}$

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

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

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

118 **Reduction semantics.** We fix a call-by-value, left-to-right reduction strategy for  $\lambda_{\text{eff}}$  by  
 119 defining the syntax of evaluation contexts as follows.

120  $\text{ECtx} \ni E ::= \square \mid E e \mid v E \mid \text{do}_l E \mid \text{handle } E \{H; r\}$

### 3:4 A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

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

$$\begin{aligned}
 124 \quad & \text{hl}(\square) \triangleq \emptyset \\
 125 \quad & \text{hl}(E e) \triangleq \text{hl}(E) \\
 126 \quad & \text{hl}(v E) \triangleq \text{hl}(E) \\
 127 \quad & \text{hl}(\text{do}_l E) \triangleq \text{hl}(E) \\
 128 \quad & \text{hl}(\text{handle } E \{l_1: h_1; \dots; l_n: h_n; r\}) \triangleq \text{hl}(E) \cup \{l_1, \dots, l_n\}
 \end{aligned}$$

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

133 The reduction semantics of  $\lambda_{\text{eff}}$  is given by the following rules.

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

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

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

$$145 \quad e \triangleq \text{handle } \text{do}_{\text{ask}} () + \text{do}_{\text{ask}} () + 2 \{\text{ask}: x, k \rightarrow k \ 5; \text{ret } y \rightarrow y\}$$

146 If  $E \triangleq \text{handle } \square \{\text{ask}: x, k \rightarrow k \ 5; \text{ret } y \rightarrow y\}$  and  $z$  is a fresh variable, then  $e$  reduces as follows:

$$\begin{aligned}
 147 \quad & e \rightarrow (\lambda z.E[z + \text{do}_{\text{ask}} () + 2]) \ 5 \\
 148 \quad & \rightarrow \text{handle } 5 + \text{do}_{\text{ask}} () + 2 \{\text{ask}: x, k \rightarrow k \ 5; \text{ret } y \rightarrow y\} \\
 149 \quad & \rightarrow (\lambda z.E[5 + z + 2]) \ 5 \\
 150 \quad & \rightarrow \text{handle } 5 + 5 + 2 \{\text{ask}: x, k \rightarrow k \ 5; \text{ret } y \rightarrow y\} \\
 151 \quad & \rightarrow^* \text{handle } 12 \{\text{ask}: x, k \rightarrow k \ 5; \text{ret } y \rightarrow y\} \\
 152 \quad & \rightarrow 12 \\
 153
 \end{aligned}$$

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

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

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

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

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

170 ► **Definition 3.** *Two expressions  $e_1$  and  $e_2$  are contextually equivalent, written  $e_1 \equiv e_2$ , if*  
171 *for all contexts  $C$ , such that  $C[e_1]$  and  $C[e_2]$  are closed, we have  $C[e_1] \Downarrow_v$  iff  $C[e_2] \Downarrow_v$ .*

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

$$175 \quad C = \text{handle } \square \{l: x, k \rightarrow \Omega; \text{ret } x \rightarrow x\}$$

176 we have  $C[e_1] \Uparrow$ , whereas  $C[v_2] \Downarrow_v$  for any value  $v_2$ , and taking

$$177 \quad C' = \text{handle } \square \{l: x, k \rightarrow x; \text{ret } x \rightarrow x\}$$

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

### 183 **3 Normal-Form Bisimilarity**

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

#### 188 **3.1 Informal Presentation**

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

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

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

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

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

## 230 3.2 Extended Calculus

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

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

$$239 \quad e ::= \dots \mid \alpha^{\bar{l}}[e] \qquad E ::= \dots \mid \alpha^{\bar{l}}[E]$$

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

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

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

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

257 Given an expression  $e$ , a context variable  $\alpha^{\bar{l}}$  and a context  $E^{\bar{l}}$ , we define the *context sub-*  
 258 *stitution*  $e\{E^{\bar{l}}/\alpha^{\bar{l}}\}$  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  
 259 propagated to the sub-expressions in the other cases.

### 260 3.3 Definition

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

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

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

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

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



278 We define progress, bisimulation and bisimilarity using these extensions.

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

- 281 ■ if  $e_1 \rightarrow e'_1$ , then there exists  $e'_2$  such that  $e_2 \rightarrow^* e'_2$  and  $e'_1 \mathcal{T} e'_2$ ;
- 282 ■ if  $e_1 = v_1$ , then there exists  $v_2$  such that  $e_2 \rightarrow^* v_2$  and  $v_1 \mathcal{S}^v v_2$ ;
- 283 ■ if  $e_1 = E_1[x v_1]$ , then there exist  $E_2$  and  $v_2$  such that  $e_2 \rightarrow^* E_2[x v_2]$ ,  $E_1 \mathcal{T}^c E_2$ , and  
 284  $v_1 \mathcal{T}^v v_2$ ;
- 285 ■ if  $e_1 = E_1[\bar{\text{do}}_l v_1]$ , then there exist  $E_2$  and  $v_2$  such that  $e_2 \rightarrow^* E_2[\bar{\text{do}}_l v_2]$ ,  $E_1 \bar{\mathcal{T}}^r E_2$ ,  
 286 and  $v_1 \mathcal{T}^v v_2$ ;
- 287 ■ if  $e_1 = E_1[\alpha^{\bar{l}}[v_1]]$ , then there exist  $E_2$  and  $v_2$  such that  $e_2 \rightarrow^* E_2[\alpha^{\bar{l}}[v_2]]$ ,  $E_1 \mathcal{S}^c E_2$ , and  
 288  $v_1 \mathcal{S}^v v_2$ ;
- 289 ■ if  $e_1 = E_1[\alpha^{\bar{l}}[E_1^{\bar{l}'}[\text{do}_{l'} v_1]]]$  with  $l \neq l'$ , then there exist  $E_2$ ,  $E_2^{\bar{l}'}$ , and  $v_2$  such that  
 290  $e_2 \rightarrow^* E_2[\alpha^{\bar{l}}[E_2^{\bar{l}'}[\text{do}_{l'} v_2]]]$ ,  $E_1 \mathcal{T}^c E_2$ ,  $E_1^{\bar{l}'} \mathcal{T}^r E_2^{\bar{l}'}$ , and  $v_1 \mathcal{T}^v v_2$ ;
- 291 ■ the symmetric of the above conditions on  $e_2$ .

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

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

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

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

$$308 \quad E_1 \triangleq \text{handle } \square \{ \text{ask: } x, k \rightarrow k z; \text{ret } x \rightarrow x \}$$

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

$$312 \quad E_2 \triangleq (\text{handle } \square \{ \text{ask: } x, k \rightarrow \lambda y. k y y; \text{ret } x \rightarrow \lambda y. x \}) z$$

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

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

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

$$322 \quad E_1[\alpha^{\bar{l}}[\text{do}_l x]] \rightarrow^2 E_1[\alpha^{\bar{l}}[z]]$$

$$323 \quad E_2[\alpha^{\bar{l}}[\text{do}_l x]] \rightarrow (\lambda y.(\lambda y'.E_2'[\alpha^{\bar{l}}[y']]) y y) z \rightarrow^2 E_2'[\alpha^{\bar{l}}[z]]z = E_2[\alpha^{\bar{l}}[z]]$$

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

### 328 3.4 Soundness and Up-to Techniques

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

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

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

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

342 In an up-to technique  $(\mathbf{s}, \mathbf{f})$ ,  $\mathbf{s}$  is said *strong* while  $\mathbf{f}$  is said *weak*. Instead of proving directly  
 343 that a pair is an up-to technique, we consider a sufficient criterion based on *respectfulness*<sup>2</sup>  
 344 and the largest respectful pair, called the *diacritical companion*  $(\mathbf{u}, \mathbf{w})$ : if a pair  $(\mathbf{s}, \mathbf{f})$  is below  
 345 the companion, then it is an up-to technique.

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

350  $\blacktriangleright$  **Definition 8.** Let  $\mathbf{f}, \mathbf{g}$  be monotone functions.

- 351  $\blacksquare$   $\mathbf{f}$  passively evolves to  $\mathbf{g}$ , written  $\mathbf{f} \mapsto^p \mathbf{g}$ , if for all  $\mathcal{R}, \mathcal{S}$ ,  $\mathcal{R} \mapsto^p \mathcal{S}$  implies  $\mathbf{f}(\mathcal{R}) \mapsto^p \mathbf{g}(\mathcal{S})$ ;
- 352  $\blacksquare$   $\mathbf{f}$  actively evolves to  $\mathbf{g}$ , written  $\mathbf{f} \mapsto^a \mathbf{g}$ , if for all  $\mathcal{R}, \mathcal{S}$ ,  $\mathcal{R} \mapsto^a \mathcal{S}$  implies  $\mathbf{f}(\mathcal{R}) \mapsto^a \mathbf{g}(\mathcal{S})$ ;
- 353  $\blacksquare$   $\mathbf{f}$  restrictively evolves to  $\mathbf{g}$ , written  $\mathbf{f} \mapsto^{\text{pla}} \mathbf{g}$ , if for all  $\mathcal{R}, \mathcal{S}$ ,  $\mathcal{R} \mapsto^p \mathcal{R} \mapsto^a \mathcal{S}$  implies  
 354  $\mathbf{f}(\mathcal{R}) \mapsto^a \mathbf{g}(\mathcal{S})$ .

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

<sup>2</sup> 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.

### 3:10 A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

359 ► **Definition 9.** A pair of monotone functions  $(s, f)$  diacritically evolves to  $(s', f')$ ,  $(s'', f'')$ ,  
 360 written  $(s, f) \rightsquigarrow (s', f'), (s'', f'')$  if

$$361 \quad s \xrightarrow{p} s' \qquad f \xrightarrow{p} f' \qquad s \xrightarrow{a} s'' \qquad f \xrightarrow{p|a} f''$$

363 A pair  $(s, f)$  is respectful if  $(s, f) \rightsquigarrow (s, f), (s, f)$ . The diacritical companion  $(u, w)$  is the largest  
 364 respectful pair.

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

368 ► **Lemma 10.** The following hold:

- 369 ■ if  $(s, f) \sqsubseteq (u, w)$ , then  $(s, f)$  is an up-to technique;
- 370 ■  $u \sqsubseteq w$ ;
- 371 ■  $w(\approx) = \approx$ .

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

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

- 381 ■ for all  $g \in \{\text{id}, s, u\}$ ,  $g \sqsubseteq S(s)$ ;
- 382 ■ for all  $g \in \{\text{id}, s, u\}$ ,  $g \circ S(s) \sqsubseteq S(s)$ ,  $S(s) \circ g \sqsubseteq S(s)$ ,  $g \cup S(s) \sqsubseteq S(s)$ , and  $S(s) \cup g \sqsubseteq S(s)$ ;
- 383 and  $W(s, f)$  inductively as the smallest function verifying:
- 384 ■ for all  $g \in \{\text{id}, s, f, w\}$ ,  $g \sqsubseteq W(s)$ ;
- 385 ■ for all  $g \in \{\text{id}, s, f, w\}$ ,  $g \circ W(s, f) \sqsubseteq W(s, f)$ ,  $W(s, f) \circ g \sqsubseteq W(s, f)$ ,  $g \cup W(s, f) \sqsubseteq W(s, f)$ ,  
 386 and  $W(s, f) \cup g \sqsubseteq W(s, f)$ ;

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

391 ► **Theorem 11.** Let  $(s, f)$  be monotone functions. If

$$392 \quad s \xrightarrow{p} S(s) \qquad f \xrightarrow{p} S(s) \circ f \circ S(s) \qquad s \xrightarrow{a} W(s, f) \qquad f \xrightarrow{p|a} W(s, f)$$

394 then  $(s, f) \sqsubseteq (u, w)$  and  $(s, f)$  is an up-to technique.

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

400  
 401 We define the up-to functions we consider for  $\lambda_{\text{eff}}$  in Figure 2. The first four are usual  
 402 and can be found in many variants of the  $\lambda$ -calculus [7, 4]. The function  $\text{red}$  is the usual  
 403 bisimulation up to reduction, where expressions can be related after some reduction steps,

Resumption position predicate.				
$\frac{}{\text{resum}(\alpha^{\bar{I}}, x)}$	$\frac{\text{resum}(\alpha^{\bar{I}}, e)}{\text{resum}(\alpha^{\bar{I}}, \lambda x.e)}$	$\frac{\text{resum}(\alpha^{\bar{I}}, e_1) \quad \text{resum}(\alpha^{\bar{I}}, e_2)}{\text{resum}(\alpha^{\bar{I}}, e_1 e_2)}$	$\frac{\text{resum}(\alpha^{\bar{I}}, e)}{\text{resum}(\alpha^{\bar{I}}, \text{do}_l v)}$	
$\frac{\text{resum}(\alpha^{\bar{I}}, e)}{\text{resum}(\alpha^{\bar{I}}, \text{handle } e \{H; \text{ret } x \rightarrow e'\})}$	$\frac{\forall l_i: x_i, k_i \rightarrow e_i \in H, \text{resum}(\alpha^{\bar{I}}, e_i) \quad \text{resum}(\alpha^{\bar{I}}, e')}{\text{resum}(\alpha^{\bar{I}}, e)}$		$\frac{\text{resum}(\alpha^{\bar{I}}, v)}{\text{resum}(\alpha^{\bar{I}}, \alpha^{\bar{I}}[v])}$	
$\frac{\text{resum}(\alpha^{\bar{I}}, v)}{\text{resum}(\alpha^{\bar{I}}, \alpha^{\bar{I}}[\text{do}_l v])}$		$\frac{\text{resum}(\alpha^{\bar{I}}, e) \quad \beta^{\bar{I}} \neq \alpha^{\bar{I}}}{\text{resum}(\alpha^{\bar{I}}, \beta^{\bar{I}}[e])}$		
Up-to techniques.				
$\frac{e_1 \rightarrow^* e'_1 \quad e_2 \rightarrow^* e'_2 \quad e'_1 \mathcal{R} e'_2}{e_1 \text{red}(\mathcal{R}) e_2}$		$\frac{}{e \text{refl}(\mathcal{R}) e}$		
$\frac{e_1 \mathcal{R} e_2 \quad v_1 \mathcal{R}^v v_2}{e_1 \{v_1/x\} \text{subst}(\mathcal{R}) e_2 \{v_2/x\}}$		$\frac{e_1 \mathcal{R} e_2}{\lambda x.e_1 \text{lam}(\mathcal{R}) \lambda x.e_2}$		
$\frac{e_1 \mathcal{R} e_2}{\alpha^{\bar{I}}[e_1] \text{cvar}(\mathcal{R}) \alpha^{\bar{I}}[e_2]}$		$\frac{e_1 \mathcal{R} e_2 \quad E_1^{\bar{I}} \mathcal{R}^c E_2^{\bar{I}}}{e_1 \{E_1^{\bar{I}}/\alpha^{\bar{I}}\} \text{csubst}(\mathcal{R}) e_2 \{E_2^{\bar{I}}/\alpha^{\bar{I}}\}}$		
$\frac{e_1 \mathcal{R} e_2 \quad E_1^{\bar{I}} \mathcal{R}^r E_2^{\bar{I}} \quad \text{resum}(\alpha^{\bar{I}}, e_1) \quad \text{resum}(\alpha^{\bar{I}}, e_2)}{e_1 \{E_1^{\bar{I}}/\alpha^{\bar{I}}\} \text{rsubst}(\mathcal{R}) e_2 \{E_2^{\bar{I}}/\alpha^{\bar{I}}\}}$				

■ **Figure 2** Up-to functions for  $\lambda_{\text{eff}}$

404 while `refl` equates any expression with itself. The function `subst` allows to replace a variable  
 405 in related expressions with related values. Finally, `lam` is compatibility w.r.t.  $\lambda$ -abstraction.

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

413 The functions `cvar`, `csubst`, and `rsubst` can be used to define a more conventional bisimu-  
 414 lation up to evaluation context, similar to the one of the plain  $\lambda$ -calculus [7].

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

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

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

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

420 The proofs for the strong techniques are simple or as in the plain  $\lambda$ -calculus [7]; we sketch  
 421 the proof for `csubst` and `rsubst` in the appendix. It is not surprising that these two functions

### 3:12 A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

are weak, as they essentially behave as bisimulation up to context, which is also weak in the plain  $\lambda$ -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 `csbst` 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  $\lambda$ -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  $C[e_1] \Downarrow_v$ , then  $C[e_2] \Downarrow_v$  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  $\beta$ -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

$$\begin{aligned} e_1 &\triangleq \lambda y. \mathbf{do}_{\text{tick}} (\Delta_y \Delta_y) & \Delta_y &\triangleq \lambda x. (\mathbf{do}_{\text{tick}} y) \lambda z. \mathbf{do}_{\text{tick}} (x x z) \\ e_2 &\triangleq \Theta \Theta & \Theta &\triangleq \lambda x. \lambda y. \mathbf{do}_{\text{tick}} ((\mathbf{do}_{\text{tick}} y) \lambda z. \mathbf{do}_{\text{tick}} (x x y z)) \end{aligned}$$

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

Plugging them with a fresh variable, we obtain two open-stuck terms, meaning that we need to relate the two identical contexts  $\mathbf{do}_{\text{tick}} \square$  and the values  $\lambda z. \mathbf{do}_{\text{tick}} (\Delta_y \Delta_y z)$  and  $\lambda z. \mathbf{do}_{\text{tick}} (\Theta \Theta y z)$ . 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 `csbst`. 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.

$$\begin{aligned} E_{BT} &\triangleq \mathbf{handle} \square \{ \mathbf{fail}: x, k \rightarrow (); \mathbf{flip}: x, k \rightarrow (\lambda z. k \text{ false}) (k \text{ true}); \mathbf{ret} x \rightarrow x \} \\ E_R &\triangleq \mathbf{handle} \square \{ \mathbf{ask}: x, k \rightarrow k z; \mathbf{ret} x \rightarrow x \} \end{aligned}$$

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

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

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

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

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

$$470 (\lambda z.((\lambda y.E_{BT}[E_R[\alpha^{\bar{l}}[y]]]) \text{false})) E_{BT}[E_R[\alpha^{\bar{l}}[\text{true}]]], \text{ and} \\ 471 E_R[(\lambda z.((\lambda y.E_{BT}[\alpha^{\bar{l}}[y]]) \text{false})) E_{BT}[\alpha^{\bar{l}}[\text{true}]]].$$

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

### 479 3.5 Completeness

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

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

487 **► Lemma 18.** If  $e_1 \equiv e_2$ , then  $e_1 \equiv_{\mathbb{E}} e_2$ .

488 **Proof.** Assume that  $e_1 \equiv e_2$  and take any evaluation context  $E$  and closing substitution  $\sigma$ ,  
489 such that  $E[e_1]\sigma \Downarrow_v$ . Then, it must be the case that  $E[e_2]\sigma \Downarrow_v$  as well, since otherwise  $e_1$   
490 and  $e_2$  would be distinguished by the following context:

$$491 C = (\lambda x_1 \dots \lambda x_n.E\sigma) v_1 \dots v_n$$

492 assuming  $\text{dom}(\sigma) = \{x_1, \dots, x_n, \alpha_1, \dots, \alpha_m\}$  and  $\sigma(x_i) = v_i$  for  $1 \leq i \leq n$ .  $\blacktriangleleft$

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

495 **► Lemma 19.**  $\equiv_{\mathbb{E}}$  is a bisimulation.

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

500 **Case:**  $e_1 = E_1[\alpha^{\bar{l}}[v_1]]$  and  $e_1 \equiv_E e_2$ . We need to show that there exist  $E_2$  and  $v_2$  such that:  
 501 (1)  $e_2 \rightarrow^* E_2[\alpha^{\bar{l}}[v_2]]$ , (2)  $v_1 \equiv_E v_2$ , and (3)  $E_1 \equiv_E E_2$ .

502 To prove (1), we take a fresh label  $l'$ , and we define a substitution  $\sigma$  as follows:

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

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

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

$$514 \quad \sigma'(\alpha^{\bar{l}}) = \sigma(\alpha^{\bar{l}})[(\lambda x. \text{if } \text{do}_{\text{get}} () \text{ then } (\text{do}_{\text{put}} \text{false}; \text{do}_{l'} x) \text{ else } x) \square]$$

515 along with

$$\begin{aligned} 516 \quad E'_b &= (\text{handle } E'' \{ \text{get}: x, k \rightarrow \lambda y. k y y; \text{put}: x, k \rightarrow \lambda y. k () x; \text{ret } x \rightarrow \lambda y. x \}) b \\ 517 \quad E'' &= \text{handle } \square \{l': x, k \rightarrow E[x z]; \text{ret } x \rightarrow x\}. \end{aligned}$$

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

$$520 \quad E'_{\text{true}}[e_i]\sigma' \rightarrow^* E'_{\text{false}}[E_i[\text{do}_{l'} v_i]]\sigma' \rightarrow^* E'_{\text{false}}[E[v_i z]]\sigma'$$

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

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

$$532 \quad E'' = \text{handle } \square \{l': x, k \rightarrow E[k z]; \text{ret } x \rightarrow x\}$$

533 and in (b) we take

$$534 \quad E'' = \text{handle } \square \{l': x, k \rightarrow E[k (\alpha^{\bar{l}''}[\text{do}_{l''} z])]; \text{ret } x \rightarrow x\}.$$

535 The remaining cases are proved similarly and can be found in Appendix B.  $\blacktriangleleft$

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

<sup>3</sup> Strictly speaking,  $\sigma'$  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  $\sigma'$  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 `shift0l` where the main reduction rule is:

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

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



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

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

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

## 606 **5 Conclusion**

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

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

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

## References

- 627 —
- 628 1 Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics*  
629 *in Functional Programming*, The University of Texas Year of Programming Series, chapter 4,  
630 pages 65–116. Addison-Wesley, 1990.
- 631 2 Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Environmental  
632 Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. *Logical*  
633 *Methods in Computer Science*, 13(3), 2017.
- 634 3 Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log.*  
635 *Algebr. Meth. Program.*, 84(1):108–123, 2015.
- 636 4 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Bisimulations for delimited-control  
637 operators. *Logical Methods in Computer Science*, 15(2), 2019.
- 638 5 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity  
639 for state. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science*  
640 *and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part*  
641 *of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague,*  
642 *Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer*  
643 *Science*, pages 98–114. Springer, 2019.
- 644 6 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Diacritical companions. In Barbara  
645 König, editor, *Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations*  
646 *of Programming Semantics*, volume 347 of *Electronic Notes in Theoretical Computer Science*,  
647 pages 25–43, London, England, 2019.
- 648 7 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Proving soundness of extensional  
649 normal-form bisimilarities. *Logical Methods in Computer Science*, 15(1), 2019.
- 650 8 Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Handle with care:  
651 relational interpretation of algebraic effects and handlers. *PACMPL*, 2(POPL):8:1–8:30, 2018.
- 652 9 Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Abstracting algebraic  
653 effects. *PACMPL*, 3(POPL):6:1–6:28, 2019.
- 654 10 Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings*  
655 *of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice,  
656 France, June 1990. ACM Press.
- 657 11 Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and  
658 Peter Mager, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of*  
659 *Programming Languages*, pages 180–190, San Diego, California, January 1988. ACM Press.
- 660 12 Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power  
661 of user-defined effects: effect handlers, monadic reflection, delimited control. *PACMPL*,  
662 1(ICFP):13:1–13:29, 2017.
- 663 13 Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for  
664 algebraic effects. In Jean-Pierre Jouannaud, editor, *Proceedings of the 25th IEEE Symposium*  
665 *on Logic in Computer Science (LICS 2010)*, pages 209–218, Edinburgh, UK, July 2010. IEEE  
666 Computer Society Press.
- 667 14 Ugo Dal Lago and Francesco Gavazzo. Effectful normal form bisimulation. In Luís Caires,  
668 editor, *Programming Languages and Systems - 28th European Symposium on Programming,*  
669 *ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of*  
670 *Software, ETAPS 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*,  
671 pages 263–292, Prague, Czech Republic, April 2019. Springer.
- 672 15 Ugo Dal Lago, Francesco Gavazzo, and Paul Blain Levy. Effectful applicative bisimilarity:  
673 Monads, relators, and Howe’s method. In *32nd Annual ACM/IEEE Symposium on Logic in*  
674 *Computer Science, LICS 2017*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE Computer  
675 Society.
- 676 16 Ugo Dal Lago, Francesco Gavazzo, and Ryo Tanaka. Effectful applicative similarity for  
677 call-by-name lambda calculi. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi  
678 Sauro, editors, *Joint Proceedings of the 18th Italian Conference on Theoretical Computer*

- 679 *Science and the 32nd Italian Conference on Computational Logic co-located with the 2017 IEEE*  
680 *International Workshop on Measurements and Networking (2017 IEEE M&N)*, volume 1949 of  
681 *CEUR Workshop Proceedings*, pages 87–98, Naples, Italy, September 2017. CEUR-WS.org.
- 682 **17** Søren B. Lassen. Eager normal form bisimulation. In Prakash Panangaden, editor, *Proceedings*  
683 *of the Twentieth Annual IEEE Symposium on Logic in Computer Science*, pages 345–354,  
684 Chicago, IL, June 2005. IEEE Computer Society Press.
- 685 **18** Jean-Marie Madiot. *Higher-order languages: dualities and bisimulation enhancements*. PhD  
686 thesis, Université de Lyon and Università di Bologna, 2015.
- 687 **19** Cristina Matache and Sam Staton. A sound and complete logic for algebraic effects. In Mikolaj  
688 Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation*  
689 *Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European*  
690 *Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, volume  
691 11425 of *Lecture Notes in Computer Science*, pages 382–399, Prague, Czech Republic, April  
692 2019. Springer.
- 693 **20** James H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, Massachu-  
694 sets Institute of Technology, 1968.
- 695 **21** Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Typed equivalence of effect handlers  
696 and delimited control. In Herman Geuvers, editor, *4th International Conference on Formal*  
697 *Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPICs*, pages 30:1–  
698 30:16, Dortmund, Germany, June 2019. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.
- 699 **22** Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in*  
700 *Computer Science*, 9(4), 2013.
- 701 **23** Damien Pous. Coinduction all the way up. In Martin Grohe, Eric Koskinen, and Natarajan  
702 Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer*  
703 *Science, LICS '16*, pages 307–316, New York, NY, USA, July 2016. ACM.
- 704 **24** John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason,  
705 editor, *Information Processing 83*, pages 513–523, Paris, France, 1983. IFIP.
- 706 **25** Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. In Andre Scedrov,  
707 editor, *LICS'92*, pages 102–109, Santa Cruz, California, June 1992. IEEE Computer Society.
- 708 **26** Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer*  
709 *Science*, 8(5):447–479, October 1998.
- 710 **27** Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-  
711 order languages. *ACM Transactions on Programming Languages and Systems*, 33(1):1–69,  
712 January 2011.
- 713 **28** Chung-chieh Shan. A static simulation of dynamic delimited control. *Higher-Order and*  
714 *Symbolic Computation*, 20(4):371–401, 2007.
- 715 **29** Alex Simpson and Niels F. W. Voorneveld. Behavioural equivalence via modalities for algebraic  
716 effects. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European*  
717 *Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on*  
718 *Theory and Practice of Software, ETAPS 2018, Proceedings*, volume 10801 of *Lecture Notes in*  
719 *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  $\lambda$ -calculus [7]. In particular, we use the fact that

► **Lemma 21.** `subst`  $\sqsubseteq$  `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.

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

We let  $\sigma$  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  $\sigma$  to  $E^{\bar{l}}$ . We define  $\mathcal{R}^\sigma$  pairwise such that we have either  $\sigma_1(\alpha^{\bar{l}}) \mathcal{R}^c \sigma_2(\alpha^{\bar{l}})$ , or  $\sigma_1(\alpha^{\bar{l}}) \mathcal{R}^r \sigma_2(\alpha^{\bar{l}})$  with `resum`( $\alpha^{\bar{l}}, e_1$ ) and `resum`( $\alpha^{\bar{l}}, e_2$ ).

► **Lemma 22.** `ssubst`  $\sqsubseteq$  `w`

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

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

If  $\sigma_1(\alpha^{\bar{l}}) = \beta^{\bar{l}}[\square]$ , then from  $\beta^{\bar{l}}[x] \mathcal{R} \sigma_2(\alpha^{\bar{l}})[x]$ , there exist  $E_2'$  and  $w$  such that  $\sigma_2(\alpha^{\bar{l}})[x] \rightarrow^* E_2'[\beta^{\bar{l}}[w]]$ ,  $\square \mathcal{R}^c E_2'$ , and  $x \mathcal{R}^v w$ . Therefore we have  $e_2 \sigma_2 \rightarrow^* E_2 \sigma_2[\sigma_2(\alpha^{\bar{l}})[v_2 \sigma_2]] \rightarrow^* E_2 \sigma_2[E_2'[\beta^{\bar{l}}[w\{v_2 \sigma_2/x\}]]]$ , 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  $v_1 \sigma_1 \text{ssubst}(\text{subst}(\mathcal{R})) w\{v_2 \sigma_2/x\}$ . For the contexts, we first relate  $E_1 \sigma_1[y]$  and  $E_2 \sigma_2[E_2'[y]]$  for a fresh  $y$ . Because  $\square \mathcal{R}^c E_2'$ , there exists  $w'$  such that  $E_2'[y] \rightarrow^* w'$  and  $y \mathcal{R}^v w'$ . As a result, we have  $E_2 \sigma_2[E_2'[y]] \rightarrow^* E_2 \sigma_2[w']$ , and therefore  $E_1 \sigma_1[y] \text{red}(\text{ssubst}(\text{subst}(\mathcal{R}))) E_2 \sigma_2[E_2'[y]]$ , which is what we need. Then we must relate  $E_1 \sigma_1[\gamma^{\bar{l}'}[\text{do}_{l'} y]]$  and  $E_2 \sigma_2[E_2'[\gamma^{\bar{l}'}[\text{do}_{l'} y]]]$  for any  $l'$  and fresh  $\gamma^{\bar{l}'}$  and  $y$ . Because  $\square \mathcal{R}^c E_2'$ , there exist  $E_2''^{\bar{l}'}$  and  $w'$  such that  $E_2'[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \rightarrow^* E_2''^{\bar{l}'}[\text{do}_{l'} w']$ ,  $\gamma^{\bar{l}'}[\square] \mathcal{S}^r E_2''^{\bar{l}'}$ , and  $y \mathcal{S}^v w'$ . From  $E_1 \mathcal{R}^c E_2$ , we get  $E_1[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \mathcal{R} E_2[\gamma^{\bar{l}'}[\text{do}_{l'} y]]$ , so if  $E_1[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \rightarrow e_1'$  for some  $e_1'$  (the case where  $l'$  is not handled is not interesting), then there exists  $e_2'$  such that  $E_2[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \rightarrow^* e_2'$  and  $e_1' \mathcal{S} e_2'$ . Therefore,  $E_2 \sigma_2[E_2'[\gamma^{\bar{l}'}[\text{do}_{l'} y]]] \rightarrow^* E_2 \sigma_2[E_2''^{\bar{l}'}[\text{do}_{l'} w']] \rightarrow^* e_2' \sigma_2'\{w'/y\}$  where  $\sigma_2'(\gamma^{\bar{l}'}) = E_2''^{\bar{l}'}$  and is equal

758 to  $\sigma_2$  otherwise. Because  $E_1\sigma_1[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \rightarrow e_1\sigma_1 = e_1\sigma_1'\{y/y\}$  where  $\sigma_1'(\gamma^{\bar{l}'}[\square]) = \gamma^{\bar{l}'}[\square]$  and is  
 759 equal to  $\sigma_1$  otherwise, we deduce  $E_1\sigma_1[\gamma^{\bar{l}'}[\text{do}_{l'} y]] \text{red}(\text{subst}(\text{ssubst}(\mathcal{S}))) E_2\sigma_2[E_2'\gamma^{\bar{l}'}[\text{do}_{l'} y]]$   
 760 which is enough to conclude.

761 Suppose  $e_1 = E_1[\alpha^{\bar{l}'}[E_1^{\bar{l}'}[\text{do}_{l'} v_1]]]$  with  $l \neq l'$  and  $\alpha^{\bar{l}'} \in \text{dom}(\sigma_1)$ ; then there exist  $E_2$ ,  
 762  $E_2^{\bar{l}'}$ , and  $v_2$  such that  $e_2 \rightarrow^* E_2[\alpha^{\bar{l}'}[E_2^{\bar{l}'}\{\text{do}_{l'} v_2\}]]$ ,  $E_1 \mathcal{T}^c E_2$ ,  $E_1^{\bar{l}'} \mathcal{T}^r E_2^{\bar{l}'}$ , and  $v_1 \mathcal{T}^v v_2$ .  
 763 From  $\sigma_1(\alpha^{\bar{l}'}) \mathcal{R}^c \sigma_2(\alpha^{\bar{l}'})$ , we get  $\sigma_1(\alpha^{\bar{l}'})[\gamma^{\bar{l}'}[\text{do}_{l'} x]] \mathcal{R} \sigma_2(\alpha^{\bar{l}'})[\gamma^{\bar{l}'}[\text{do}_{l'} x]]$  for fresh  $\gamma^{\bar{l}'}$  and  $x$ . If  
 764  $\sigma_1(\alpha^{\bar{l}'})[\gamma^{\bar{l}'}[\text{do}_{l'} x]] \rightarrow e_1'$  for some  $e_1'$ , then there exists  $e_2'$  such that  $\sigma_2(\alpha^{\bar{l}'})[\gamma^{\bar{l}'}[\text{do}_{l'} x]] \rightarrow^* e_2'$  and  
 765  $e_1' \mathcal{S} e_2'$ . Then  $e_1\sigma_1 \rightarrow E_1[e_1'\{v_1/x\}\{E_1^{\bar{l}'}/\gamma^{\bar{l}'}\}]\sigma_1$  and  $e_2\sigma_2 \rightarrow^* E_2[e_2'\{v_2/x\}\{E_2^{\bar{l}'}/\gamma^{\bar{l}'}\}]\sigma_2$ ,  
 766 and the resulting expressions are in  $\text{ssubst}(\text{ssubst}(\text{cvar}(\text{ssubst}(\text{subst}(\mathcal{S}))))))$ , which is fine,  
 767 because we are in an active clause.  $\blacktriangleleft$

## 768 B Completeness Proof Sketch

769 The proof proceeds as described in Section 3.5: given  $e_1 \equiv_E e_2$ , we check that for each  
 770 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  
 771 normal form of the same kind, and the normal forms can be decomposed into related sub-parts.  
 772 For each case, we give the substitution  $\sigma$  and the context  $E$  enforcing (1). Checking that  
 773 related sub-parts are contextually equivalent relies in most cases on an encoding of a mutable  
 774 state using handlers, as in Section 3.5. In all the subcases below, we assume the labels `get`  
 775 and `put` to be fresh, and given a boolean  $b$  and a context  $E''$ , we define

$$776 E'_b = (\text{handle } E'' \{ \text{get}: z, k \rightarrow \lambda y. k \ y \ y; \text{put}: z, k \rightarrow \lambda y. k \ () \ z; \text{ret } z \rightarrow \lambda y. z \}) b$$

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

778 **Case:**  $e_1 \rightarrow e_1'$ . Because the reduction is deterministic, we still have  $e_1' \equiv_E e_2$ .

779 **Case:**  $e_1 = v_1$ . To check (1), take  $\sigma$  as follows:

$$780 \begin{aligned} \sigma(x) &= \lambda y. \Omega && \text{for } x \in \text{fv}(e_1) \cup \text{fv}(e_2) \\ \sigma(\alpha^{\bar{l}'}) &= \text{handle } \square \{ H_l; \text{ret } x \rightarrow \Omega \} && \text{for } \alpha^{\bar{l}'} \in \text{cv}(e_1) \cup \text{cv}(e_2) \end{aligned}$$

781 where  $H_l = l_1: x, k \rightarrow \Omega; \dots; l_n: x, k \rightarrow \Omega$  with  $\{l_1, \dots, l_n\} = \text{lbl}(e_1) \cup \text{lbl}(e_2) \setminus \{l\}$ , and  $E = \square$ .  
 782 Hence, there exists  $v_2$  such that  $e_2 \rightarrow^* v_2$ ; we check that  $v_1 \equiv_E^v v_2$ .

783 Let  $x$  be a fresh variable,  $E$  a context, and  $\sigma$  a closing substitution such that  $E[v_1 x]\sigma \Downarrow_v$ .  
 784 Then  $E[e_2 x]\sigma \rightarrow^* E[v_2 x]\sigma$  and since  $e_1 \equiv_E e_2$ , we also have  $E[v_2 x]\sigma \Downarrow_v$ .

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

$$786 \begin{aligned} \sigma(z) &= \lambda y. \Omega && \text{for } z \in \text{fv}(e_1) \cup \text{fv}(e_2) \setminus \{x\} \\ \sigma(x) &= \lambda y. \text{do}_{l'} \lambda z. z \\ \sigma(\alpha^{\bar{l}'}) &= \text{handle } \square \{ H_l; \text{ret } x \rightarrow \Omega \} && \text{for } \alpha^{\bar{l}'} \in \text{cv}(e_1) \cup \text{cv}(e_2) \end{aligned}$$

787 where  $l' \notin \text{lbl}(e_1) \cup \text{lbl}(e_2)$ ,  $H_l = l_1: x, k \rightarrow \Omega; \dots; l_n: x, k \rightarrow \Omega$  with  $\{l_1, \dots, l_n\} = \text{lbl}(e_1) \cup$   
 788  $\text{lbl}(e_2) \setminus \{l'\}$ , and  $E = \text{handle } \square \{ l': y, k \rightarrow y; \text{ret } x \rightarrow \Omega \}$ . Hence, there exists  $E_2[x v_2]$  such that  
 789  $e_2 \rightarrow^* E_2[x v_2]$ ; we check that (2)  $v_1 \equiv_E^v v_2$  and (3)  $E_1 \equiv_E^c E_2$ .

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

$$793 \sigma'(x) = \lambda z. \text{if } \text{do}_{\text{get}} \ () \ \text{then } (\text{do}_{\text{put}} \ \text{false}; \text{do}_{l'} \ z) \ \text{else } \sigma(x)$$

794 and consider

$$795 \quad E'' = \text{handle } \square \{l': z, k \rightarrow E[z y]; \text{ret } z \rightarrow z\}.$$

796 Then  $E'_{\text{true}}$  and  $\sigma'$  are the *discriminating arguments*, i.e.,  $E[v_1 y]\sigma \Downarrow_v$  iff  $E'_{\text{true}}[e_1]\sigma' \Downarrow_v$  iff  
797  $E'_{\text{true}}[e_2]\sigma' \Downarrow_v$  iff  $E[v_2 y]\sigma \Downarrow_v$ .

798 Proving (3) requires (a)  $E_1[y] \equiv_E E_2[y]$  for a fresh  $y$ , and (b)  $E_1[\alpha^{\bar{l}'}[\text{do}_{l''} y]] \equiv_E$   
799  $E_2[\alpha^{\bar{l}'}[\text{do}_l y]]$  for any  $l$  and fresh  $\alpha^{\bar{l}'}$  and  $y$ . Assuming the same testing arguments  $E$  and  $\sigma$ ,  
800 both cases are proved as in (2), except that in (a) we take

$$801 \quad E'' = \text{handle } \square \{l': z, k \rightarrow E[k y]; \text{ret } z \rightarrow z\}$$

802 and in (b) we take

$$803 \quad E'' = \text{handle } \square \{l': z, k \rightarrow E[k (\alpha^{\bar{l}'}[\text{do}_l y])]; \text{ret } z \rightarrow z\}.$$

804 **Case:**  $e_1 = E_1^{\bar{l}'}[\text{do}_l v_1]$ . To check (1), take  $\sigma$  as follows:

$$805 \quad \begin{aligned} \sigma(x) &= \lambda y. \Omega && \text{for } x \in \text{fv}(e_1) \cup \text{fv}(e_2) \\ \sigma(\alpha^{\bar{l}'}) &= \text{handle } \square \{H_{l''}; \text{ret } x \rightarrow \Omega\} && \text{for } \alpha^{\bar{l}'} \in \text{cv}(e_1) \cup \text{cv}(e_2) \end{aligned}$$

806 where  $H_{l''} = l_1: x, k \rightarrow \Omega; \dots; l_n: x, k \rightarrow \Omega$  with  $\{l_1, \dots, l_n\} = \text{lbl}(e_1) \cup \text{lbl}(e_2) \setminus \{l'\}$ , and  $E =$   
807  $\text{handle } \square \{l: y, k \rightarrow y; \text{ret } x \rightarrow \Omega\}$ . Hence, there exists  $E_2^{\bar{l}'}[\text{do}_l v_2]$  such that  $e_2 \rightarrow^* E_2^{\bar{l}'}[\text{do}_l v_2]$ ;  
808 we check that (2)  $v_1 \equiv_E v_2$  and (3)  $E_1^{\bar{l}'} \equiv_E E_2^{\bar{l}'}$ .

809 Assuming we use a fresh variable  $x$  and  $E, \sigma$  as testing arguments, we conclude in the  
810 former case by considering  $E' = \text{handle } \square \{l: z, k \rightarrow E[z x]; \text{ret } z \rightarrow z\}$  and  $\sigma$  as discriminating  
811 arguments.

812 We prove (3) assuming  $x$  fresh and  $E, \sigma$  as testing arguments. Let  $l', l''$  be fresh labels;  
813 we define

$$814 \quad E'' = \text{handle } E''' \{l': z, k \rightarrow E_{l''}[z x]; \text{ret } z \rightarrow z\}.$$

815 where

$$816 \quad E''' = \text{handle } \square \{l: z, k \rightarrow \text{if } \text{do}_{\text{get}}() \text{ then } (\text{do}_{\text{put}} \text{ false}; \text{do}_{l'} k) \text{ else } k (\text{do}_{l''} z); \text{ret } z \rightarrow z\}.$$

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

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

822 **Case:**  $e_1 = E_1[\alpha^{\bar{l}'}[E_1^{\bar{l}'}[\text{do}_l v_1]]]$ . To check (1), take  $\sigma$  as follows:

$$823 \quad \begin{aligned} \sigma(x) &= \lambda y. \Omega && \text{for } x \in \text{fv}(e_1) \cup \text{fv}(e_2) \\ \sigma(\beta^{\bar{l}''}) &= \text{handle } \square \{H_{l''}; \text{ret } x \rightarrow \Omega\} && \text{for } \beta^{\bar{l}''} \in \text{cv}(e_1) \cup \text{cv}(e_2) \text{ and } \beta^{\bar{l}''} \neq \alpha^{\bar{l}'} \\ \sigma(\alpha^{\bar{l}'}) &= \text{handle } \square \{l: x, k \rightarrow \text{do}_{l''} x; \text{ret } x \rightarrow \Omega\} \end{aligned}$$

824 where  $l'' \notin \text{lbl}(e_1) \cup \text{lbl}(e_2)$ ,  $H_{l''} = l_1: x, k \rightarrow \Omega; \dots; l_n: x, k \rightarrow \Omega$  with  $\{l_1, \dots, l_n\} = \text{lbl}(e_1) \cup$   
825  $\text{lbl}(e_2) \setminus \{l''\}$ , and  $E = \text{handle } \square \{l''': x, k \rightarrow x; \text{ret } x \rightarrow \Omega\}$ . Hence, there exists  $E_2[\alpha^{\bar{l}'}[E_2^{\bar{l}'}[\text{do}_l v_2]]]$

### 3:22 A Complete Normal-Form Bisimilarity for Algebraic Effects and Handlers

826 such that  $e_2 \rightarrow^* E_2[\alpha^{\bar{l}}[E_2^{\bar{l}}[\text{do}_l v_2]]]$ ; we check that (2)  $v_1 \equiv_E^v v_2$ , (3)  $E_1^{\bar{l}} \equiv_E^r E_2^{\bar{l}}$ , and (4)  
 827  $E_1 \equiv_E^c E_2$ . In each case, we assume  $x$  and  $l''$  to be fresh and the testing arguments to be  $E$   
 828 and  $\sigma$ .

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

$$831 \quad \sigma'(\alpha^{\bar{l}}) = \sigma(\alpha^{\bar{l}})[\text{handle } \square \{l: z, k \rightarrow \text{if } \text{do}_{\text{get}}() \text{ then } (\text{do}_{\text{put}} \text{ false}; \text{do}_{l''} z) \text{ else } k(\text{do}_l x); \text{ret } z \rightarrow z\}],$$

832 and  $E'_{\text{true}}$  assuming

$$833 \quad E'' = \text{handle } \square \{l'': z, k \rightarrow E[z x]; \text{ret } z \rightarrow z\}.$$

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

$$836 \quad \sigma'(\alpha^{\bar{l}}) = \sigma(\alpha^{\bar{l}})[\text{handle } \square \{l: z, k \rightarrow \text{if } \text{do}_{\text{get}}() \text{ then } (\text{do}_{\text{put}} \text{ false}; \text{do}_{l''} k) \text{ else } k(\text{do}_{l'''} x); \text{ret } z \rightarrow z\}]$$

837 and

$$838 \quad E'' = \text{handle } \square \{l'': z, k \rightarrow E_{l'''}[z x]; \text{ret } z \rightarrow z\}$$

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

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

$$843 \quad E'' = \text{handle } \square \{l'': z, k \rightarrow E[k x]; \text{ret } z \rightarrow z\}$$

844 and in (b) we take

$$845 \quad E'' = \text{handle } \square \{l'': z, k \rightarrow E[k(\alpha^{\bar{l}'''}[\text{do}_{l'''} x])]; \text{ret } z \rightarrow z\}.$$