PROVING SOUNDNESS
OF EXTENSIONAL NORMAL-FORM BISIMILARITIES

DARIUSZ BIERNACKI [a], SERGUEÏ LENGLET [b], AND PIOTR POLESIUK [a]

[a] University of Wrocław, Wrocław, Poland
  e-mail address: dabi@cs.uni.wroc.pl
  e-mail address: ppolesiuk@cs.uni.wroc.pl

[b] Université de Lorraine, Nancy, France
  e-mail address: serguei.lenglet@univ-lorraine.fr

ABSTRACT. Normal-form bisimilarity is a simple, easy-to-use behavioral equivalence that relates terms in λ-calculi by decomposing their normal forms into bisimilar subterms. Moreover, it typically allows for powerful up-to techniques, such as bisimulation up to context, which simplify bisimulation proofs even further. However, proving soundness of these relations becomes complicated in the presence of η-expansion and usually relies on ad hoc proof methods which depend on the language. In this paper we propose a more systematic proof method to show that an extensional normal-form bisimilarity along with its corresponding up to context technique are sound. We illustrate our technique with three calculi: the call-by-value λ-calculus, the call-by-value λ-calculus with the delimited-control operators `shift` and `reset`, and the call-by-value λ-calculus with the abortive control operators `call/cc` and `abort`. In the first two cases, there was previously no sound up to context technique validating the η-law, whereas no theory of normal-form bisimulations for a calculus with `call/cc` and `abort` has been presented before. Our results have been fully formalized in the Coq proof assistant.

1. INTRODUCTION

In formal languages inspired by the λ-calculus, the behavioral equivalence of choice is usually formulated as a Morris-style contextual equivalence [28]: two terms are equivalent if they behave the same in any context. This criterion captures quite naturally the idea that replacing a term by an equivalent one in a bigger program should not affect the behavior of the whole program. However, the quantification over contexts makes contextual equivalence hard to use in practice to prove the equivalence of two given terms. Therefore, it is common to look for easier-to-use, *sound* alternatives that are at least included in contextual equivalence, such as coinductively defined *bisimilarities*.

Different styles of bisimilarities have been defined for the λ-calculus, including *applicative bisimilarity* [1], *normal-form bisimilarity* [21] (originally called *open bisimilarity* in [32]), and *environmental bisimilarity* [33]. Applicative and environmental bisimilarities compare terms by applying them to function arguments, while normal-form bisimilarity reduces terms to normal forms, which are then decomposed into bisimilar subterms. As we can see, applicative and environmental bisimilarities still rely on some form of quantification over arguments, which is not the case of normal-form bisimilarity. As a drawback, the latter is usually not *complete* w.r.t. contextual equivalence—there exist contextually equivalent terms that are not normal-form bisimilar—while the former are. Like environmental bisimilarity, normal-form bisimilarity usually allows for *up-to techniques* [30], relations which simplify equivalence proofs of terms by having less requirements than regular bisimilarities. For example, reasoning up to context allows to forget about a common context: to equate $C[t]$ and $C[s]$, it is enough to relate t and s with a bisimulation up to context.

In the call-by-value λ-calculus, the simplest definition of normal-form bisimilarity compares values by equating a variable only with itself, and a λ-abstraction only with a λ-abstraction such that their bodies are bisimilar. Such a definition does not respect call-by-value η-expansion, since it distinguishes x from $\lambda y.x\,y$. A less discriminating definition instead compares values by applying them to a fresh variable, thus relating $\lambda y.v\,y$ and v for any value v such that y is not free in v: given a fresh z, $(\lambda y.v\,y)\,z$ reduces to $v\,z$. Such a bisimilarity, that we call *extensional bisimilarity*,[1] relates more contextually equivalent terms, but proving its soundness as well as proving the soundness of its up-to techniques is more difficult, and usually requires ad hoc proof methods, as we detail in the related work section (Section 2).

Madiot et al. [26] propose a framework where proving the soundness of up-to techniques is quite uniform and simpler. It also allows to factorize proofs, since showing that reasoning up to context is sound directly implies that the corresponding bisimilarity is a congruence, which is the main property needed for proving its soundness. Madiot et al. apply the method to environmental bisimilarities for the plain call-by-name λ-calculus and for a call-by-value λ-calculus with references, as well as to a bisimilarity for the π-calculus. In a subsequent work [3], we extend this framework to define environmental bisimilarities for a call-by-value λ-calculus with multi-prompted delimited-control operators. We propose a distinction between strong and regular up-to techniques, where regular up-to techniques cannot be used in certain bisimilarity tests, while strong ones can always be used. This distinction allows to prove sound more powerful up-to techniques, by forbidding their use in cases where it would be unsound to apply them.

So far, the method developed in [26, 3] have been used in the λ-calculus only for environmental bisimilarities. In this paper, we show that our extended framework [3] can also be used to prove the soundness of extensional normal-form bisimilarities and their corresponding bisimulation up to context. We first apply it to the plain call-by-value λ-calculus, in which an extensional normal-form bisimilarity, albeit without a corresponding bisimulation up to context, have already been proved sound [21], to show how our framework allows to prove soundness for both proof techniques at once. We then consider a call-by-value λ-calculus with the delimited-control operators `shift` and `reset` [12], for which there has been no sound bisimulation up to context validating the η-law either, and we show that

---

[1]Lassen uses the term *bisimilarity up to η* [20] for a normal-form bisimilarity that validates the η-law, but we prefer the term *extensional bisimilarity* so that there is no confusion with notions referring to up-to techniques such as bisimulation up to context.

our method applies seamlessly in that setting as well. Finally, we address a calculus of abortive control, i.e., the call-by-value λ-calculus with `call/cc` and `abort` [15, 16]. If a normal-form bisimilarity has been defined for the $\lambda\mu$-calculus [34], a more expressive calculus with abortive control, no theory of normal-form bisimulations has been proposed so far for `call/cc` and `abort` themselves. This confirms the robustness of the proof method we advocate in this article, but it also provides a new operational technique for reasoning about classical calculi of abortive continuations introduced by Felleisen et al.

Our results have been fully formalized in the Coq proof assistant. The goal of the formalization is twofold: to increase our confidence in the correctness of the proof method, but also to leave most of the routine and repeatable compatibility proofs outside of the article proper. While the proof method is introduced and illustrated in detail in the main body of the paper, the interested reader can consult the formalization, commented and with pointers to the paper, for the complete development (excluding the practical examples that are presented here for illustration purposes only). The Coq formalization, available at `https://bitbucket.org/pl-uwr/diacritical`, uses a de Bruijn representation for λ-terms, where the de Bruijn indices are encoded using nested datatypes [11].

The paper is organized as follows: in Section 2, we discuss the previous proofs of soundness of extensional normal-form bisimilarities. In Section 3, we present the proof method for the call-by-value λ-calculus, that we then apply to the λ-calculus with delimited control in Section 4, and to the λ-calculus with abortive control in Section 5. We conclude in Section 6. Compared to the conference article [10], Section 5 is entirely new, whereas minor revisions have been done to the remaining sections.

2. Related Work

Normal-form bisimilarity has been first introduced by Sangiorgi [32] and has then been defined for many variants of the λ-calculus, considering η-expansion [20, 21, 23, 34, 24, 25, 8, 9] or not [19, 22]. In this section we focus on the articles treating the η-law, and in particular on the congruence and soundness proofs presented therein.

In [20], Lassen defines several equivalences for the call-by-name λ-calculus, depending on the chosen semantics. He defines *head-normal-form (hnf) bisimulation* and *hnf bisimulation up to η* for the semantics based on reduction to head normal form (where η-expansion applies to any term t, not only to a value as in the call-by-value λ-calculus), and *weak-head-normal-form (whnf) bisimulation* based on reduction to weak head normal form. (It does not make sense to consider a *whnf bisimulation up to η*, since it would be unsound, e.g., it would relate a non-terminating term Ω with a normal form $\lambda x.\Omega\, x$.) The paper also defines a bisimulation up to context for each bisimilarity.

The congruence proofs for the three bisimilarities follow from the main lemma stating that if a relation is a bisimulation, then so is its substitutive and context closure. The lemma is proved by nested induction on the definition of the closure and on the number of steps in the evaluation of terms to normal forms. It can be easily strengthened to prove the soundness of a bisimulation up to context: if a relation is a bisimulation up to context, then its substitutive and context closure is a bisimulation. The nested induction proof method has been then applied to prove congruence for a whnf bisimilarity for the call-by-name $\lambda\mu$-calculus [19] (a calculus with continuations), an extensional hnf bisimilarity for the call-by-name λ-calculus with pairs [23], and a whnf bisimilarity for a call-by-name λ-calculus

with McCarthy's ambiguous choice (`amb`) operator [22]. These papers do not define any corresponding bisimulation up to context.

Lassen uses another proof technique in [21], where he defines an *eager normal form (enf) bisimilarity* and an *enf bisimilarity up to η*.[2] Lassen shows that the bisimilarities correspond to Böhm trees equivalence (up to η) after a continuation-passing style (CPS) translation, and then he deduces congruence of the enf bisimilarities from the congruence of the Böhm trees equivalence. A CPS-translation based technique has also been used in [23] to prove congruence of the extensional bisimilarity for the call-by-name λ-calculus (also with surjective pairing), the $\lambda\mu$-calculus, and the $\Lambda\mu$-calculus. Unlike the nested induction proof method, this technique does not extend to a soundness proof of a bisimulation up to context.

In [21], Lassen claims that *"It is also possible to prove congruence of enf bisimilarity and enf bisimilarity up to η directly like the congruence proofs for other normal form bisimilarities (tree equivalences) in [20], although the congruence proofs (...) require non-trivial changes to the relational substitutive context closure operation in op.cit. (...) Moreover, from the direct congruence proofs, we can derive bisimulation "up to context" proof principles like those for other normal form bisimilarities in op.cit."* To our knowledge, such a proof is not published anywhere; we tried to carry out the congruence proof by following this comment, but we do not know how to conclude in the case of enf bisimilarity up to η. We discuss what the problem is at the end of the proof of Lemma 3.20.

Støvring and Lassen [34] define extensional enf bisimilarities for three calculi: $\lambda\mu$ (continuations), $\lambda\rho$ (mutable state), and $\lambda\mu\rho$ (continuations and mutable state). The congruence proof is rather convoluted and is done in two stages: first, prove congruence of a non-extensional bisimilarity using the nested induction of [20], then extend the result to the extensional bisimilarity by a syntactic translation that takes advantage of an infinite η-expansion combinator. The paper does not mention bisimulation up to context.

Lassen and Levy [24, 25] define a normal-form bisimilarity for a CPS calculus called JWA equipped with a rich type system (including product, sum, recursive types; [25] adds existential types). The bisimilarity respects the η-law, and the congruence proof is done in terms of game semantics notions. Again, these papers do not mention bisimulation up to context.

In a previous work [8], we define extensional enf bisimilarities and bisimulations up to context for a call-by-value λ-calculus with delimited-control operators. The (unpublished) congruence and soundness proofs follow Lassen [20], but are incorrect: one case in the induction, that turns out to be problematic, has been forgotten. In [9] we fix the congruence proof of the extensional bisimilarity, by doing a nested induction on a different notion of closure than Lassen. This approach fails when proving soundness of a bisimulation up to context, and therefore bisimulation up to context does not respect the η-law in [9].

To summarize:

- The soundness proofs for extensional hnf bisimilarities are uniformly done using a nested induction proof method [20, 23]. The proof can then be turned into a soundness proof for bisimulation up to context.
- The soundness proofs of extensional enf bisimilarities either follow from a CPS translation [21, 23], or other ad hoc arguments [34, 24, 25, 9] which do not carry over to a soundness proof for a bisimulation up to context.

---

[2]While weak head normal forms are normal forms under call-by-name evaluation, eager normal forms are normal forms under call-by-value evaluation of λ-terms.

- The only claims about congruence of an extensional enf bisimilarity as well as soundness of the corresponding bisimulation up to context using a nested induction proof are either wrong [8] or are not substantiated by a presentation of the actual proof [21]. The reason the nested induction proof works for extensional hnf bisimilarities and not for extensional enf bisimilarities stems from the difference in the requirements on the shape of λ-abstractions the two normal forms impose: whereas the body of a λ-abstraction in hnf is also a hnf, the body of a λ-abstraction in enf is an arbitrary term.

In this paper, we consider an extensional enf bisimilarity for three calculi: the plain λ-calculus and its extensions with delimited and abortive continuations, and in each case we present a soundness proof of the corresponding enf bisimulation up to context from which congruence of the bisimilarity follows.

3. Call-by-value λ-calculus

We introduce a new approach to normal-form bisimulations that is based on the framework we developed previously [3]. The calculus of discourse is the plain call-by-value λ-calculus.

3.1. **Syntax, semantics, and normal-form bisimulations.**

We let x, y, z range over variables. The syntax of terms (t, s), values (v, w), and call-by-value evaluation contexts (E) is given as follows:

$$
\begin{array}{rcl}
t, s & ::= & v \mid t\,s \\
v, w & ::= & x \mid \lambda x.t \\
E & ::= & \Box \mid E\,t \mid v\,E
\end{array}
$$

An abstraction $\lambda x.t$ binds x in t; a variable that is not bound is called free. The set of free variables in a term t is written $\mathsf{fv}(t)$. We work modulo α-conversion of bound variables, and a variable is called fresh if it does not occur in the terms under consideration. Contexts are represented outside-in, and we write $E[t]$ for plugging a term in a context. We write $t\{v/x\}$ for the capture-avoiding substitution of v for x in t. We write successive λ-abstractions $\lambda x.\lambda y.t$ as $\lambda xy.t$.

We consider a call-by-value reduction semantics for the language

$$
E[(\lambda x.t)\,v] \rightarrow E[t\{v/x\}]
$$

We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$, and $t \Downarrow s$ if $t \rightarrow^* s$ and s cannot reduce; we say that t evaluates to s.

Eager normal forms are either values or *open stuck terms* of the form $E[x\,v]$. Normal-form bisimilarity relates terms by comparing their normal forms (if they exist). For values, a first possibility is to relate separately variables and λ-abstractions: a variable x can be equated only to x, and $\lambda x.t$ is bisimilar to $\lambda x.s$ if t is bisimilar to s. As explained in the introduction, this does not respect η-expansion: the η-respecting definition compares values by applying them to a fresh variable. Given a relation $\mathcal{R}$ on terms, we reflect how values and open stuck terms are tested by the relations $\mathcal{R}^{\mathsf{v}}$, $\mathcal{R}^{\mathsf{ctx}}$, and $\mathcal{R}^{\mathsf{o}}$, defined as follows:

$$
\frac{v\,x\ \mathcal{R}\ w\,x \qquad x \text{ fresh}}{v\ \mathcal{R}^{\mathsf{v}}\ w} \qquad\qquad \frac{E[x]\ \mathcal{R}\ E'[x] \qquad x \text{ fresh}}{E\ \mathcal{R}^{\mathsf{ctx}}\ E'} \qquad\qquad \frac{E\ \mathcal{R}^{\mathsf{ctx}}\ E' \qquad v\ \mathcal{R}^{\mathsf{v}}\ w}{E[x\,v]\ \mathcal{R}^{\mathsf{o}}\ E'[x\,w]}
$$

Remark 3.1. Traditionally, normal-form bisimulations are construed as an open version of applicative bisimulations in that they test values by applying them to a free variable [21], rather than to all possible closed values [1]. However, a connection with Böhm or Lévy-Longo trees [20] aside, one could introduce a separate category of variables that would represent *abstract values*, and use these for the purpose of testing functional values. In such an approach, the reduction relation would cater for closed terms only, as far as the term variables are concerned, and the notion of an open stuck term could be replaced with a notion of a *value-stuck term*. In this work we stick to the traditional approach to testing functional values (witness the definition of R^{v}), but in Section 5 we propose an extension which is analogous to the one sketched in this remark, and we introduce a separate category of variables representing *abstract contexts*, a notion dual to that of abstract values.

We can now define (extensional) normal-form bisimulation and bisimilarity, using a notion of progress.

Definition 3.2. A relation $\mathcal{R}$ progresses to $\mathcal{S}$ if $t \mathrel{\mathcal{R}} s$ implies:
- if $t \to t'$, then there exists s' such that $s \to^* s'$ and $t' \mathrel{\mathcal{S}} s'$;
- if $t = v$, then there exists w such that $s \Downarrow w$, and $v \mathrel{\mathcal{S}^{\mathsf{v}}} w$;
- if $t = E[x\,v]$, then there exist E', w such that $s \Downarrow E'[x\,w]$ and $E[x\,v] \mathrel{\mathcal{S}^{\circ}} E'[x\,w]$;
- the converse of the above conditions on s.

A bisimulation is then defined as a relation which progresses to itself, and bisimilarity as the union of all bisimulations. Our definition is in a small-step style, unlike Lassen's [21], as we believe small-step is more flexible, since we can recover a big-step reasoning with up to reduction (Section 3.3). In usual definitions [21, 34, 9], the β-reduction is directly performed when a λ-abstraction is applied to a fresh variable, whereas we construct an application in order to uniformly treat all kinds of values, and hence account for η-expansion. However, with this approach a naive reasoning up to context would be unsound because it would equate any two values: if v and w are related, then $v\,x$ and $w\,x$ are related up to context. In our framework, we prevent this issue by not allowing this up-to technique in that case, as we now explain.

We recast the definition of normal-form bisimilarity in the framework of our previous work [3], which is itself an extension of a work by Madiot et al. [26, 27]. The goal is to factorize the congruence proof of the bisimilarity with the soundness proofs of the up-to techniques. The novelty in [3] is that we distinguish between *active* and *passive* clauses, and we forbid some up-to techniques to be applied in a passive clause. Whereas this distinction does not change the notions of bisimulation or bisimilarity, it has an impact on the bisimilarity congruence proof.

Definition 3.3. A relation $\mathcal{R}$ *diacritically progresses* to $\mathcal{S}, \mathcal{T}$ written $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{S} \subseteq \mathcal{T}$, and $t \mathrel{\mathcal{R}} s$ implies:
- if $t \to t'$, then there exists s' such that $s \to^* s'$ and $t' \mathrel{\mathcal{T}} s'$;
- if $t = v$, then there exists w such that $s \Downarrow w$, and $v \mathrel{\mathcal{S}^{\mathsf{v}}} w$;
- if $t = E[x\,v]$, then there exist E', w such that $s \Downarrow E'[x\,w]$ and $E[x\,v] \mathrel{\mathcal{T}^{\circ}} E'[x\,w]$;
- the converse of the above conditions on s.

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

The difference between Definitions 3.3 and 3.2 is only in the clause for values, where we progress towards a different relation than in the other clauses of Definition 3.3. We say that the clause for values is passive, while the others are active. A bisimulation $\mathcal{R}$ progresses towards $\mathcal{R}$ in passive and active clauses, so the two notions of bisimulation coincide: $\mathcal{R}$ is a bisimulation according to Definition 3.3 iff it is a bisimulation by Definition 3.2. Consequently, the resulting bisimilarity is also the same between the two definitions.

Example 3.4. Let $\theta \stackrel{\mathsf{def}}{=} \lambda z x.x \, \lambda y.z \, z \, x \, y$ and $\mathsf{fix}(v) \stackrel{\mathsf{def}}{=} \lambda x.\theta \, \theta \, v \, x$ for a given v; note that $\mathsf{fix}(v) \, x \to^* v \, \mathsf{fix}(v) \, x$. Wadsworth's infinite η-expansion combinator [5] can be defined as $\mathsf{J} \stackrel{\mathsf{def}}{=} \mathsf{fix}(\lambda f x y.x \, (f \, y))$. Let $\mathcal{I} \stackrel{\mathsf{def}}{=} \{(t,t) \mid t \text{ any term}\}$ be the identity bisimulation. We prove that $\lambda x.x \approx \mathsf{J}$, by showing that

$$\mathcal{R} \stackrel{\mathsf{def}}{=} \mathcal{I} \cup \{(\lambda x.x, \mathsf{J})\} \cup \{(t,s) \mid (\lambda x.x) \, y \to^* t, \mathsf{J} \, y \to^* s \mid y \text{ fresh}\}$$
$$\cup \{(y \, z, t) \mid (\lambda x.y \, (\mathsf{J} \, x)) \, z \to^* t \mid y, z \text{ fresh}\}$$

is a bisimulation. Indeed, to compare $\lambda x.x$ and J, we have to relate $(\lambda x.x) \, y$ and $\mathsf{J} \, y$, but $\mathsf{J} \, y \to^* \lambda x.y \, (\mathsf{J} \, x)$. We then have to equate $y \, z$ and $(\lambda x.y \, (\mathsf{J} \, x)) \, z$, the latter evaluating to $y \, \lambda x.z \, (\mathsf{J} \, x)$. To relate these open stuck terms, we have to equate $\square$ and $\square$ (with $\mathcal{I}$), and z with $\lambda x.z \, (\mathsf{J} \, x)$, but these terms are already in $\mathcal{R}$.

The purpose of the distinction between active and passive is not to change regular bisimulation proofs, but instead to affect how up-to techniques are applied, by forbidding the use of some of them in a passive clause. In particular, reasoning up to context is not allowed in the value case, meaning that we cannot use the fact that $v \, x$ and $w \, x$ are related up to context whenever v and w are related to conclude a proof with bisimulation up to context. In contrast, it is safe to use any (combination of) up-to techniques in an active clause. In the next section, we explain how to prove that a function on relations is an up-to technique and whether it can be safely used in a passive clause or not.

3.2. **Up-to techniques, general definitions.** We recall here the definitions from our previous work [3]. The goal of up-to techniques is to simplify bisimulation proofs: instead of proving that a relation $\mathcal{R}$ is a bisimulation, we show that $\mathcal{R}$ respects some looser constraints which still imply bisimilarity. In our setting, we distinguish the up-to techniques which can be used in passive clauses (called *strong* up-to techniques), from the ones which cannot.

Definition 3.5. An up-to technique (resp. strong up-to technique) is a function f on relations such that $\mathcal{R} \rightarrowtail \mathcal{R}, f(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail f(\mathcal{R}), f(\mathcal{R})$) implies $\mathcal{R} \subseteq \approx$. When f is an up-to technique (resp. strong up-to technique) and $\mathcal{R} \rightarrowtail \mathcal{R}, f(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail f(\mathcal{R}), f(\mathcal{R})$), we say that $\mathcal{R}$ is a bisimulation up to f.

Example 3.6. In Example 3.4, we have to include identical terms (related by $\mathcal{I}$) in the definition of the candidate relation, although we already know such terms are bisimilar. To avoid doing so, we define the up to reflexivity technique refl as follows.

$$\overline{t \, \mathsf{refl}(\mathcal{R}) \, t}$$

In our setting, we show that refl is a strong up-to technique and can be used after active and passive clause. We can then define $\mathcal{R}$ in Example 3.4 without $\mathcal{I}$ and show it is a bisimulation up to reflexivity.

Example 3.7. In deterministic languages, a common up-to technique is to reason *up to reduction* [30], which allows terms to reduce before being related.

$$\frac{t \to^* t' \qquad s \to^* s' \qquad t' \, \mathcal{R} \, s'}{t \, \mathsf{red}(\mathcal{R}) \, s}$$

With this technique, one can ignore the intermediary reduction steps and reason in a big-step way even with a small-step semantics. We prove in this section that red is another example of strong up-to technique. In contrast, up to context, discussed in Section 3.3 is not strong and cannot be used after passive transitions.

Proving that a given f is an up-to technique is difficult with Definition 3.5, in part because this notion is not stable under composition or union [27]. Following Madiot, Pous, and Sangiorgi [30, 26], we rely on a notion of *compatibility*, which gives sufficient conditions for f to be an up-to technique, and is easier to establish, as functions built out of compatible functions using composition and union remain compatible.

We first need some auxiliary notions on notations on functions on relations, ranged over by f, g, and h in what follows. We define $f \subseteq g$ and $f \cup g$ argument-wise, e.g., $(f \cup g)(\mathcal{R}) = f(\mathcal{R}) \cup g(\mathcal{R})$ for all $\mathcal{R}$. We define f^ω as $\bigcup_{n \in \mathbb{N}} f^n$. We write id for the identity function on relations, and $\widehat{f}$ for $f \cup \mathsf{id}$. The technique refl of Example 3.6 differs from id as the former builds the identify on terms (for all $\mathcal{R}$ and t, $t \, \mathsf{refl}(\mathcal{R}) \, t$), while the latter is the identity on relations (for all $\mathcal{R}$, t, and s, $t \, \mathcal{R} \, s$ implies $t \, \mathsf{id}(\mathcal{R}) \, s$).

Given a set $\mathfrak{F}$ of functions, we also write $\mathfrak{F}$ for the function defined as $\bigcup_{f \in \mathfrak{F}} f$. We say a function f is *generated from* $\mathfrak{F}$ if f can be built from functions in $\mathfrak{F}$ and id using union, composition, and $\cdot^\omega$. The largest function generated from $\mathfrak{F}$ is $\widehat{\mathfrak{F}}^\omega$.

A function f is *monotone* if $\mathcal{R} \subseteq \mathcal{S}$ implies $f(\mathcal{R}) \subseteq f(\mathcal{S})$. We write $\mathcal{P}_{fin}(\mathcal{R})$ for the set of finite subsets of $\mathcal{R}$, and we say f is *continuous* if it can be defined by its image on these finite subsets, i.e., if $f(\mathcal{R}) \subseteq \bigcup_{\mathcal{S} \in \mathcal{P}_{fin}(\mathcal{R})} f(\mathcal{S})$. The up-to techniques of the present paper are defined by inference rules with a finite number of premises, like red or refl, so we can easily show they are continuous. Continuous functions are interesting because of their properties:[3]

Lemma 3.8. *If f and g are continuous, then $f \circ g$ and $f \cup g$ are continuous. If f is continuous, then f is monotone, and $f \circ \widehat{f}^\omega \subseteq \widehat{f}^\omega$.*

Compatibility relies on a notion of *evolution* for functions on relations, which can be seen has the higher-order equivalent of progress.

Definition 3.9. A function f evolves to g, h, written $f \rightsquigarrow g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\twoheadrightarrow g(\mathcal{R}), h(\mathcal{T})$. A function f *strongly* evolves to g, h, written $f \rightsquigarrow_\mathsf{s} g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\twoheadrightarrow g(\mathcal{S}), h(\mathcal{T})$.

We have $f \rightsquigarrow g, h$ or $f \rightsquigarrow_\mathsf{s} g, h$ if terms related by $f(\mathcal{R})$ become related by respectively $g(\mathcal{R})$ and $h(\mathcal{T})$, or $g(\mathcal{S})$ and $h(\mathcal{T})$, depending on how $\mathcal{R}$ progresses. Like with progress, evolution distinguishes passive from active clauses, so that g is used after passive clauses and h after active ones. We further distinguish between regular evolution $\rightsquigarrow$ and strong evolution $\rightsquigarrow_\mathsf{s}$: for regular evolution, we consider $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$, while strong evolution is more liberal, as it allows for relations $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$. It follows the discrepancy between

---

[3]Our formalization revealed an error in previous works [3, 27] which use f instead of $\widehat{f}$ in the last property of Lemma 3.8 (expressing idempotence of $\widehat{f}^\omega$)—id has to be factored in for the property to hold.

regular and strong up-to techniques, the former being not allowed after passive clauses, while the latter are.

The next examples illustrate strong evolution; for an example of regular evolution, see Lemma 3.20.

Example 3.10. It is easy to prove that $\mathsf{refl} \leadsto_{\mathsf{s}} \mathsf{refl}, \mathsf{refl}$, as identical terms remain equal when they progress.

Example 3.11. We show that $\mathsf{red} \leadsto_{\mathsf{s}} \mathsf{red} \cup \mathsf{id}, \mathsf{red} \cup \mathsf{id}$. Let $\mathcal{R} \rightarrowtail\!\!\!\rightarrow \mathcal{S}, \mathcal{T}$. We first have to show the inclusions $\mathsf{red}(\mathcal{R}) \subseteq \mathsf{red}(\mathcal{S}) \cup \mathcal{S}$ and $\mathsf{red}(\mathcal{S}) \cup \mathcal{S} \subseteq \mathsf{red}(\mathcal{T}) \cup \mathcal{T}$; these hold because $\mathcal{R} \subseteq \mathcal{S}, \mathcal{S} \subseteq \mathcal{T}$ (by definition of $\rightarrowtail\!\!\!\rightarrow$) and red is continuous and therefore monotone.

Next, let $t \, \mathsf{red}(\mathcal{R}) \, s$; by definition, there exist t' and s' such that $t \rightarrow^* t'$, $s \rightarrow^* s'$, and $t' \, \mathcal{R} \, s'$. We consider in turn the different progress clauses of Definition 3.3.

For the reduction clause, let t'' such that $t \rightarrow t''$; because this clause is active, we want to find s'' such that $s \rightarrow^* s''$ and $t'' \, (\mathsf{red} \cup \mathsf{id})(\mathcal{T}) \, s''$. We distinguish two cases: first, suppose $t \neq t'$. Because the semantics is deterministic, we have $t'' \rightarrow^* t'$, and therefore $t'' \, \mathsf{red}(\mathcal{R}) \, s$ holds. But $\mathcal{R} \subseteq \mathcal{T}$ by definition of progress, and red is monotone, hence we have $t'' \, \mathsf{red}(\mathcal{T}) \, s$, which implies $t'' \, (\mathsf{red} \cup \mathsf{id})(\mathcal{T}) \, s$, as wished.

Suppose now that $t = t'$; then $t \, \mathcal{R} \, s'$. Because $\mathcal{R} \rightarrowtail\!\!\!\rightarrow \mathcal{S}, \mathcal{T}$ and $t \rightarrow t''$, there exists s'' such that $s' \rightarrow^* s''$ and $t'' \, \mathcal{T} \, s''$. Therefore, $s \rightarrow^* s''$, and $t'' \, \mathcal{T} \, s''$ implies $t'' \, (\mathsf{red} \cup \mathsf{id})(\mathcal{T}) \, s''$, as required.

For the passive value clause, we have $t = v$, and we want to find w such that $s \rightarrow^* w$ and $v \, \mathcal{S}^{\mathsf{v}} \, w$. Because t cannot reduce, we have $t' = v$, and therefore $v \, \mathcal{R} \, s'$. Because $\mathcal{R} \rightarrowtail\!\!\!\rightarrow \mathcal{S}, \mathcal{T}$, there exists w such that $s' \rightarrow^* w$ and $v \, \mathcal{S}^{\mathsf{v}} \, w$. But we also have $s \rightarrow^* w$, hence the result holds. The reasoning for the active open stuck term clause is the same, but with $\mathcal{T}$ instead of $\mathcal{S}$.

Like in Madiot et al. [26], we say that a set of functions $\mathfrak{F}$ is compatible if each function f in $\mathfrak{F}$ evolves towards functions generated from $\mathfrak{F}$. However, in contrast with Madiot et al., we need some restrictions on the resulting combinations, which depend on whether f is a strong up-to technique or not.

Definition 3.12. A set $\mathfrak{F}$ of continuous functions is *diacritically compatible* if $\mathfrak{F}$ has a subset $\mathfrak{S}$ such that
- for all $f \in \mathfrak{S}$, we have $f \leadsto_{\mathsf{s}} \widehat{\mathfrak{S}}^\omega, \widehat{\mathfrak{F}}^\omega$;
- for all $f \in \mathfrak{F}$, we have $f \leadsto \widehat{\mathfrak{S}}^\omega \circ \widehat{\mathfrak{F}} \circ \widehat{\mathfrak{S}}^\omega, \widehat{\mathfrak{F}}^\omega$.

The (possibly empty) subset $\mathfrak{S}$ represents the strong up-to techniques of $\mathfrak{F}$, and a function f may evolve differently depending on whether f belongs to $\mathfrak{S}$ or not. In both cases, f may evolve towards any function generated from $\mathfrak{F}$ after an active clause. The difference is after passive clauses, where a function in $\mathfrak{S}$ has to evolve towards a function generated from $\mathfrak{S}$ only, while composing with a non-strong function is allowed if $f \notin \mathfrak{S}$. In the latter case, we progress from $f(\mathcal{R})$ with f not strong, and we expect to progress towards a combination which still includes f. It is safe to do so, as long as f (or in fact, any non-strong function in $\mathfrak{F}$) is used at most once.

If $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are subsets of $\mathfrak{F}$ which verify the conditions of Definition 3.12, then $\mathfrak{S}_1 \cup \mathfrak{S}_2$ also does, so there exists the largest subset of $\mathfrak{F}$ which satisfies the conditions, written $\mathsf{strong}(\mathfrak{F})$. The next lemma shows that being in a compatible set is a sufficient criterion to be a (possibly strong) up-to technique. In practice, proving that f is in a compatible set $\mathfrak{F}$ is easier than proving directly it is an up-to technique.

Lemma 3.13. *Let $\mathfrak{F}$ be a diacritically compatible set.*

- *If $\mathcal{R} \rightarrowtail \widehat{\mathsf{strong}(\mathfrak{F})}^{\omega}(\mathcal{R}), \widehat{\mathfrak{F}}^{\omega}(\mathcal{R})$, then $\widehat{\mathfrak{F}}^{\omega}(\mathcal{R})$ is a bisimulation.*
- *any function generated from $\mathfrak{F}$ is an up-to technique, and any function generated from $\mathsf{strong}(\mathfrak{F})$ is a strong up-to technique.*
- *For all $f \in \mathfrak{F}$, we have $f(\approx) \subseteq \approx$.*

The second point implies that combining functions from a compatible set using union, composition, or $\cdot^{\omega}$ produces up-to techniques. In particular, if $f \in \mathfrak{F}$, then f is an up-to technique, and similarly, if $f \in \mathsf{strong}(\mathfrak{F})$, then f is a strong up-to technique. The last item states that bisimilarity respects compatible functions, so proving that up to context is compatible implies that bisimilarity is preserved by contexts.

The first item suggests a more flexible notion of up-to technique, as it shows that given a compatible set $\mathfrak{F}$, a relation may progress towards different functions f and g, $\mathcal{R} \rightarrowtail f(\mathcal{R}), g(\mathcal{R})$, and still be included in the bisimilarity as long as f is generated from $\mathsf{strong}(\mathfrak{F})$ and g is generated from $\mathfrak{F}$. In what follows, we rely on that property in examples and say that in that case, $\mathcal{R}$ is a bisimulation up to $\mathfrak{F}$, or $\mathcal{R}$ is a bisimulation up to $f_1 \ldots f_n$ if $\mathfrak{F} = \{f_1 \ldots f_n\}$.

Example 3.14. The set $\mathfrak{F} \stackrel{\mathsf{def}}{=} \{\mathsf{red}\}$ is diacritically compatible, with $\mathsf{strong}(\mathfrak{F}) = \{\mathsf{red}\}$. Indeed, we prove in Example 3.11 that red strongly progresses towards combinations of red and id which respects the conditions of Definition 3.12. As a consequence, red is a strong up-to technique by Lemma 3.13, and the bisimilarity is preserved by reduction: if $t \rightarrow^* t'$, $s \rightarrow^* s'$, and $t' \approx s'$, then $t \approx s$.

Similarly, $\{\mathsf{refl}\}$ is diacritically compatible and refl is strong, and $\mathsf{refl}(\approx) \subseteq \approx$ implies that two equal terms are bisimilar.

Example 3.15. We can simplify the definition of $\mathcal{R}$ in Example 3.4 to just

$$\mathcal{R} \stackrel{\mathsf{def}}{=} \{(\lambda x.x, \mathsf{J}), (y, \lambda x.y\,(\mathsf{J}\,x)) \mid y \text{ fresh}\}$$

and show that $\mathcal{R}$ is a bisimulation up to refl and red.

3.3. **Up to context for normal-form bisimilarity.** Our primary goal in this section is to prove that reasoning up to context is an up-to technique. We let C range over contexts, i.e., terms with a hole $\square$, and define reasoning up to context as follows.

$$\frac{t \; \mathcal{R} \; s}{C[t] \; \mathsf{ctx}(\mathcal{R}) \; C[s]}$$

Unlike red or refl in the previous section, we cannot prove up to context is an up-to technique by itself. We need some extra techniques, but we also decompose ctx into smaller techniques, to allow for a finer-grained distinction between strong and regular up-to techniques.

Figure 1 presents the techniques we consider for the λ-calculus. The substitutive closure subst is already used in previous works [20, 23, 9]. The closure by evaluation contexts ectx is more unconventional, although we define it in a previous work [9]. It is not the same as reasoning up to context, since we can factor out different contexts, as long as they are related when we plug a fresh variable inside them. It is reminiscent of $\star$-bisimilarity [3] which can also factor out different contexts in its up-to techniques, except that $\star$-bisimilarity compares contexts with values and not simply variables.
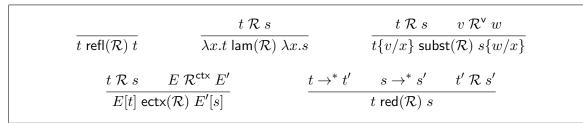
$$\frac{}{t \; \mathsf{refl}(\mathcal{R}) \; t} \qquad \frac{t \; \mathcal{R} \; s}{\lambda x.t \; \mathsf{lam}(\mathcal{R}) \; \lambda x.s} \qquad \frac{t \; \mathcal{R} \; s \quad v \; \mathcal{R}^{\mathsf{v}} \; w}{t\{v/x\} \; \mathsf{subst}(\mathcal{R}) \; s\{w/x\}}$$

$$\frac{t \; \mathcal{R} \; s \quad E \; \mathcal{R}^{\mathsf{ctx}} \; E'}{E[t] \; \mathsf{ectx}(\mathcal{R}) \; E'[s]} \qquad \frac{t \to^* t' \quad s \to^* s' \quad t' \; \mathcal{R} \; s'}{t \; \mathsf{red}(\mathcal{R}) \; s}$$

Figure 1: Up-to techniques for the λ-calculus

We first show that we can indeed build ctx out of the techniques of Figure 1. Closure w.r.t. λ-abstraction is achieved through lam, and closure w.r.t. variables is a consequence of refl, as we have $x \; \mathsf{refl}(\mathcal{R}) \; x$ for all x. We can construct an application out of ectx and refl.

Lemma 3.16. *If $t \; \mathcal{R} \; t'$ and $s \; \mathcal{R} \; s'$, then $t \, s \; (\mathsf{ectx} \circ (\mathsf{id} \cup \mathsf{ectx} \circ (\mathsf{id} \cup \mathsf{refl})))(\mathcal{R}) \; t' \, s'$.*

Let x be a fresh variable; then $x \; \square \; \mathsf{refl}(\mathcal{R})^{\mathsf{ctx}} \; x \; \square$. Combined with $s \; \mathcal{R} \; s'$, it implies $x \, s \; \mathsf{ectx}((\mathsf{id} \cup \mathsf{refl})(\mathcal{R})) \; x \, s'$, i.e., $\square \, s \; \mathsf{ectx}((\mathsf{id} \cup \mathsf{refl})(\mathcal{R}))^{\mathsf{ctx}} \; \square \, s'$. This combined with $t \; \mathcal{R} \; t'$ using ectx gives the result of Lemma 3.16. In the end, if we define app as the combination of techniques of Lemma 3.16, then the following holds.

Lemma 3.17. $t \; \mathsf{ctx}(\mathcal{R}) \; s$ *iff* $t \; (\mathsf{refl} \cup \mathsf{lam} \cup \mathsf{app})^{\omega} \; s$.

Proving that $\mathfrak{F} \overset{\mathsf{def}}{=} \{\mathsf{refl}, \mathsf{lam}, \mathsf{subst}, \mathsf{ectx}, \mathsf{id}, \mathsf{red}\}$ is diacritically compatible amounts to showing that each function in $\mathfrak{F}$ evolves towards functions generated from $\mathfrak{F}$ which respect Definition 3.12. Before discussing some of the compatibility proofs, we compare ourselves to Lassen's proof [20]. Lassen defines a closure combining all the techniques of Figure 1, and then proves by induction on its definition that it is a bisimulation, using Definition 3.2.

A first difference is that our proofs are in a small-step style while Lassen's is big-step, which means that he has to perform an extra induction on the number of reduction steps. Our technique is also more modular, as we can isolate smaller techniques and do each compatibility proof separately, instead of reasoning on a single closure. Apart from these (minor) points, our proof and Lassen's are quite similar, as they consist in case analyses on the possible reductions the related terms can make. Therefore, the main difference between our setting and Lassen's is not so much the proof technique itself, but the notion of progress it is based upon. The crucial case is when proving Lemma 3.20, discussed below, where we can conclude thanks to diacritical progress, while we do not know how to complete the proof with a regular notion of progress (Definition 3.2).

We now sketch some evolution proofs, starting with the simplest ones. We already discussed the proofs for red and refl in Section 3.2. The strong evolution proof for lam is also quite simple.

Lemma 3.18. $\mathsf{lam} \rightsquigarrow_{\mathsf{s}} \mathsf{lam} \cup \mathsf{red}, \mathsf{lam} \cup \mathsf{red}$.

Sketch. Let $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$; we want to prove that $\mathsf{lam}(\mathcal{R}) \rightarrowtail \mathsf{lam}(\mathcal{S}) \cup \mathsf{red}(\mathcal{S}), \mathsf{lam}(\mathcal{T}) \cup \mathsf{red}(\mathcal{T})$. Let $\lambda x.t \; \mathsf{lam}(\mathcal{R}) \; \lambda x.s$ such that $t \; \mathcal{R} \; s$. The only clause to check is the one for values: we have $(\lambda x.t) \, x \to t$ and $(\lambda x.s) \, x \to s$, i.e., $(\lambda x.t) \, x \; \mathsf{red}(\mathcal{R}) \; (\lambda x.s) \, x$, which implies $(\lambda x.t) \, x \; \mathsf{red}(\mathcal{S}) \; (\lambda x.s) \, x$ because $\mathcal{R} \subseteq \mathcal{S}$ and red is monotone. $\square$

The technique subst is also strong, as we can show the following result.

Lemma 3.19. $\mathsf{subst} \rightsquigarrow_{\mathsf{s}} \mathsf{subst}, (\mathsf{id} \cup \mathsf{ectx}) \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})$.

Sketch. Let $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$, and $t\{v/x\}$ subst$(\mathcal{R})$ $s\{w/x\}$ such that $t\ \mathcal{R}\ s$ and $v\ \mathcal{R}^{\vee}\ w$. We check the different clauses by case analysis on t. If t is a value, then there exists a value s' such that $s \Downarrow s'$ and $t\ \mathcal{R}^{\vee}\ s'$. But then $t\{v/x\}$ and $s'\{w/x\}$ are also values, and then we can prove that $t\{v/x\}$ subst$(\mathcal{S})^{\vee}$ $s'\{w/x\}$ holds. If $t \to t'$, then there exists s' such that $s \to^{*} s'$ and $t'\ \mathcal{T}\ s'$. Then $t\{v/x\} \to t'\{v/x\}$, $s\{w/x\} \to^{*} s'\{w/x\}$, and $t'\{v/x\}$ subst$(\mathcal{T})$ $s'\{w/x\}$.

Finally, if $t = E[y\ v']$, then there exists s' such that $s \Downarrow s'$ and $t\ \mathcal{T}^{\circ}\ s'$. If $y \neq x$, then $t\{v/x\}$ and $s'\{w/x\}$ are open stuck terms in subst$(\mathcal{T})^{\circ}$. Otherwise, we distinguish cases based on whether v is a λ-abstraction or not. In the former case, let $v = \lambda z.t'$, $s' = E'[x\ w']$. Then $t\{v/x\} = E\{v/x\}[v\ v'\{v/x\}] \to E\{v/x\}[t'\{v'\{v/x\}/z\}]$. From $v\ \mathcal{R}^{\vee}\ w$ and $v\ z \to t'$, we know that there exists s'' such that $w\ z \to^{*} s''$ and $t'\ \mathcal{T}\ s''$. Consequently, we have $s\{w/x\} \to^{*} s'\{w/x\} = E'\{w/x\}[w\ w'\{w/x\}] \to^{*} E'\{w/x\}[s''\{w'\{w/x\}/z\}]$. Then $E\{v/x\}[x']$ subst$(\mathcal{T})$ $E'\{w/x\}[x']$ for a fresh x', but also $t'\{v'\{v/x\}/z\}$ subst(subst$(\mathcal{T})$) $s''\{w'\{w/x\}/z\}$, so after plugging, we obtain terms in ectx $\circ$ subst $\circ$(id $\cup$ subst)$(\mathcal{T})$.

If v is a variable, a similar reasoning shows that $t\{v/x\}$ and $s'\{w/x\}$ evaluate to open stuck terms, whose contexts are related by ectx $\circ$ subst $\circ$(id $\cup$ subst)$(\mathcal{T})$ and whose arguments are related by subst(subst$(\mathcal{T})$). $\qquad\square$

The proof for subst does not require the clause for values to be passive, and is thus similar to the corresponding subcase in Lassen's proof by induction [20]. In contrast, we need testing values to be passive when dealing with ectx; we present the problematic subcase below. This is where our proof technique differs from Lassen's, as we do not know how to make this subcase go through in a proof by induction.

Lemma 3.20. ectx $\rightsquigarrow$ ectx, subst $\cup$ ectx $\cup$ (id $\cup$ ectx) $\circ$ subst $\circ$(id $\cup$ subst).

Sketch. Let $\mathcal{R} \rightarrowtail \mathcal{R}, \mathcal{S}$, and $E[t]$ ectx$(\mathcal{R})$ $E'[s]$ such that $E\ \mathcal{R}^{\mathsf{ctx}}\ E'$ and $t\ \mathcal{R}\ s$. We proceed by case analysis on E and t. Most cases are straightforward; the problematic case is when t is a variable x and $E = \square\ w$. Because $t\ \mathcal{R}\ s$, there exists v_2 such that $s \Downarrow v_2$ and $x\ \mathcal{R}^{\vee}\ v_2$. Because $E\ \mathcal{R}^{\mathsf{ctx}}\ E'$, we have $E[y]\ \mathcal{R}\ E'[y]$ for a fresh y, and therefore $E[x]$ subst$(\mathcal{R})$ $E'[v_2]$. We can conclude using Lemma 3.19: there exists an open stuck term s' such that $E'[s] \to^{*} E'[v_2] \to^{*} s'$ and $x\ w$ ((id $\cup$ ectx) $\circ$ subst $\circ$(id $\cup$ subst)$(\mathcal{S}))^{\circ}\ s'$.

In an induction proof with Definition 3.2, we would have in that case $x\ \mathcal{S}^{\vee}\ v_2$ instead of $\mathcal{R}$, and $E'[y] \Downarrow s'$ for some s' such that $y\ w\ \mathcal{S}^{\circ}\ s'$. We do not see how to go further in the case v_2 is a λ-abstraction: we have to prove that $s'\{v_2/y\}$ evaluates to an open stuck term, but we do not have any progress hypothesis about $\mathcal{S}$. $\qquad\square$

Each technique of $\mathfrak{F}$ evolves towards functions generated from $\mathfrak{F}$ respecting Definition 3.12: for passive clauses, red, refl, lam, and subst strongly evolves towards combinations of strong techniques, while ectx evolves towards itself, thus respecting the criterion of at most one not strong technique in that case. As a result, the following theorem holds.

Theorem 3.21. *The set $\mathfrak{F}$ is diacritically compatible, with* strong$(\mathfrak{F}) = \mathfrak{F} \setminus \{\mathsf{ectx}\}$.

A direct consequence of Lemma 3.13 is that any function generated from $\mathfrak{F}$ is an up-to technique, so in particular ctx (cf Lemma 3.17). Lemma 3.13 then also implies that $\approx$ is preserved by contexts. As it is easy to show that $\approx$ is also an equivalence relation, we have the following corollary.

Corollary 3.22. $\approx$ *is a congruence.*

This corollary, in turn, immediately implies the soundness of $\approx$ w.r.t. the usual contextual equivalence of the λ-calculus, where we observe termination of evaluation [1]—the notion of

contextual equivalence that we take throughout the paper. However, as proved in [21], $\approx$ is not complete w.r.t. contextual equivalence.

Example 3.23. With the same definitions as in Examples 3.4, let

$$v \stackrel{\mathsf{def}}{=} \mathsf{fix}(\lambda zxy.z\,x) \text{ and } w \stackrel{\mathsf{def}}{=} \mathsf{fix}(\lambda zxy.z\,(\mathsf{J}\,x)).$$

We prove these values are bisimilar by showing that

$$\mathcal{R} \stackrel{\mathsf{def}}{=} \{(v,w),(v\,x,w\,x),(x,\lambda y.x\,(\mathsf{J}\,y)) \mid x \text{ fresh}\}$$

is a bisimulation up to $\mathfrak{F}$. For the first pair, we have directly $v\,\mathcal{R}^{\mathsf{v}}\,w$. For the second pair, we have

$$v\,x \to^* (\lambda zxy.z\,x)\,v\,x \to^* \lambda y.v\,x \text{ and } w\,x \to^* (\lambda zxy.z\,(\mathsf{J}\,x))\,w\,x \to^* \lambda y.w\,(\mathsf{J}\,x).$$

We show the two resulting values are related up to $\mathfrak{F}$. Indeed, we have $v\,\mathcal{R}\,w$, and $x\,\mathsf{red}(\mathcal{R})\,\mathsf{J}x$, because $\mathsf{J}\,x \to^* \lambda y.x\,(\mathsf{J}\,y)$, therefore $\lambda y.v\,x\,\mathsf{lam}(\mathsf{app}((\mathsf{id} \cup \mathsf{red})(\mathcal{R})))\,\lambda y.w\,(\mathsf{J}\,x)$. Note that we can use app—which is built out of the not strong technique ectx (cf. Lemma 3.16)—in that case, as the reduction clause is active.

For the last pair, we compare $x\,z$ and $\lambda y.x\,(\mathsf{J}\,y)\,z$, but $\lambda y.x\,(\mathsf{J}\,y)\,z \to^* x\,\lambda y.z\,(\mathsf{J}\,y)$, so we can conclude with up to reduction and reflexivity.

4. Delimited-control operators

We show that the results of Section 3 seamlessly carry over to the call-by-value λ-calculus extended with `shift` and `reset` [12, 9], thus demonstrating the robustness of the approach, but also improving on the previous results on extensional normal-form bisimulations for this calculus [9].

4.1. **Syntax, semantics, and normal-form bisimulations.** We extend the grammar of terms and values given in Section 3 as follows:

$$\begin{aligned} t,s &\quad ::= \quad \ldots \mid \langle t \rangle \\ v,w &\quad ::= \quad \ldots \mid \mathsf{S} \end{aligned}$$

where $\langle \cdot \rangle$ is the control delimiter `reset` and S is the delimited-control operator `shift`. Usually, `shift` is presented as a binder $\mathsf{S}x.t$ [12, 9] or as a special form $\mathsf{S}\,t$ [17], but here we choose a more liberal syntax treating `shift` as a value (as, e.g., in [18]). This makes the calculus a little more interesting since `shift` becomes a subject to η-expansion just as any other value, and moreover it makes it possible to study terms such as $\mathsf{S}\,\mathsf{S}$. We call *pure terms* effect-free terms, i.e., values and terms of the form $\langle t \rangle$.

We distinguish a subclass of pure contexts (E) among evaluation contexts (F):

$$\begin{aligned} E &\quad ::= \quad \square \mid v\,E \mid E\,t \\ F &\quad ::= \quad \square \mid v\,F \mid F\,t \mid \langle F \rangle \end{aligned}$$

We extend the function fv to both kinds of contexts. Note that an evaluation context F is either pure or can be written $F'[\langle E' \rangle]$ for some F' and E'. Pure contexts can be captured by S, as we can see in the following rules defining the call-by-value left-to-right reduction semantics of the calculus:

$$F[(\lambda x.t)\,v] \to F[t\{v/x\}]$$

$$F[\langle E[\mathsf{S}\,v]\rangle] \to F[\langle v\,\lambda x.\langle E[x]\rangle\rangle] \text{ with } x \notin \mathsf{fv}(E)$$

$$F[\langle v\rangle] \to F[v]$$

The first rule is the usual call-by-value β-reduction. When S is applied to a value v, it captures its surrounding pure context E up to the dynamically nearest enclosing reset, and provides its term representation $\lambda x.\langle E[x]\rangle$ as an argument to v. Finally, a reset which encloses a value can be removed, since the delimited subcomputation is finished. All these reductions may occur within a metalevel context F that encodes the chosen call-by-value evaluation strategy. As in Section 3, the reduction relation $\to$ is preserved by evaluation contexts.

Example 4.1 [9]**.** Let $i \stackrel{\mathsf{def}}{=} \lambda x.x$, $\omega \stackrel{\mathsf{def}}{=} \lambda x.x\,x$, and $\Omega \stackrel{\mathsf{def}}{=} \omega\,\omega$. We present the sequence of reductions initiated by $\langle((\mathsf{S}\,\lambda k.i\,(k\,i))\,(\mathsf{S}\,\lambda k.\omega))\,\Omega\rangle$:

$$
\begin{aligned}
\langle((\mathsf{S}\,\lambda k.i\,(k\,i))\,(\mathsf{S}\,\lambda k.\omega))\,\Omega\rangle &\to & (1)\\
\langle(\lambda k.i\,(k\,i))\,(\lambda x.\langle x\,(\mathsf{S}\,\lambda k.\omega)\,\Omega\rangle)\rangle &\to & (2)\\
\langle i\,((\lambda x.\langle x\,(\mathsf{S}\,\lambda k.\omega)\,\Omega\rangle)\,i)\rangle &\to & (3)\\
\langle i\,\langle i\,(\mathsf{S}\,\lambda k.\omega)\,\Omega\rangle\rangle &\to & (4)\\
\langle i\,\langle(\lambda k.\omega)\,(\lambda x.\langle i\,x\,\Omega\rangle)\rangle\rangle &\to & (5)\\
\langle i\,\langle\omega\rangle\rangle &\to & (6)\\
\langle i\,\omega\rangle &\to & (7)\\
\langle\omega\rangle &\to & (8)\\
\omega
\end{aligned}
$$

In step (1) the pure context $(\square\,(\mathsf{S}\,\lambda k.\omega))\,(\omega\,\omega)$, is captured and reified as a term that in step (2) is substituted for k in the argument of `shift`. In step (3) the captured context is reactivated by β-reduction, and thanks to the `reset` enclosing the body of the lambda representing the captured context, it is not *merged* with the current context, but *composed* with it. As a result, the capture triggered by `shift` in step (4) leaves the outer i intact. [4] In step (5) the context captured in step (4) is discarded, again in terms of β-reduction. In steps (6) and (8) a control delimiter guarding a value is removed, whereas in step (7) a regular function application takes place. Note that even though the reduction strategy is call-by-value, some function arguments are not evaluated, like the non-terminating term Ω in this example.

Example 4.2. This example illustrates the operational behavior of S as a value:

$$
\begin{aligned}
\langle E[\mathsf{S}\,\mathsf{S}]\rangle &\to\\
\langle \mathsf{S}\,\lambda x.\langle E[x]\rangle\rangle &\to\\
\langle(\lambda x.\langle E[x]\rangle)\,(\lambda x.\langle x\rangle)\rangle &\to\\
\langle\langle E[\lambda x.\langle x\rangle]\rangle\rangle &
\end{aligned}
$$

---

[4]For this reason `shift` and `reset` are called *static* control operators, in contrast to Felleisen's `control` and `prompt` [14] that are called *dynamic* control operators [7].

In particular, if $E = \Box$, then the value of the initial term, after two additional reduction steps, is $\lambda x.\langle x \rangle$, i.e., the representation of the empty context in the calculus of delimited control.

A term t either uniquely reduces to another term, or is an eager normal form: it is either a value v, an open stuck term $F[x\,v]$, or a *control-stuck term* $E[\mathsf{S}\,v]$. The latter cannot reduce further since it lacks a reset enclosing S. In the original reduction semantics [6], derived from the higher-order evaluator implementing the denotational semantics of `shift` and `reset` [12], it was assumed that programs are evaluated in an enclosing top-level `reset`. Here, however, we consider a relaxed semantics [9] that lifts this requirement. Such a semantics corresponds to some of the existing implementations of delimited-control operators [17] that make it possible to observe control-stuck programs (raising a "missing reset" exception if one forgot the top-level reset) and it scales to control operators that can remove the enclosing delimiter, e.g., as in general calculi with multiple prompts [13, 4]. This choice does not influence the operational semantics of `shift` and `reset` in any other way.

Because `shift` can decompose contexts, we have to change the relation R^{ctx} as discussed in [9]:

$$\frac{E[x] \ \mathcal{R} \ E'[x] \qquad x \text{ fresh}}{E \ \mathcal{R}^{\mathsf{ctx}} \ E'} \qquad\qquad \frac{\langle E[x]\rangle \ \mathcal{R} \ \langle E'[x]\rangle \qquad F[x] \ \mathcal{R} \ F'[x] \qquad x \text{ fresh}}{F[\langle E\rangle] \ \mathcal{R}^{\mathsf{ctx}} \ F'[\langle E'\rangle]}$$

We also introduce a relation R^{c} to handle control-stuck terms:

$$\frac{E \ \mathcal{R}^{\mathsf{ctx}} \ E' \qquad \langle v\,x\rangle \ \mathcal{R} \ \langle w\,x\rangle \qquad x \text{ fresh}}{E[\mathsf{S}\,v] \ \mathcal{R}^{\mathsf{c}} \ E'[\mathsf{S}\,w]}$$

whereas the relation $\mathcal{R}^{\mathsf{v}}$ remains unchanged, so that it accounts for the η-law, even though the values now include S.

We can now define (extensional) normal-form bisimulation and bisimilarity for the extended calculus, again using the notion of diacritical progress.

Definition 4.3. A relation $\mathcal{R}$ diacritically progresses to $\mathcal{S}, \mathcal{T}$ written $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{S} \subseteq \mathcal{T}$, and $t \ \mathcal{R} \ s$ implies:
- if $t \rightarrow t'$, then there exists s' such that $s \rightarrow^* s'$ and $t' \ \mathcal{T} \ s'$;
- if $t = v$, then there exists w such that $s \Downarrow w$, and $v \ \mathcal{S}^{\mathsf{v}} \ w$;
- if $t = F[x\,v]$, then there exist F', w such that $s \Downarrow F'[x\,w]$ and $F[x\,v] \ \mathcal{T}^{\mathsf{o}} \ F'[x\,w]$;
- if $t = E[\mathsf{S}\,v]$, then there exist E', w such that $s \Downarrow E'[\mathsf{S}\,w]$ and $E[\mathsf{S}\,v] \ \mathcal{T}^{\mathsf{c}} \ E'[\mathsf{S}\,w]$;
- the converse of the above conditions on s.

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

Only the clause for values is passive, as in Definition 3.3.

Example 4.4. The terms $\mathsf{S}\,\mathsf{S}$ and $\mathsf{S}\,(\lambda k.k\,(\lambda x.x))$ are bisimilar since the following relation is a normal-form bisimulation:

$$\frac{t \; \mathcal{R} \; s \qquad E \; \mathcal{R}^{\mathsf{ctx}} \; E'}{E[t] \; \mathsf{pctx}(\mathcal{R}) \; E'[s]} \qquad\qquad \frac{t \; \mathcal{R} \; s \qquad \langle E \rangle \; \mathcal{R}^{\mathsf{ctx}} \; \langle E' \rangle}{\langle E[t] \rangle \; \mathsf{pctxrst}(\mathcal{R}) \; \langle E'[s] \rangle}$$

$$\frac{t \; \mathcal{R} \; s \qquad t, s \; \mathrm{pure} \qquad F[x] \; \mathcal{R} \; F'[x] \qquad x \; \mathrm{fresh}}{F[t] \; \mathsf{ectxpure}(\mathcal{R}) \; F'[s]}$$

Figure 2: Up-to techniques specific to the λ-calculus extended with `shift` and `reset`

$$\mathcal{I} \cup \{ \begin{array}{lll} (\mathsf{S}\,\mathsf{S}, & \mathsf{S}\,(\lambda k.k\,(\lambda x.x))), & (1) \\ (\langle \mathsf{S}\,z \rangle, & \langle (\lambda k.k\,(\lambda x.x))\,z \rangle), & (2) \\ (\langle z\,(\lambda x.\langle x \rangle) \rangle, & \langle z\,(\lambda x.x) \rangle), & (3) \\ ((\lambda x.\langle x \rangle)\,y, & (\lambda x.x)\,y), & (4) \\ (\langle y \rangle, & y) \} & (5) \end{array}$$

where $\mathcal{I}$ is the identity relation and x, y, and z fresh variables. In (1) we compare two control-stuck terms, so to validate the bisimulation conditions, we have to compare the two empty contexts (which are in $\mathcal{I}^{\mathsf{ctx}}$) and the arguments of `shift`. Here, extensionality plays an important role, as these arguments are of different kinds (S vs a λ-abstraction). We compare them by passing them a fresh variable z, thus we include the pair (2) in the bisimulation. The terms of (2) can be reduced to those in (3), where we compare open stuck terms, so we have to include (4) to compare the arguments of z. The terms in (4) can then be reduced to the ones in (5) which in turn reduce to identical terms.

4.2. **Up-to techniques.** The up-to techniques we consider for this calculus are the same as in Figure 1, except we replace `ectx` by three more fine-grained up-to techniques defined in Figure 2. The techniques `pctx`, `pctxrst` allow to factor out related pure contexts and pure contexts with a surrounding reset. The third one (`ectxpure`) can be used only with pure terms, but uses a naive comparison between any evaluation contexts instead of $\cdot^{\mathsf{ctx}}$. Indeed, a pure term cannot evaluate to a control-stuck term, so decomposing contexts with $\cdot^{\mathsf{ctx}}$ is not necessary. The usual reasoning up to evaluation context `ectx` can be obtained by composing these three up-to techniques.

Lemma 4.5. *If $t \; \mathcal{R} \; t'$ and $F \; \mathcal{R}^{\mathsf{ctx}} \; F'$ then $F[t] \; (\mathsf{pctx} \cup (\mathsf{ectxpure} \circ \mathsf{pctxrst}))(\mathcal{R}) \; F'[t']$.*

We do not define extra up-to techniques corresponding to the new constructs of the language, as `shift` is dealt with like variables—using `refl`, and closure w.r.t. `reset` can be deduced from `pctxrst` by taking the empty context. Defining a dedicated up-to technique for `reset` would have some merit since it could be proved strong. It is not so for `pctxrst`, as we can see in the next theorem:

Theorem 4.6. *The set $\mathfrak{F} \overset{\mathrm{def}}{=} \{\mathsf{refl}, \mathsf{lam}, \mathsf{subst}, \mathsf{pctx}, \mathsf{pctxrst}, \mathsf{ectxpure}, \mathsf{id}, \mathsf{red}\}$ is diacritically compatible, with $\mathsf{strong}(\mathfrak{F}) = \mathfrak{F} \setminus \{\mathsf{pctx}, \mathsf{pctxrst}, \mathsf{ectxpure}\}$.*

The evolution proofs are by case analysis on the possible reductions that the related terms can do. The techniques `pctx`, `pctxrst`, and `ectxpure` are not strong as they exhibit the same problematic case presented in Lemma 3.20 (for `pctx` and `ectxpure`) or a slight variant ($\langle E \rangle = \langle \square \, v \rangle$ and $t = x$ for `pctxrst`).

The main difference with the λ-calculus evolution proofs is the case analysis on the behavior of `shift`, which may perform a capture or not, depending on its surrounding evaluation context. For example, in the case of `subst`, the substituted value may be `shift`, which, when replacing x in a term $t = F_1[x\,v]$, may either produce a control-stuck term if F is pure, or otherwise reduce. The hypotheses we have on the evaluation contexts thanks to $\cdot^{\mathsf{ctx}}$ allow us to conclude in each case. Indeed, if $t\,\mathcal{R}\,s$ and $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$, then $s \Downarrow s'$ for some $s' = F_2[x\,w]$ such that $F_1\,\mathcal{T}^{\mathsf{ctx}}\,F_2$ and $v\,\mathcal{T}^{\mathsf{v}}\,w$. If F_1 is pure, then F_2 is also pure and $t\{\mathsf{S}/x\}$ and $s'\{\mathsf{S}/x\}$ are both control-stuck. Otherwise, $F_1 = F_1'[\langle E_1\rangle]$ and $F_2 = F_2'[\langle E_2\rangle]$ for some F_1', F_2', E_1, and E_2 such that $F_1'[y]\,\mathcal{T}\,F_2'[y]$ and $\langle E_1[y]\rangle\,\mathcal{T}\,\langle E_2[y]\rangle$ for a fresh y. Then $t\{\mathsf{S}/x\} \to F_1'[\langle v\,\lambda y.\langle E_1[y]\rangle\rangle]\{\mathsf{S}/x\}$ and $s'\{\mathsf{S}/x\} \to F_2'[\langle w\,\lambda y.\langle E_2[y]\rangle\rangle]\{\mathsf{S}/x\}$, and the two resulting terms can be related using mainly `ectxpure`, `lam` and `subst`.

As in Section 3, from Theorem 4.6 and Lemma 3.13 it follows that $\approx$ is a congruence, and, therefore, is sound w.r.t. the contextual equivalence of [9]; it is not complete as showed in op. cit.

Example 4.7. With these up-to techniques, we can simplify the bisimulation of Example 4.4 to just a single pair

$$\mathcal{R} \overset{\mathsf{def}}{=} \{(\mathsf{S}\,\mathsf{S}, \mathsf{S}\,(\lambda k.k\,(\lambda x.x)))\}.$$

Indeed, we have $\lambda x.\langle x\rangle\ \mathsf{lam}(\mathsf{red}(\mathsf{refl}(\mathcal{R})))\ \lambda x.x$, and $z\,\square\ \mathsf{refl}(\mathcal{R})^{\mathsf{ctx}}\ z\,\square$, so (3) is in $f(\mathcal{R})$, where $f \overset{\mathsf{def}}{=} \mathsf{pctxrst} \circ (\mathsf{refl} \cup (\mathsf{lam} \circ \mathsf{red} \circ \mathsf{refl}))$. Then, (2) is in $\mathsf{red}(f(\mathcal{R}))$, and for (1), we have also to relate $\square$ with $\square$, thus (1) is in $\mathsf{refl}(\mathcal{R}) \cup \mathsf{red}(f(\mathcal{R}))$. As a result, $\mathcal{R}$ is a bisimulation up to `red`, `refl`, `pctxrst`, and `lam`.

5. Abortive Control

In contrast with delimited-control operators, abortive control operators capture the whole surrounding context and reify it as an abortive computation. As such they are considerably more challenging as far as modular reasoning is concerned in that one has to take into account reduction of complete programs, rather than of their subexpressions. In this section we apply our technique to a calculus with `call/cc` and `abort` [15, 16], for which no sound normal-form bisimilarity has been defined so far.

5.1. **Syntax and semantics.** The syntax of the λ-calculus with abortive control operators `call/cc` (K) and `abort` (A) can be defined as the following extension of the syntax of Section 3:

$$
\begin{aligned}
t, s &\;::=\; \ldots \mid \mathsf{A}\,t \\
v, w &\;::=\; \ldots \mid \mathsf{K}
\end{aligned}
$$

with the same syntax of evaluation contexts as for the λ-calculus:

$$E ::= \square \mid v\,E \mid E\,t$$

However, the reduction rules for abortive control operators considered in this section, unlike the ones for the pure lambda calculus or delimited control operators, are not compatible with respect to evaluation contexts, i.e., $t \to t'$ does not imply $E[t] \to E[t']$. As such they are meant to describe evaluation of complete programs. In order to reflect this requirement,

we introduce a separate syntactic category of programs p, and we adjust the grammar of terms accordingly:

$$t, s \quad ::= \quad \dots \ | \ \mathsf{A} \ p$$
$$p, q \quad ::= \quad t$$

The remaining productions for terms as well as the grammars of values and evaluation contexts do not change. However, we introduce a category of *program contexts* F, comprising all evaluation contexts

$$F ::= E$$

and make it explicit in the reduction rules below that they give semantics to complete programs by manipulating program contexts. While this modification may seem cosmetic, it reflects the idea of abortive continuations representing "the rest of the computation" and it allows us to introduce, later on in this section, a minimal extension that makes it possible to define sound normal-form bisimulations for the original calculus.

The call-by-value semantics of the calculus is defined by the following rules.

$$F[(\lambda x.t) \ v] \to F[t\{v/x\}]$$

$$F[\mathsf{K} \ v] \to F[v \ \lambda y.\mathsf{A} \ F[y]] \text{ with } y \notin \mathsf{fv}(F)$$

$$F[\mathsf{A} \ p] \to p$$

When the capture operator K is applied to a value v, it makes a copy of the current program context F as a value and passes it to v. The captured context F is reified as a λ-abstraction whose body is guarded by `abort`, which when evaluated removes its enclosing context, effectively restoring F as the current program context. Hence the syntactic requirement that `abort` be applied to a program and not to an arbitrary term. The semantics of the control operators is illustrated in the following examples.

Example 5.1. Assuming that $x \notin \mathsf{fv}(F_2) \cup \mathsf{fv}(v)$, we have the following reduction sequence:

$$F_1[\mathsf{K} \ \lambda x.F_2[x \ v]] \to$$
$$F_1[(\lambda x.F_2[x \ v]) \ \lambda y.\mathsf{A} \ F_1[y]] \to$$
$$F_1[F_2[\lambda y.\mathsf{A} \ F_1[y] \ v]] \to$$
$$F_1[F_2[\mathsf{A} \ F_1[v]]] \to$$
$$F_1[v]$$

Example 5.2. This example illustrates the operational behavior of `call/cc` as a value:

$$F[\mathsf{K} \ \mathsf{K}] \to$$
$$F[\mathsf{K} \ \lambda x.\mathsf{A} \ F[x]] \to$$
$$F[(\lambda x.\mathsf{A} \ F[x]) \ \lambda x.\mathsf{A} \ F[x]] \to$$
$$F[\mathsf{A} \ F[\lambda x.\mathsf{A} \ F[x]]] \to$$
$$F[\lambda x.\mathsf{A} \ F[x]]$$

In particular, if $F = \square$, then the value of the initial term is $\lambda x.\mathsf{A}\ x$, i.e., the representation of the empty context in the calculus of abortive control.

5.2. **Normal-form bisimulation.** In the presence of abortive control operators the notion of normal form makes sense only for complete programs. Therefore, in order to define a notion of normal-form bisimulation for the calculus under consideration we need to somehow take into account how two given terms behave in any program context. In order to make such tests possible we introduce a distinct set of *context variables*, ranged over by c, to represent abstract contexts that are a dual concept to the fresh variables used for testing functional values, as argued in Remark 3.1.

We only need to extend the syntax of programs and program contexts:

$$
\begin{aligned}
p &\ ::=\ \ \dots\ |\ c[t]\\
F &\ ::=\ \ \dots\ |\ c[E]
\end{aligned}
$$

A program $c[t]$ stands for the term t plugged in a program context represented by the context variable c—we deliberately abuse the notation. Similarly, a program context $c[E]$ stands for the context E completed with a program context represented by c. We write $\mathsf{cv}(p)$ for the set of context variables of a program p.

Free variables represent unknown or abstract values in a program and the substitution $p\{v/x\}$ can be seen as an operation that replaces an abstract value x with a concrete value v. We define an analogous concretization operation for context variables that we call *context substitution*: program $p\langle F/c\rangle$ is a program p, where all occurrences of c are replaced by F. More formally, we have

$$
\begin{aligned}
c[t]\langle F/c\rangle &\ =\ F[t\langle F/c\rangle]\\
c'[t]\langle F/c\rangle &\ =\ c'[t\langle F/c\rangle] \qquad \text{when } c' \neq c
\end{aligned}
$$

and for terms the context substitution is applied recursively to its sub-terms and sub-programs. Note that in the first case the substitution plugs the term $t\langle F/c\rangle$ in the program context F. Context substitution preserves reduction.

Lemma 5.3. *If $p \to q$, then $p\langle F/c\rangle \to q\langle F/c\rangle$.*

When we substitute for a context variable, this property can be seen as a form of preservation of reduction by program contexts. For example, if $p = c[(\mathsf{K}\ \lambda x.t)]$ with $c \notin \mathsf{cv}(F)$, then $p \to^* c[t\{\lambda y.\mathsf{A}\ c[y]/x\}]$, and $p\langle F/c\rangle \to^* F[t\{\lambda y.\mathsf{A}\ F[y]/x\}]$.

Normal forms of the extended language are either values v, open stuck terms $F[x\ v]$, or *context-stuck terms* of the form $c[v]$. We define progress, bisimulation, and bisimilarity on programs of the extended language, and we define $\mathcal{R}^{\mathsf{t}}$ to apply these notions to terms. We change $\mathcal{R}^{\mathsf{v}}$ accordingly and we remind the definitions of $\mathcal{R}^{\mathsf{ctx}}$ and $\mathcal{R}^{\mathsf{o}}$.

$$
\frac{c[t]\ \mathcal{R}\ c[s] \qquad c \text{ fresh}}{t\ \mathcal{R}^{\mathsf{t}}\ s}
\qquad\qquad
\frac{v\ x\ \mathcal{R}^{\mathsf{t}}\ w\ x \qquad x \text{ fresh}}{v\ \mathcal{R}^{\mathsf{v}}\ w}
$$

$$
\frac{F[x]\ \mathcal{R}\ F'[x] \qquad x \text{ fresh}}{F\ \mathcal{R}^{\mathsf{ctx}}\ F'}
\qquad\qquad
\frac{F\ \mathcal{R}^{\mathsf{ctx}}\ F' \qquad v\ \mathcal{R}^{\mathsf{v}}\ w}{F[x\ v]\ \mathcal{R}^{\mathsf{o}}\ F'[x\ w]}
$$

Definition 5.4. A relation $\mathcal{R}$ *diacritically progresses* to $\mathcal{S}, \mathcal{T}$ written $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{S} \subseteq \mathcal{T}$, and $p \, \mathcal{R} \, q$ implies:

- if $p \rightarrow p'$, then there exists q' such that $q \rightarrow^* q'$ and $p' \, \mathcal{T} \, q'$;
- if $p = v$, then there exists w such that $q \Downarrow w$;
- if $p = F[x \, v]$, then there exist F', w such that $q \Downarrow F'[x \, w]$ and $F[x \, v] \, \mathcal{T}^\circ \, F'[x \, w]$;
- if $p = c[v]$, then there exist w such that $q \Downarrow c[w]$ and $v \, \mathcal{S}^\vee \, w$;
- the converse of the above conditions on q.

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

In the value case $p = v$, we simply ask q to evaluate to w without requiring anything of v and w; we therefore equate all values when considered as programs. A value without context variable implies that the context has been aborted, and therefore v or w cannot be applied to a testing variable. As an example, $\mathsf{A} \, v$ and $\mathsf{A} \, w$ are bisimilar for all v and w; when plugged into a context F, these terms remove their current evaluation context as soon as they are run, so there is no way for the context to pass an argument to these values.

In contrast, if $p = c[v]$, then we require that q evaluates to a context-stuck term $c[w]$ with the same context variable c, and we test the two values by applying them to an abstract value in an abstract context. Indeed, the program context represented by c can either discard the value plugged in it or it can apply it to an argument. Obviously, we cannot equate $c[v]$ and $c'[v]$ if $c \neq c'$, as c and c' stand for possibly different program contexts.

The bisimilarity presented here has a similar structure to the bisimilarity for $\lambda\mu$-calculus [29] presented in [34]. It should come as no surprise, because the programs and program contexts of the extended calculus are reminiscent of, respectively, named terms and named contexts of the $\lambda\mu$-calculus, and the context substitution coincides with some presentations of structural substitution in $\lambda\mu$-calculus [2, 34]. The resemblance is superficial though. In particular, context variables come from the 'abstract context' interpretation and are not bound by any construct. The structure of programs is also richer: not all programs have to be considered in an abstract context, so the second case of the definition 5.4 is characteristic of the calculus with `call/cc` and `abort`, and is not present in the definition of bisimulations for $\lambda\mu$-calculus.

Remark 5.5. One could introduce the notion of program and program context along with context variables in Section 3 and 4 to define the notion of normal-form bisimulation explicitly on programs of the form $c[t]$, in the spirit of Definition 5.4. However, since the constructs in these calculi do not manipulate the complete program contexts, such definitions would trivially reduce to the ones we have presented.

We use the `call/cc` axiomatization of Sabry and Felleisen [31] as a source of examples.

Example 5.6. To prove that $\lambda x.\mathsf{K} \, \lambda y.x \, y \approx^\vee \mathsf{K}$ (η_{v_2} axiom), we compare $c[(\lambda x.\mathsf{K} \, \lambda y.x \, y) \, z]$ and $c[\mathsf{K} \, z]$, and both reduce to $c[z \, \lambda x.\mathsf{A} \, c[x]]$.

Example 5.7. The $C_{current}$ axiom equates $\mathsf{K} \, \lambda x.x \, t$ and $\mathsf{K} \, \lambda x.t$ for all t. We have to relate $c[\mathsf{K} \, \lambda x.x \, t]$ and $c[\mathsf{K} \, \lambda x.t]$ for a fresh c. Reducing these programs give us respectively $c[(\lambda y.\mathsf{A} \, c[y]) \, t\{\lambda y.\mathsf{A} \, c[y]/x\}]$ and $c[t\{\lambda y.\mathsf{A} \, c[y]/x\}]$. From there, we can conclude with a tedious case analysis on the behavior of t; we need to distinguishes cases based on whether $c'[t\{\lambda y.\mathsf{A} \, c[y]/x\}]$ (where c' is fresh) evaluates to a value, an open stuck term, or an in-context value, with c' as a context variable or not. The proof is greatly simplified with up-to techniques (see Example 5.10).
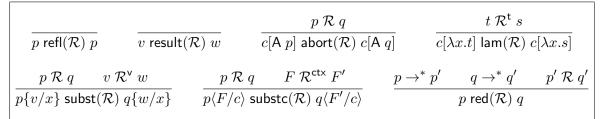
$$\frac{}{p \; \mathsf{refl}(\mathcal{R}) \; p} \qquad \frac{}{v \; \mathsf{result}(\mathcal{R}) \; w} \qquad \frac{p \; \mathcal{R} \; q}{c[\mathsf{A} \; p] \; \mathsf{abort}(\mathcal{R}) \; c[\mathsf{A} \; q]} \qquad \frac{t \; \mathcal{R}^{\mathsf{t}} \; s}{c[\lambda x.t] \; \mathsf{lam}(\mathcal{R}) \; c[\lambda x.s]}$$

$$\frac{p \; \mathcal{R} \; q \qquad v \; \mathcal{R}^{\mathsf{v}} \; w}{p\{v/x\} \; \mathsf{subst}(\mathcal{R}) \; q\{w/x\}} \qquad \frac{p \; \mathcal{R} \; q \qquad F \; \mathcal{R}^{\mathsf{ctx}} \; F'}{p\langle F/c\rangle \; \mathsf{substc}(\mathcal{R}) \; q\langle F'/c\rangle} \qquad \frac{p \to^* p' \qquad q \to^* q' \qquad p' \; \mathcal{R} \; q'}{p \; \mathsf{red}(\mathcal{R}) \; q}$$

Figure 3: Up-to techniques for the λ-calculus with `call/cc` and `abort`

5.3. **Up-to techniques.** Figure 3 presents the up-to techniques for the λ-calculus with `call/cc` and `abort`. The main difference with the previous calculi is that reasoning up to evaluation contexts ectx is replaced by substc, which is bit more general. We can deduce the former from the latter, as $t \; \mathcal{R}^{\mathsf{t}} \; s$ implies $c[t] \; \mathcal{R} \; c[s]$ for a fresh c, which in turn implies $c[t]\langle F/c\rangle \; \mathsf{substc}(\mathcal{R}) \; c[s]\langle F'/c\rangle$, i.e., $F[t] \; \mathsf{substc}(\mathcal{R}) \; F'[s]$. But we can also factorize several occurrences of a context with substc, as, e.g., $F[\mathsf{A} \; F[t]]$ can be written $c[\mathsf{A} \; c[t]]\langle F/c\rangle$ for $c \notin \mathsf{cv}(t)$.

The other novelty is result, which expresses the fact that values do not need to be tested when they are not in-context. Among all these up-to techniques, only substc is not strong, which is expected as it behaves like ectx.

Theorem 5.8. *The set* $\mathfrak{F} \stackrel{\mathsf{def}}{=} \{\mathsf{refl}, \mathsf{result}, \mathsf{abort}, \mathsf{lam}, \mathsf{subst}, \mathsf{substc}, \mathsf{red}\}$ *is diacritically compatible, with* $\mathsf{strong}(\mathfrak{F}) = \mathfrak{F} \setminus \{\mathsf{substc}\}$.

Like for the previous calculi, we show that each function in $\mathfrak{F}$ evolves towards a combination of functions in $\mathfrak{F}$, by doing a case analysis on the possible reductions of the related terms. Context variables make the treatment of the capture by `call/cc` a bit different than for `shift`, as now each capture should be written as a context substitution, to be able to conclude using substc. For example, consider as in Section 4.2 $t\{\mathsf{K}/x\} \; \mathsf{subst}(\mathcal{R})$ $s\{\mathsf{K}/x\}$ for some $t = F[x \; v]$ and $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{S}, \mathcal{T}$. There exists F' and w such that $s \Downarrow F'[x \; w]$, $F \; \mathcal{T}^{\mathsf{ctx}} \; F'$ and $v \; \mathcal{T}^{\mathsf{v}} \; w$. Then $t\{\mathsf{K}/x\} \to F[v \; \lambda y.\mathsf{A} \; F[y]]\{\mathsf{K}/x\}$ and $s\{\mathsf{K}/x\} \to^* F'[w \; \lambda y.\mathsf{A} \; F'[y]]\{\mathsf{K}/x\}$ for a fresh y; to relate the two resulting terms, we start from $v \; \mathcal{T}^{\mathsf{v}} \; w$, which by definition implies $c[v \; z] \; \mathcal{T} \; c[w \; z]$ for fresh c and z. Then because $\lambda y.\mathsf{A} \; c[y] \; \mathsf{refl}(\mathcal{T}) \; \lambda y.\mathsf{A} \; c[y]$, we can relate $c[v \; \lambda y.\mathsf{A} \; c[y]]$ and $c[w \; \lambda y.\mathsf{A} \; c[y]]$ with refl and subst, and then $c[v \; \lambda y.\mathsf{A} \; c[y]]\langle F/c\rangle\{\mathsf{K}/x\}$ and $c[w \; \lambda y.\mathsf{A} \; c[y]]\langle F'/c\rangle\{\mathsf{K}/x\}$ using also substc. The last two programs are equal to the ones we want to relate, so we can conclude.

As before, we get as a corollary of Theorem 5.8 and Lemma 3.13 that $\approx^{\mathsf{t}}$ is a congruence. The bisimilarity is also sound w.r.t. contextual equivalence when we restrict ourselves to the calculus of Section 5.1. (It can be shown that it is not complete, e.g., $\lambda x.x \; I$ and $\lambda x.(\lambda y.x \; I) \; (x \; I)$ are contextually equivalent, but not bisimilar.)

Theorem 5.9. *Let t and s be terms built from the grammar of Section 5.1. If $t \approx^{\mathsf{t}} s$, then t and s are contextually equivalent.*

Proof. Let C be a context built from the grammar of Section 5.1 which closes t and s. By Theorem 5.8 and Lemma 3.13, we have $C[t] \approx C[s]$. Therefore, by the definition of the bisimilarity, $C[t]$ terminates iff $C[s]$ terminates. $\square$

Example 5.10. We prove the $C_{current}$ axiom (Example 5.7) by showing that

$$\mathcal{R} \stackrel{\mathsf{def}}{=} \{(c[\mathsf{K} \; \lambda x.x \; t], c[\mathsf{K} \; \lambda x.t])\}$$

is a bisimulation up to substc, red, and refl. These programs reduce in two steps to respectively $c[(\lambda y.\mathsf{A}\ c[y])\ t\{\lambda y.\mathsf{A}\ c[y]/x\}]$ and $c[t\{\lambda y.\mathsf{A}\ c[y]/x\}]$. These two programs can be written $p\langle F_1/c'\rangle$ and $p\langle F_2/c'\rangle$ with c' fresh, $p = c'[t\{\lambda y.\mathsf{A}\ c[y]/x\}]$, $F_1 = c[(\lambda y.\mathsf{A}\ c[y])\ \Box]$, and $F_2 = c[\Box]$. Because $c[(\lambda y.\mathsf{A}\ c[y])\ z] \to^* c[z]$ for a fresh z, we have $F_1\ \mathsf{red}(\mathsf{refl}(\mathcal{R}))^{\mathsf{ctx}}\ F_2$, which implies $c[(\lambda y.\mathsf{A}\ c[y])\ t\{\lambda y.\mathsf{A}\ c[y]/x\}]\ \mathsf{substc}(\mathsf{red}(\mathsf{refl}(\mathcal{R})))\ c[t\{\lambda y.\mathsf{A}\ c[y]/x\}]$.

Example 5.11. The C_{tail} axiom relates $\mathsf{K}\ \lambda y.(\lambda x.t)\ s$ and $(\lambda x.\mathsf{K}\ \lambda y.t)\ s$ if $y \notin \mathsf{fv}(s)$. When surrounded with a fresh context variable c, the former reduces in two steps to $c[(\lambda x.t\{\lambda z.\mathsf{A}\ c[z]/y\})\ s]$, so we prove that

$$\mathcal{R} \stackrel{\mathsf{def}}{=} \{(c[(\lambda x.t\{\lambda z.\mathsf{A}\ c[z]/y\})\ s], c[(\lambda x.\mathsf{K}\ \lambda y.t)\ s])\}$$

is a bisimulation up to substc, red, and refl. Again, we notice that these programs can be written $c'[s]\langle F_1/c'\rangle$ and $c'[s]\langle F_2/c'\rangle$ with c' fresh, $F_1 = c[(\lambda x.t\{\lambda z.\mathsf{A}\ c[z]/y\})\ \Box]$, and $F_2 = c[(\lambda x.\mathsf{K}\ \lambda y.t)\ \Box]$. If z' is a fresh variable, then $F_1[z'] \to^* c[t\{\lambda z.\mathsf{A}\ c[z]/y\}\{z'/x\}]$ and $F_2[z'] \to^* c[t\{z'/x\}\{\lambda z.\mathsf{A}\ c[z]/y\}]$. The variables being pairwise distinct, the order of substitutions does not matter, therefore $F_1[z']$ and $F_2[z']$ reduce to identical programs. We can then conclude as in Example 5.10. Without up-to techniques, we would have to do a case analysis on the behavior of s to conclude.

The remaining axioms [31] can be proved easily in a similar way thanks to up-to techniques.

6. Conclusion

In this article we present a new approach to proving soundness of normal-form bisimilarities as well as of bisimulations up to context that allow for η-expansion. The method we develop is based on our framework [3] that generalizes the work of Madiot et al. [26, 27] in that it allows for a special treatment of some of the clauses in the definition of bisimulation. In particular, we show soundness of an extensional bisimilarity for the call-by-value λ-calculus and of the corresponding up to context, where it is critical that comparing values in a way that respects η-expansion is done passively, i.e., by requiring progress of a relation to itself. Following the same route, we obtained similar results for the extension of the call-by-value λ-calculus with delimited control, where the set of normal forms is richer, and we believe this provides an evidence for the robustness of the method. To the best of our knowledge, there has been no soundness proof of extensional normal-form up to context technique for any of the two calculi before. Furthermore, we successfully applied our technique to a theory of extensional normal-form bisimulations for the call-by-value λ-calculus with `call/cc` and `abort`, which has not existed in the literature before.

The proof method we propose should trivially apply to the existing non-extensional whnf bisimilarities [19, 20, 22] and extensional hnf bisimilarities [20, 23] for the λ-calculus and its variants. Since such whnf bisimilarities do not take into account η-expansion, their testing of values would be active and all their up-to techniques would be strong, so actually Madiot's original framework is sufficient to account for them. In the case of extensional hnf bisimilarities, normal forms are generated by the grammar:

$$
\begin{array}{lll}
h & ::= & \lambda x.h \mid n \\
n & ::= & x \mid n\ t
\end{array}
$$

and in order to account for η-expansion a λ-abstraction $\lambda x.h$ is related to a normal form n, provided h is related to $n\,x$, a freshly created normal form. Then, relating normal forms $x\,t_1\,\ldots t_m$ and $y\,s_1\,\ldots s_n$ requires that $x = y$, $m = n$, and for all i, t_i is related to s_i. Thus, in extensional hnf bisimulations the relation on normal forms provides enough information to make testing of normal forms active just like in non-extensional whnf bisimulations.

A possible direction for future research is to investigate whether our method can be adapted to enriched normal-form bisimulations that take advantage of an additional structure akin to environments used in environmental bisimulations [33]. One such theory is the complete enf bisimilarity for the $\lambda\mu\rho$-calculus of sequential control and state [34], for which the existing proof of congruence is intricately involved and no up-to context technique has been developed.

References

[1] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.

[2] Z. M. Ariola, H. Herbelin, and A. Sabry. A proof-theoretic foundation of abortive continuations. *Higher-Order and Symbolic Computation*, 20(4):403–429, 2007.

[3] A. Aristizábal, D. Biernacki, S. Lenglet, and P. Polesiuk. Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. In D. Kesner and B. Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[4] A. Aristizábal, D. Biernacki, S. Lenglet, and P. Polesiuk. Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. *Logical Methods in Computer Science*, 13(3), 2017.

[5] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North-Holland, revised edition, 1984.

[6] M. Biernacka, D. Biernacki, and O. Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science*, 1(2:5):1–39, Nov. 2005.

[7] D. Biernacki, O. Danvy, and C. Shan. On the static and dynamic extents of delimited continuations. *Science of Computer Programming*, 60(3):274–297, 2006.

[8] D. Biernacki and S. Lenglet. Normal form bisimulations for delimited-control operators. In T. Schrijvers and P. Thiemann, editors, *FLOPS'12*, number 7294 in LNCS, pages 47–61, Kobe, Japan, May 2012. Springer-Verlag.

[9] D. Biernacki, S. Lenglet, and P. Polesiuk. Bisimulations for delimited-control operators. Research report 9096, Inria Nancy – Grand Est., Sept. 2017. Avalaible at `https://hal.inria.fr/hal-01207112`.

[10] D. Biernacki, S. Lenglet, and P. Polesiuk. Proving soundness of extensional normal-form bisimilarities. In A. Silva, editor, *Proceedings of the 33th Annual Conference on Mathematical Foundations of Programming Semantics(MFPS XXXIII)*, volume 336 of *Electronic Notes in Theoretical Computer Science*, pages 41–56, Ljubljana, Slovenia, June 2017.

[11] R. S. Bird and R. Paterson. De Bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999.

[12] O. Danvy and A. Filinski. Abstracting control. In M. Wand, editor, *LFP'90*, pages 151–160, Nice, France, June 1990. ACM Press.

[13] R. K. Dybvig, S. Peyton-Jones, and A. Sabry. A monadic framework for delimited continuations. *Journal of Functional Programming*, 17(6):687–730, 2007.

[14] M. Felleisen. The theory and practice of first-class prompts. In J. Ferrante and P. Mager, editors, *POPL'88*, pages 180–190, San Diego, California, Jan. 1988. ACM Press.

[15] M. Felleisen and D. P. Friedman. Control operators, the SECD machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.

[16] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[17] A. Filinski. Representing monads. In H.-J. Boehm, editor, *POPL'94*, pages 446–457, Portland, Oregon, Jan. 1994. ACM Press.

[18] Y. Kameyama. Axioms for control operators in the CPS hierarchy. *Higher-Order and Symbolic Computation*, 20(4):339–369, 2007.

[19] S. B. Lassen. Bisimulation for pure untyped $\lambda\mu$-caluclus (extended abstract). Unpublished note, Jan. 1999.

[20] S. B. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In M. M. Stephen Brookes, Achim Jung and A. Scedrov, editors, *MFPS'99*, volume 20 of *ENTCS*, pages 346–374, New Orleans, LA, Apr. 1999. Elsevier Science.

[21] S. B. Lassen. Eager normal form bisimulation. In P. Panangaden, editor, *LICS'05*, pages 345–354, Chicago, IL, June 2005. IEEE Computer Society Press.

[22] S. B. Lassen. Normal form simulation for McCarthy's amb. In M. Escardó, A. Jung, and M. Mislove, editors, *MFPS'05*, volume 155 of *ENTCS*, pages 445–465, Birmingham, UK, May 2005. Elsevier Science Publishers.

[23] S. B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus. In R. Alur, editor, *LICS'06*, pages 297–306, Seattle, WA, Aug. 2006. IEEE Computer Society Press.

[24] S. B. Lassen and P. B. Levy. Typed normal form bisimulation. In J. Duparc and T. A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 283–297, Lausanne, Switzerland, Sept. 2007. Springer.

[25] S. B. Lassen and P. B. Levy. Typed normal form bisimulation for parametric polymorphism. In F. Pfenning, editor, *LICS'08*, pages 341–352, Pittsburgh, Pennsylvania, June 2008. IEEE Computer Society Press.

[26] J. Madiot, D. Pous, and D. Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In P. Baldan and D. Gorla, editors, *25th International Conference on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108, Rome, Italy, Sept. 2014. Springer.

[27] J.-M. Madiot. *Higher-order languages: dualities and bisimulation enhancements*. PhD thesis, Université de Lyon and Università di Bologna, 2015.

[28] J. H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, Massachusets Institute of Technology, 1968.

[29] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *LPAR'92*, number 624 in LNAI, pages 190–201, St. Petersburg, Russia, July 1992. Springer-Verlag.

[30] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, chapter 6, pages 233–289. Cambridge University Press, 2011.

[31] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. In W. Clinger, editor, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. V, No. 1, pages 288–298, San Francisco, California, June 1992. ACM Press.

[32] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. In A. Scedrov, editor, *LICS'92*, pages 102–109, Santa Cruz, California, June 1992. IEEE Computer Society.

[33] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33(1):1–69, Jan. 2011.

[34] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In M. Felleisen, editor, *POPL'07*, SIGPLAN Notices, Vol. 42, No. 1, pages 161–172, Nice, France, Jan. 2007. ACM Press.