Proving Soundness of Extensional Normal-Form Bisimilarities

Dariusz Biernacki, Sergueï Lenglet, Piotr Polesiuk

Proving Soundness of Extensional Normal-Form Bisimilarities [1]

Dariusz Biernacki[a,2] Sergueï Lenglet[b,3,4] Piotr Polesiuk[a,5]

[a] *Institute of Computer Science, University of Wrocław, Wrocław, Poland*

[b] *Loria, Université de Lorraine, Nancy, France*

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. Besides, they allow 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 bisimulation up to context are sound. We illustrate our technique with the call-by-value λ-calculus, before applying it to a call-by-value λ-calculus with the delimited-control operators `shift` and `reset`. In both cases, there was previously no sound bisimulation up to context validating the η-law. Our results have been formalized in the Coq proof assistant.

Keywords: lambda calculus, normal-form bisimulations, eta-expansion, congruence, bisimulation up to context, control operators

1 Introduction

In formal languages inspired by the λ-calculus, the behavioral equivalence of choice is usually formulated as a Morris-style contextual equivalence [18]: 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*.

[1] This work was partially supported by PHC Polonium and National Science Centre, Poland, grant no. 2014/15/B/ST6/00619.

[2] Email: dabi@cs.uni.wroc.pl

[3] Email: serguei.lenglet@univ-lorraine.fr

[4] The author carried out this work while at Irisa, Université de Rennes 1, thanks to a CNRS grant.

[5] Email: ppolesiuk@cs.uni.wroc.pl

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Different styles of bisimilarities have been defined for the λ-calculus, including *applicative bisimilarity* [1], *normal-form bisimilarity* [11] (originally called *open bisimilarity* in [20]), and *environmental bisimilarity* [21]. 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* [19], relations which simplify equivalence proofs of terms by having less requirements than regular bisimilarities. For example, bisimulation 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*,[6] 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).

In [16], Madiot et al. 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 bisimulation up to context is sound directly implies that the corresponding bisimilarity is a congruence, which is the main property needed for proving its soundness. In [16], the method is applied 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 [2], 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 [16,2] have been used in the λ-calculus only for environmental bisimilarities. In this paper, we show that the framework of [2] 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

[6] Lassen uses the term *bisimilarity up to* η [10] 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.

a corresponding bisimulation up to context, have already been proved sound [11], 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` [6], for which there has been no sound bisimulation up to context validating the η-law either, and we show that our method applies seamlessly in that setting as well. Our results have been formalized in the Coq proof assistant, thus increasing the confidence in proofs that can be quite meticulous.

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 a λ-calculus with `shift` and `reset` in Section 4. We conclude in Section 5, and the Coq developments are available at `http://www.ii.uni.wroc.pl/~ppolesiuk/apnfbisim`.

2 Related Work

Normal-form bisimilarity has been first introduced in [20], where it was named open bisimilarity, and has then been defined for many variants of the λ-calculus, considering η-expansion [10,11,13,22,14,15,4,5] or not [9,12]. In this section we focus on the articles treating the η-law, and in particular on the congruence and soundness proofs presented therein.

In [10], 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 [9] (a calculus with continuations), an extensional hnf bisimilarity for the call-by-name λ-calculus with pairs [13], and a whnf bisimilarity for a call-by-name λ-calculus with McCarthy's ambiguous choice (`amb`) operator [12]. These papers do not define any corresponding bisimulation up to context.

Lassen uses another proof technique in [11], where he defines an *eager normal form (enf) bisimilarity* and an *enf bisimilarity up to* η.[7] Lassen shows that the bisimilarities correspond to Böhm trees equivalence (up to η) after a continuation-

[7] 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.

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 [13] 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 [11], 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 [10], 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 3.11.

In [22], the authors 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 [10], 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.

In [14,15], the authors define a normal-form bisimilarity for a CPS calculus called JWA equipped with a rich type system (including product, sum, recursive types, and [15] 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 [4], 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 [10], but are incorrect: one case in the induction, that turns out to be problematic, has been forgotten. In [5] we fix the congruence proof of the extensional bisimilarity, by doing a nested induction on a different notion of closure than in [10]. 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 [5].

To summarize:

- The soundness proofs for extensional hnf bisimilarities are uniformly done using a nested induction proof method [10,13]. 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 [11,13], or other ad-hoc arguments [22,14,15,5] which do not carry over to a soundness proof for a bisimulation up to context.
- The only claims about congruence of an extensional enf bisimilarity as well as soundness of the corresponding bisimulation up to context using a nested induc-

tion proof are either wrong [4] or are not substantiated by a presentation of the actual proof [11]. 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 two calculi: the plain λ-calculus and its extension with delimited 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

In this section we introduce a new approach to normal-form bisimulations that is based on the framework we developed in [2]. 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:

$$t, s ::= v \mid t\ s \qquad v, w ::= x \mid \lambda x.t \qquad E ::= \Box \mid E\ t \mid v\ E$$

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

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

We write $\to^*$ for the reflexive and transitive closure of $\to$, and $t \Downarrow s$ if $t \to^* s$ 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 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:

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

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

Definition 3.1 A relation $\mathcal{R}$ progresses to $\mathcal{S}$ if $t\ \mathcal{R}\ s$ implies:

- if $t \to t'$, then there exists s' such that $s \to^* s'$ and $t'\ \mathcal{S}\ s'$;
- if $t = v$, then there exists w such that $s \Downarrow w$, and $v\ \mathcal{S}^{\mathsf{v}}\ w$;
- if $t = E[x\ v]$, then there exist E', w such that $s \Downarrow E'[x\ w]$ and $E[x\ v]\ \mathcal{S}^{\circ}\ E'[x\ w]$;
- 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 in [11], as we believe small-step is more flexible, since we can recover a big-step reasoning with bisimulation up to reduction (Section 3.3). In usual definitions [11,22,5], 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 definition of bisimulation 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 as explained after Definition 3.2.

We now recast the definition of normal-form bisimilarity in the framework of [2], which is itself an extension of a work by Madiot et al. [16,17]. The goal is to factorize the congruence proof of the bisimilarity with the soundness proofs of the up-to techniques. The novelty in [2] 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 3.2 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 $t\ \mathcal{R}\ s$ implies:

- if $t \to t'$, then there exists s' such that $s \to^* s'$ 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 $t = E[x\ v]$, then there exist E', w such that $s \Downarrow E'[x\ w]$ and $E[x\ v]\ \mathcal{T}^{\circ}\ E'[x\ w]$;
- the converse of the above conditions on s.

An 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 3.2 and 3.1 is only in the clause for values, where we progress towards a different relation than in the other clauses of Definition 3.2. 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 definitions generate the same bisimilarity. However, we prevent some up-to techniques

from being applied in a passive clause. In particular, up to context is not allowed, as explained in Section 3.3, meaning that we cannot deduce that $v\,x$ and $w\,x$ are related up to context just because v and w are related. In contrast, we allow any up-to techniques when we test a value in the open stuck term case, since we cannot deduce from $E[x\,v]$ related to $E'[x\,w]$ that $v\,y$ and $w\,y$ are related up to context.

Example 3.3 Let $\theta \stackrel{\text{def}}{=} \lambda zx.x\;\lambda y.z\;z\;x\;y$ and $\mathsf{fix}(v) \stackrel{\text{def}}{=} \lambda x.\theta\;\theta\;v\;x$ for a given v; note that $\mathsf{fix}(v)\;x \to^* v\;\mathsf{fix}(v)\;x$. Wadsworth's infinite η-expansion combinator [3] can be defined as $\mathsf{J} \stackrel{\text{def}}{=} \mathsf{fix}(\lambda fxy.x\;(f\;y))$. Let $\mathcal{I} \stackrel{\text{def}}{=} \{(t,t) \mid t \text{ any term}\}$ be the identity bisimulation. We prove that $\lambda x.x \approx \mathsf{J}$, by showing that

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

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 $\Box$ and $\Box$ (with $\mathcal{I}$), and z with $\lambda x.z\;(\mathsf{J}\;x)$, but these terms are already in $\mathcal{R}$. As usual, the quite lengthy definition of $\mathcal{R}$ can be simplified with up-to techniques (see Example 3.13). $\Box$

3.2 Up-to techniques, general definitions

We recall here the main definitions we use from [2]. 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. An up-to technique (resp. strong up-to technique) is a function f such that $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, f(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow f(\mathcal{R}), f(\mathcal{R})$) implies $\mathcal{R} \subseteq \approx$. Proving that a given f is an up-to technique is difficult with this definition, so following [19,16], we rely on a notion of *compatibility* instead, which gives sufficient conditions for f to be an up-to technique.

We first define some auxiliary notions and notations. We write $f \subseteq g$ if $f(\mathcal{R}) \subseteq g(\mathcal{R})$ for all $\mathcal{R}$. We define $f \cup g$ argument-wise, i.e., $(f \cup g)(\mathcal{R}) = f(\mathcal{R}) \cup g(\mathcal{R})$, and given a set $\mathfrak{F}$ of functions, we also write $\mathfrak{F}$ for the function defined as $\bigcup_{f\in\mathfrak{F}} f$. 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 $f \cup \mathsf{id}$. 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, so they are trivially continuous. Continuous functions are interesting because of their properties: [8]

Lemma 3.4 *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 $f \circ \widehat{f}^\omega \subseteq \widehat{f}^\omega$.

[8] Our formalization revealed an error in previous works [2,17] which use f instead of $\widehat{f}$ in the last property of Lemma 3.4 (expressing idempotence of $\widehat{f}^\omega$)—id has to be factored in for the property to hold.

Definition 3.5 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 $f(\mathcal{R}) \rightarrowtail\!\!\!\!\twoheadrightarrow g(\mathcal{S}), h(\mathcal{T})$.

Evolution can be seen as a notion of progress for functions on relations. Note that strong evolution does not put any condition on how $\mathcal{R}$ progresses, while regular evolution is more restricted, as it requires a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$.

Definition 3.6 A set $\mathfrak{F}$ of continuous functions is *diacritically compatible* if there exists $\mathfrak{S}$ such that $\mathfrak{S} \subseteq \mathfrak{F}$ and

- for all $f \in \mathfrak{S}$, we have $f \rightsquigarrow_{\mathsf{s}} \widehat{\mathfrak{S}}^{\omega}, \widehat{\mathfrak{F}}^{\omega}$;
- for all $f \in \mathfrak{F}$, we have $f \rightsquigarrow \widehat{\mathfrak{S}}^{\omega} \circ \widehat{\mathfrak{F}} \circ \widehat{\mathfrak{S}}^{\omega}, \widehat{\mathfrak{F}}^{\omega}$.

In words, a function is in a compatible set $\mathfrak{F}$ if it evolves towards a combination of functions in $\mathfrak{F}$. The (possibly empty) subset $\mathfrak{S}$ intuitively represents the strong up-to techniques of $\mathfrak{F}$. Any combination of functions can be used in an active clause. In a passive one, only strong functions can be used, except in the second case, where we progress from $f(\mathcal{R})$, with f not strong. In that case, it is expected to progress towards a combination that 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 the definition, then $\mathfrak{S}_1 \cup \mathfrak{S}_2$ also does, so there exists the largest subset of $\mathfrak{F}$ which satisfies the conditions, written $\mathsf{strong}(\mathfrak{F})$.

Lemma 3.7 *Let $\mathfrak{F}$ be a diacritically compatible set.*

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

The proof takes advantage of Lemma 3.4. In practice, proving that f is in a compatible set $\mathfrak{F}$ is easier than proving it is an up-to technique. Besides, if we prove that a bisimulation up to context is compatible, then we get for free that $\approx$ is a congruence thanks to the last property of Lemma 3.7.

3.3 Up-to techniques for normal-form bisimilarity

Figure 1 presents the up-to techniques we define for the λ-calculus. Combined altogether, they define a closure as in the nested induction proof method [10,13]; we use a more fine-grained approach to distinguish between strong and regular up-to techniques. As in [10,13,5], we use the substitutive closure subst. We also rely on the closure by evaluation contexts ectx, which is less common, but already used in [5]. The closure ectx is not the same as bisimulation up to context, since we can factor out different contexts, as long as they are related when we plug a fresh variable inside them. The technique red, used in the compatibility proofs, is the classic bisimulation up to reduction, which allows terms to reduce before being related.

Theorem 3.8 *The set $\mathfrak{F} \stackrel{\mathsf{def}}{=} \{\mathsf{refl}, \mathsf{app}, \mathsf{lam}, \mathsf{subst}, \mathsf{ectx}, \mathsf{id}, \mathsf{red}\}$ is diacritically compatible, with $\mathsf{strong}(\mathfrak{F}) = \{\mathsf{refl}, \mathsf{lam}, \mathsf{subst}, \mathsf{id}, \mathsf{red}\}$.*

$$\frac{}{t\ \mathsf{refl}(\mathcal{R})\ t} \qquad \frac{t\ \mathcal{R}\ t' \qquad s\ \mathcal{R}\ s'}{t\ s\ \mathsf{app}(\mathcal{R})\ t'\ s'} \qquad \frac{t\ \mathcal{R}\ s}{\lambda x.t\ \mathsf{lam}(\mathcal{R})\ \lambda x.s}$$

$$\frac{t\ \mathcal{R}\ s \qquad v\ \mathcal{R}^{\mathsf{v}}\ w}{t\{v/x\}\ \mathsf{subst}(\mathcal{R})\ s\{w/x\}} \qquad \frac{t\ \mathcal{R}\ s \qquad E\ \mathcal{R}^{\mathsf{ctx}}\ E'}{E[t]\ \mathsf{ectx}(\mathcal{R})\ E'[s]}$$

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

Fig. 1: Up-to techniques for the λ-calculus

The complete proof of Theorem 3.8 can be found in the Coq formalization. We sketch some of the compatibility proofs to show how proofs are done in our framework, in particular the crucial case of app, where we need the distinction between active and passive tests. We compare ourselves to the proof technique of [10], which would do an induction on the definition of the closure using Definition 3.1. We do not need an induction on the number of evaluation steps for our small-step definition, but a nested induction proof for a big-step relation would exhibit the same issues. The strong up-to techniques lam, refl, and red are easy to deal with; we detail the proof for lam.

Lemma 3.9 $\mathsf{lam} \rightsquigarrow_{\mathsf{s}} \mathsf{lam} \cup \mathsf{red}, \mathsf{lam} \cup \mathsf{red}$.

Proof. Let $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$; we want to prove that $\mathsf{lam}(\mathcal{R}) \rightarrowtail\!\!\!\!\twoheadrightarrow \mathsf{lam}(\mathcal{S}) \cup \mathsf{red}(\mathcal{S}), \mathsf{lam}(\mathcal{T}) \cup \mathsf{red}(\mathcal{T})$. The inclusions $\mathsf{lam}(\mathcal{R}) \subseteq \mathsf{lam}(\mathcal{S}) \cup \mathsf{red}(\mathcal{S})$ and $\mathsf{lam}(\mathcal{S}) \cup \mathsf{red}(\mathcal{S}) \subseteq \mathsf{lam}(\mathcal{T}) \cup \mathsf{red}(\mathcal{T})$ hold because $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{S} \subseteq \mathcal{T}$ (by definition of $\rightarrowtail\!\!\!\!\twoheadrightarrow$) and the functions are monotone. Next, 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.□

We now sketch the proof for subst. The proof method is by case analysis on the related terms, similar to what one would do in the proof by induction of [10].

Lemma 3.10 $\mathsf{subst} \rightsquigarrow_{\mathsf{s}} \mathsf{subst}, (\mathsf{id} \cup \mathsf{ectx}) \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})$.

Proof (Sketch) Let $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, and $t\{v/x\}\ \mathsf{subst}(\mathcal{R})\ s\{w/x\}$ such that $t\ \mathcal{R}\ s$ and $v\ \mathcal{R}^{\mathsf{v}}\ w$. 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 $t\ \mathcal{R}^{\mathsf{v}}\ s'$. But then $t\{v/x\}$ and $s'\{w/x\}$ are also values, and then we can prove that $t\{v/x\}\ \mathsf{subst}(\mathcal{S})^{\mathsf{v}}\ s'\{w/x\}$ 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 $t'\{v/x\}\ \mathsf{subst}(\mathcal{T})\ s'\{w/x\}$.

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 $\mathsf{subst}(\mathcal{T})^{\circ}$. 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 $v\ \mathcal{R}^{\mathsf{v}}\ w$ 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 $E\{v/x\}[x']$ $\mathsf{subst}(\mathcal{T})$ $E\{w/x\}[x']$ for a fresh x', but also $t'\{v'\{v/x\}/z\}$ $\mathsf{subst}(\mathsf{subst}(\mathcal{T}))$ $s''\{w'\{w/x\}/z\}$, so after plugging, we obtain terms in $\mathsf{ectx} \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})(\mathcal{T})$.

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 $\mathsf{ectx} \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})(\mathcal{T})$ and whose arguments are related by $\mathsf{subst}(\mathsf{subst}(\mathcal{T}))$. □

The proof for subst does not require the clause for values to be passive, and is thus similar to the corresponding case of an induction proof [10]. In contrast, we need testing values to be passive when dealing with app and ectx; we present the problematic subcase in the proof below. We do not know how to make this subcase go through in a proof as in [10].

Lemma 3.11 $\mathsf{app} \rightsquigarrow \mathsf{app}, \mathsf{app} \cup \mathsf{subst} \cup \mathsf{ectx} \cup (\mathsf{id} \cup \mathsf{ectx}) \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})$.

Proof (Sketch) Let $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{S}$, and $t_1\ s_1\ \mathsf{app}(\mathcal{R})\ t_2\ s_2$ such that $t_1\ \mathcal{R}\ t_2$ and $s_1\ \mathcal{R}\ s_2$. We proceed by case analysis on t_1 and s_1. Most cases are straightforward; the problematic case is when t_1 is a variable x and s_1 a value w_1. Because $t_1\ \mathcal{R}\ t_2$ and $s_1\ \mathcal{R}\ s_2$, there exists v_2 and w_2 such that $t_2 \Downarrow v_2$, $s_2 \Downarrow w_2$, $x\ \mathcal{R}^{\mathsf{v}}\ v_2$, and $w_1\ \mathcal{R}^{\mathsf{v}}\ w_2$. From $x\ \mathcal{R}^{\mathsf{v}}\ v_2$, we have $x\ y\ \mathcal{R}\ v_2\ y$ for a fresh y, and therefore $x\ w_1\ \mathsf{subst}(\mathcal{R})\ v_2\ w_2$. We can conclude using Lemma 3.10: there exists an open stuck term s' such that $t_2\ s_2 \to^* v_2\ s_2 \to^* v_2\ w_2 \to^* s'$ and $x\ v_2\ ((\mathsf{id} \cup \mathsf{ectx}) \circ \mathsf{subst} \circ (\mathsf{id} \cup \mathsf{subst})(\mathcal{S}))^{\mathsf{o}}\ s'$.

In an induction proof with Definition 3.1, we would have in that case $x\ y\ \mathcal{S}\ v_2\ y$ and $w_1\ \mathcal{S}^{\mathsf{v}}\ w_2$ instead of $\mathcal{R}$. We do not see how to go further in the case w_1 is a λ-abstraction $\lambda x.t$: we have to prove that $t\{w_2/x\}$ evaluates to an open stuck term, but we do not have any progress hypothesis about $\mathcal{S}$. □

The technique ectx exhibits a similar problematic subcase, when $E = \Box\ v$ and $t = x$. We obtain the following corollary that follows from Theorem 3.8 (refl, lam and app are compatible) and Lemma 3.7 (the third item).

Corollary 3.12 $\approx$ *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]. However, as proved in [11], $\approx$ is not complete w.r.t. contextual equivalence.

Example 3.13 We can simplify the definition of $\mathcal{R}$ in Example 3.3 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. To illustrate how bisimulation up to context can help, we define $v \stackrel{\mathsf{def}}{=} \mathsf{fix}(\lambda zxy.z\,x)$ and $w \stackrel{\mathsf{def}}{=} \mathsf{fix}(\lambda zxy.z\,(\mathsf{J}\,x))$ and prove that these values are bisimilar by showing that $\mathcal{R}' \stackrel{\mathsf{def}}{=} \{(v, w), (v\,x, w\,x), ((\lambda y.v\,x)\,z, (\lambda y.w\,(\mathsf{J}\,x))\,z) \mid x, z \text{ fresh}\} \cup \mathcal{R}$ is a bisimulation up to ectx, refl, and red. Indeed, we have $vx \to^* (\lambda zxy.zx)vx \to^* \lambda y.vx$ and $w\ x \to^* (\lambda zxy.z\ (\mathsf{J}\ x))\ w\ x \to^* \lambda y.w\ (\mathsf{J}\ x)$. To relate the two resulting values, we compare $(\lambda y.v\ x)\ z$ and $(\lambda y.w\ (\mathsf{J}\ x))\ z$ for a fresh z. These terms reduce to respectively $v\ x$ and $w\ (\mathsf{J}\,x)$, which are in $\mathsf{ectx}(\mathcal{R}')$, because $v\ y\ \mathcal{R}'\ w\ y$ for a fresh y and $x\ \mathcal{R}\ \mathsf{J}\ x$. Without ectx, we would have to reduce these terms further and continue the bisimulation game. Note that we use ectx after a reduction step, i.e.,

in an active clause. We also have $(\lambda y.v\ x)\ z\ \mathsf{red}(\mathsf{ectx}(\mathcal{R}))\ (\lambda y.w\ (\mathsf{J}\ x))\ z$, but we cannot conclude with this, as we would use ectx in the passive clause for values. □

4 Delimited-control operators

In this section we turn to the call-by-value λ-calculus extended with `shift` and `reset` [6,5]. We show that the results of Section 3 seamlessly carry over to this calculus, thus demonstrating the robustness of the approach, but also improving on the previous results on extensional normal-form bisimulations for this calculus [5].

4.1 Syntax, semantics, and normal-form bisimulations

We extend the grammar of terms and values given in Section 3 as follows:

$$t, s ::= \ldots \mid \langle t \rangle \qquad v, w ::= \ldots \mid \mathcal{S}$$

where $\langle\cdot\rangle$ is the control delimiter `reset` and $\mathcal{S}$ is the delimited-control operator `shift`. Usually, `shift` is presented as a binder $\mathcal{S}x.t$ [6,5] or as a special form $\mathcal{S}\,t$ [7], but here we choose a more liberal syntax treating `shift` as a value (as, e.g., in [8]). 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 $\mathcal{S}\,\mathcal{S}$. 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):

$$E ::= \Box \mid v\ E \mid E\ t \qquad F ::= \Box \mid v\ F \mid F\ t \mid \langle F \rangle$$

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 $\mathcal{S}$, as we can see in the following rules defining the call-by-value left-to-right reduction semantics of the calculus:

$$\begin{aligned} F[(\lambda x.t)\ v] &\to F[t\{v/x\}] \\ F[\langle E[\mathcal{S}\ v] \rangle] &\to F[\langle v\ \lambda x.\langle E[x] \rangle \rangle] \text{ with } x \notin \mathsf{fv}(E) \\ F[\langle v \rangle] &\to F[v] \end{aligned}$$

The first rule is the usual call-by-value β-reduction. When $\mathcal{S}$ 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 This example illustrates the operational behavior of $\mathcal{S}$ as a value:

$$\langle E[\mathcal{S}\ \mathcal{S}] \rangle \to \langle \mathcal{S}\ \lambda x.\langle E[x] \rangle \rangle \to \langle (\lambda x.\langle E[x] \rangle)\ (\lambda x.\langle x \rangle) \rangle \to \langle \langle E[\lambda x.\langle x \rangle] \rangle \rangle$$

In particular, if $E = \Box$, then the value of the initial term is $\lambda x.\langle x\rangle$, i.e., the representation of the empty context. $\Box$

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* $E[\mathcal{S}\ v]$. The latter cannot reduce further since it lacks a reset enclosing $\mathcal{S}$. Because `shift` can decompose contexts, we have to change the relation R^{ctx} as discussed in [5]:

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

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

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

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

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

Definition 4.2 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 $t\ \mathcal{R}\ s$ implies:

- if $t \to t'$, then there exists s' such that $s \to^* s'$ 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 $F[x\ v]\ \mathcal{T}^{\circ}\ F'[x\ w]$;
- if $t = E[\mathcal{S}\ v]$, then there exist E', w such that $s \Downarrow E'[\mathcal{S}\ w]$ and $E[\mathcal{S}\ v]\ \mathcal{T}^{\mathsf{c}}\ E'[\mathcal{S}\ w]$;
- 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.

Note that only the clause for values is passive, as in Definition 3.2.

Example 4.3 The terms $\mathcal{S}\ \mathcal{S}$ and $\mathcal{S}\ (\lambda k.k\ (\lambda x.x))$ are bisimilar since the following relation is a normal-form bisimulation:

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

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,

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

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

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

extensionality plays an important role, as these arguments are of different kinds ($\mathcal{S}$ 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 bisimulation up to evaluation context ectx can be obtained by composing these three up-to techniques.

Lemma 4.4 *If* $t\ \mathcal{R}\ t'$ *and* $F\ \mathcal{R}^{\mathsf{ctx}}\ F'$ *then* $F[t]\ (\mathsf{pctx} \cup (\mathsf{ectxpure} \circ \mathsf{pctxrst}))(\mathcal{R})\ F'[t']$.

Note that we do not define extra up-to techniques corresponding to the new constructs of the language: `shift` is dealt with like variables—using refl, and congruence 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 4.5 *The set* $\mathfrak{F} \stackrel{\mathsf{def}}{=} \{\mathsf{refl}, \mathsf{app}, \mathsf{lam}, \mathsf{subst}, \mathsf{pctx}, \mathsf{pctxrst}, \mathsf{ectxpure}, \mathsf{id}, \mathsf{red}\}$ *is diacritically compatible, with* $\mathsf{strong}(\mathfrak{F}) = \{\mathsf{refl}, \mathsf{lam}, \mathsf{subst}, \mathsf{id}, \mathsf{red}\}$.

The evolution proofs are as in the pure λ-calculus, by case analysis on the possible reductions that the related terms can do. The techniques app, pctx, pctxrst, and ectxpure are not strong as they exhibit the same problematic case presented in Lemma 3.11 (for app, pctx, and ectxpure) or a slight variant ($\langle E\rangle = \langle \Box\ v\rangle$ and $t = x$ for pctxrst). As in Section 3, from Theorem 4.5 and Lemma 3.7 it follows that $\approx$ is a congruence, and, therefore, is sound w.r.t. the contextual equivalence of [5]; it is not complete as showed in op. cit.

Example 4.6 With these up-to techniques, we can simplify the bisimulation of Example 4.3 to just a single pair $\mathcal{R} \stackrel{\mathsf{def}}{=} \{(\mathcal{S}\ \mathcal{S}, \mathcal{S}\ (\lambda k.k\ (\lambda x.x)))\}$. Indeed, we have

$\lambda x.\langle x\rangle$ $\mathsf{lam}(\mathsf{red}(\mathsf{refl}(\mathcal{R})))$ $\lambda x.x$, and $z\,\Box$ $\mathsf{refl}(\mathcal{R})^{\mathsf{ctx}}$ $z\,\Box$, so (3) is in $f(\mathcal{R})$, where $f \stackrel{\mathsf{def}}{=} \mathsf{pctxrst} \circ (\mathsf{refl} \cup (\mathsf{lam} \circ \mathsf{red} \circ \mathsf{refl}))$. Then, (2) is in $\mathsf{red}(f(\mathcal{R}))$, and for (1), we have also to relate $\Box$ with $\Box$, 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 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 [2] that generalizes the work of Madiot et al. [16,17] 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 bisimulation 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 bisimulation up to context for any of the two calculi before.

The proof method we propose should trivially apply to the existing non-extensional whnf bisimilarities [9,10,12] and extensional hnf bisimilarities [10,13] for the λ-calculus and its variants. Since 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:

$$h ::= \lambda x.h \mid n \qquad n ::= x \mid n\,t$$

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. Thus, in extensional enf bisimulations the relation on normal forms provides enough information to make testing of normal forms active just like in whnf bisimulations.

A future work that seems a worthwhile task would be to tackle the complete enf bisimilarity for the $\lambda\mu\rho$-calculus of sequential control and state [22]. It would be interesting to investigate whether our method can be adapted to the 'relation-set' structure of bisimulations that capture the behavior of mutable references.

Acknowledgement

We thank the anonymous reviewers for their helpful comments on the presentation of this work.

References

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

[2] 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, Porto, Portugal, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] 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.

[4] D. Biernacki and S. Lenglet. Normal form bisimulations for delimited-control operators. In T. Schrijvers and P. Thiemann, editors, *Functional and Logic Programming, 13th International Symposium (FLOPS'12)*, volume 7294 of *Lecture Notes in Computer Science*, pages 47–61, Kobe, Japan, May 2012. Springer.

[5] D. Biernacki, S. Lenglet, and P. Polesiuk. Bisimulations for delimited-control operators. Avalaible at https://hal.inria.fr/hal-01207112, 2015. Accepted for publication in *Information and Computation*.

[6] O. Danvy and A. Filinski. Abstracting control. In M. Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

[7] A. Filinski. Representing monads. In H.-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 446–457, Portland, Oregon, Jan. 1994. ACM Press.

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

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

[10] 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, *Proceedings of the 15th Annual Conference on Mathematical Foundations of Programming Semantics*, volume 20 of *Electronic Notes in Theoretical Computer Science*, pages 346–374, New Orleans, LA, Apr. 1999.

[11] S. B. Lassen. Eager normal form bisimulation. In P. Panangaden, editor, *Proceedings of the 20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 345–354, Chicago, IL, June 2005. IEEE Computer Society Press.

[12] S. B. Lassen. Normal form simulation for McCarthy's amb. In M. Escardó, A. Jung, and M. Mislove, editors, *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI)*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 445–465, Birmingham, UK, May 2005.

[13] S. B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus. In R. Alur, editor, *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 297–306, Seattle, WA, USA, Aug. 2006. IEEE Computer Society Press.

[14] S. B. Lassen and P. B. Levy. Typed normal form bisimulation. In J. Duparc and T. A. Henzinger, editors, *Computer Science Logic, CSL'07*, volume 4646 of *Lecture Notes in Computer Science*, pages 283–297, Lausanne, Switzerland, Sept. 2007. Springer.

[15] S. B. Lassen and P. B. Levy. Typed normal form bisimulation for parametric polymorphism. In F. Pfenning, editor, *Proceedings of the 23rd IEEE Symposium on Logic in Computer Science (LICS 2008)*, pages 341–352, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

[16] 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.

[17] J.-M. Madiot. *Higher-Order Languages: Dualities and Bisimulation Enhancements*. PhD thesis, Université de Lyon and Università di Bologna, 2015.

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

[19] 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.

[20] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. In A. Scedrov, editor, *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science (LICS'92)*, pages 102–109, Santa Cruz, California, June 1992. IEEE Computer Society.

[21] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In J. Marcinkowski, editor, *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 293–302, Wroclaw, Poland, July 2007. IEEE Computer Society Press.

[22] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In M. Felleisen, editor, *Proceedings of the 34th Annual ACM Symposium on Principles of Programming Languages*, pages 161–172, Nice, France, Jan. 2007. ACM Press.